ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Цифрова система електронного документообігу з управлінням документами та їх обробкою на основі Node.js»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Дарина БОНДАРЕНКО
(підпис)

Виконала: здобувачка вищої освіти групи ТЦР-42

Дарина БОНДАРЕНКО

Керівник: Ірина ГНІДЕНКО
Старший викладач кафедри

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бондаренко Дарині Вадимівні

1. Тема кваліфікаційної роботи: «Цифрова система електронного документообігу з управлінням документами та їх обробкою на основі Node.js »
керівник кваліфікаційної роботи старший викладач кафедри Ірина ГНІДЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20»лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «18» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про принципи організації електронного документообігу, методи побудови клієнт-серверних веб-застосунків, опис технологій Node.js, Express.js та PostgreSQL, технічна документація з використання REST API, JWT-автентифікації та засобів обробки й зберігання електронних документів, матеріали щодо проєктування інтерфейсів користувача та забезпечення безпеки веб-застосунків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз існуючих програмних рішень для електронного документообігу та управління документами в організаціях.
2. Формулювання вимог та постановка задачі розробки веб-додатку для електронного документообігу.
3. Проєктування архітектури системи електронного документообігу з підтримкою авторизації, пошуку та електронного підпису документів.
4. Реалізація серверної та клієнтської частини веб-додатку з використанням REST API та окремого фронтенду.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Архітектурна система компонентів
5. Алгоритм роботи
6. Екранні форми.
7. Апробація результатів дослідження

6. Дата видачі завдання «20» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	20.02-24.02.2026	виконано
2	Аналіз та дослідження існуючих аналогів	25.02-01.03.2026	виконано
3	Огляд існуючих алгоритмів обробки тексту та методів і технологій побудови тематичного словника	02.03-08.03.2026	виконано
4	Проектування застосунку для автоматизованої побудови тематичного словника навчальної дисципліни	09.03-22.03.2026	виконано
5	Програмна реалізація застосунку	23.03-05.04.2026	виконано
6	Тестування застосунку	06.05-12.04.2026	виконано
7	Оформлення роботи: вступ, висновки, реферат	13.04-26.04.2026	виконано

8	Розробка демонстраційних матеріалів	27.04-01.05.2026	виконано
9	Попередній захист роботи	11.05-22.05.2026	виконано

Здобувачка вищої освіти _____ Дарина БОНДАРЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ГНІДЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 72 стор., 2 табл., 3 рис., 25 джерел.

Мета роботи – розробка веб-застосунку електронного документообігу для підвищення ефективності створення, обробки, зберігання та управління документами, що забезпечує автоматизацію документообігу та зручний доступ до інформації.

Об'єкт дослідження – процес організації та управління електронним документообігом у інформаційних системах.

Предмет дослідження – веб-застосунок для автоматизації процесів створення, обробки та контролю електронних документів.

Короткий зміст роботи: у роботі проаналізовано сучасні програмні рішення у сфері електронного документообігу, визначено їх переваги та недоліки. На основі проведеного аналізу сформульовано функціональні та нефункціональні вимоги до системи. Розроблено архітектуру веб-застосунку на основі клієнт-серверного підходу, що забезпечує розподіл функціональності між клієнтською та серверною частинами.

Програмна реалізація включає клієнтську частину, створену з використанням HTML, CSS та JavaScript, а також серверну частину, реалізовану на базі Node.js та Express.js. Для зберігання даних використано систему керування базами даних PostgreSQL. Взаємодія між компонентами системи здійснюється через REST API, а доступ до функціоналу захищено за допомогою токенної автентифікації (JWT).

У системі реалізовано основні функціональні можливості: створення, редагування, перегляд та видалення документів, завантаження файлів, а також управління доступом користувачів до інформації. Особлива увага приділена забезпеченню цілісності даних та зручності користувацького інтерфейсу.

Проведено функціональне тестування системи, перевірено коректність роботи основних модулів та виконано оптимізацію продуктивності.

Сферою використання застосунку є автоматизація документообігу в організаціях, а також використання для навчальних або особистих цілей з метою підвищення ефективності роботи з документами.

КЛЮЧОВІ СЛОВА: ЕЛЕКТРОННИЙ ДОКУМЕНТООБІГ, ВЕБ-ЗАСТОСУНОК, NODE.JS, EXPRESS.JS, POSTGRESQL, REST API, JWT, АВТЕНТИФІКАЦІЯ, УПРАВЛІННЯ ДОКУМЕНТАМИ, АВТОМАТИЗАЦІЯ.

ЗМІСТ

ВСТУП

У сучасних умовах цифровізації суспільства та стрімкого розвитку інформаційних технологій актуальності набуває питання ефективного управління документами. Велика частина інформації в організаціях, навчальних закладах та бізнес-середовищі представлена у вигляді документів, що потребують створення, обробки, зберігання, передачі та підтвердження їх достовірності. Традиційні паперові формати документообігу поступово втрачають свою актуальність через складність у використанні, витрати часу та обмежені можливості доступу. Саме тому впровадження електронних систем документообігу є важливим кроком у напрямку автоматизації та оптимізації роботи з інформацією.

Актуальність теми спричинена необхідністю створення зручних, безпечних і функціональних цифрових рішень для роботи з документами. Існуючі системи часто є складними у використанні або перевантаженими не потрібним функціоналом, що ускладнює їх адаптацію для окремих користувачів або невеликих організацій. Тому розробка власної системи електронного документообігу, яка поєднує простоту, зрозумілий інтерфейс та базові необхідні функції, є актуальним і практично значущим завданням.

Метою роботи є підвищення ефективності процесу електронного документообігу за допомогою розробленого веб-застосунку управління документами на основі клієнт-серверної архітектури, сучасних веб-технологій та методів автоматизації обробки даних.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати існуючі системи електронного документообігу та визначити їх переваги і недоліки.
2. Сформулювати вимоги до функціоналу майбутнього веб-застосунку.
3. Спроєктувати архітектуру системи з використанням клієнт-серверної моделі.
4. Реалізувати серверну частину із використанням Node.js та Express.js.

5. Налаштувати базу даних PostgreSQL для збереження користувачів та документів.

6. Розробити клієнтську частину із використанням HTML, CSS та JavaScript.

7. Реалізувати механізм авторизації користувачів на основі JWT.

8. Додати функціонал створення, редагування, підписання та видалення документів.

9. Провести тестування системи та оцінити її працездатність.

Об’єктом дослідження є процеси електронного документообігу в цифровому середовищі.

Предметом дослідження є програмні засоби та технології, що використовуються для створення систем управління документами з можливістю їх підписання та спільного використання.

Джерельну базу дослідження становлять сучасні наукові публікації, технічна документація до використовуваних технологій (Node.js, Express.js, PostgreSQL), матеріали щодо реалізації REST API, систем авторизації на основі JWT, а також інтернет-ресурси, присвячені розробці веб-застосунків та UI/UX-дизайну.

Фактичний матеріал роботи включає розроблений веб-застосунок електронного документообігу, створений із використанням технологій Node.js, Express.js, PostgreSQL, HTML, CSS та JavaScript. У ході реалізації були створені API-маршрути для роботи з користувачами та документами, реалізовано механізм авторизації, обробку файлів та функціонал електронного підпису. Система була протестована в умовах локального середовища з моделюванням реальних сценаріїв використання.

Методи дослідження, використані у роботі, включають: метод аналізу, метод проєктування, метод програмного експерименту, метод системного аналізу.

Наукова новизна роботи полягає в розробці веб-застосунку, який поєднує базові функції електронного документообігу з можливістю взаємодії між користувачами та реалізацією електронного підпису. Запропоновано підхід до

організації простої та зрозумілої системи, що може бути легко адаптована під різні потреби.

Теоретична цінність роботи полягає в узагальненні підходів до побудови клієнт-серверних веб-застосунків для роботи з документами та використанні сучасних технологій для забезпечення їх безпеки та зручності.

Практичне значення полягає в можливості використання розробленої системи як інструменту для роботи з документами в навчальних цілях, невеликих командах або індивідуальній діяльності. Система може бути розширена шляхом додавання нових функцій, таких як рольова модель доступу, покращений електронний підпис або інтеграція з іншими сервісами.

Апробація результатів здійснювалася шляхом тестування веб-застосунку в локальному середовищі. Було перевірено роботу основних функцій: реєстрація користувачів, авторизація, створення документів, редагування, підписання та взаємодія між користувачами. Виявлені помилки були усунені, що дозволило підвищити стабільність і надійність системи.

Отже, розроблена система електронного документообігу відповідає сучасним вимогам, є функціональною, зручною у використанні та може слугувати основою для подальшого розвитку та вдосконалення.

1 АНАЛІЗ ІСНУЮЧИХ ЗАСОБІВ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної області

Документи завжди були базою управління та комунікації в організаціях. Вони підтверджують факти, регулюють взаємовідносини між суб'єктами, зберігають інформацію та виступають інструментом обліку і контролю. Проте традиційно документообіг здійснювався у паперовому форматі, що призводило до значних витрат часу та ресурсів на їх обробку та переміщення. Компанії та підприємства змушені були витрачати додаткові зусилля на друк, копіювання, передачу та зберігання документів.

Крім того, паперовий документообіг супроводжується рядом проблем, серед яких складність пошуку необхідної інформації, ризик втрати або пошкодження документів, а також обмежений доступ до них. Особливо це проявляється в організаціях із великою кількістю працівників або складною структурою управління, де документи проходять через декілька рівнів погодження.

Ще однією важливою проблемою є затримки в процесі прийняття рішень. Передача документа від одного підрозділу до іншого може займати значний час, що негативно впливає на ефективність роботи організації в цілому. У сучасних умовах, коли швидкість обробки інформації має вирішальне значення, такі затримки стають суттєвим недоліком.

Також варто враховувати фінансовий аспект: витрати на папір, друк, обслуговування техніки та організацію архівів можуть бути значними, особливо для великих підприємств. Окрім цього, паперовий документообіг має негативний вплив на довкілля через використання природних ресурсів.

Ще одним недоліком паперового документообігу є складність контролю за виконанням документів. Відстеження статусу документа, визначення відповідальних осіб та контроль термінів виконання часто здійснюються вручну, що створює додаткове навантаження на працівників і підвищує ризик помилок. У

разі втрати документа або затримки на одному з етапів його обробки знайти причину проблеми буває складно.

З розвитком інформаційних технологій та поширенням мережі Інтернет виникла можливість переходу до електронного документообігу. Сучасні інформаційні системи дозволяють не лише зберігати документи у цифровому вигляді, але й автоматизувати більшість процесів, пов’язаних з їх обробкою. Це включає маршрутизацію документів, автоматичне повідомлення користувачів, контроль виконання завдань та ведення історії змін.

Електронний документообіг забезпечує централізоване зберігання інформації, що значно спрощує доступ до неї. Користувачі можуть працювати з документами незалежно від свого місцезнаходження, що особливо актуально в умовах дистанційної роботи. Крім того, значно спрощується процес пошуку інформації, оскільки документи можна знаходити за ключовими словами, датами або іншими параметрами.

Важливою перевагою електронного документообігу є можливість інтеграції з іншими інформаційними системами. Наприклад, системи документообігу можуть взаємодіяти з бухгалтерськими програмами, системами управління персоналом або іншими корпоративними рішеннями. Це дозволяє створити єдине інформаційне середовище, в якому дані не дублюються, а використовуються спільно різними підрозділами.

Також електронні системи забезпечують підвищений рівень безпеки даних. Використання сучасних методів захисту, таких як аутентифікація користувачів, розмежування прав доступу та шифрування даних, дозволяє зменшити ризик несанкціонованого доступу до інформації. Крім того, електронні системи дозволяють вести журнал дій користувачів, що дає змогу відстежувати всі операції з документами.

Однією з ключових складових сучасних систем є електронний підпис. Він забезпечує можливість підтвердження автентичності документа та дозволяє визначити, ким саме був підписаний документ. Це важливо не лише з точки зору безпеки, але й для надання документам юридичної сили. Використання

електронного підпису дозволяє повністю відмовитися від паперових копій у багатьох процесах.

Однак впровадження електронного документообігу також має певні складнощі. Воно потребує адаптації існуючих бізнес-процесів, навчання персоналу та технічного забезпечення. У деяких випадках працівники можуть негативно сприймати нові технології через необхідність змінювати звичні способи роботи. Тому важливо правильно організувати процес впровадження та забезпечити підтримку користувачів.

Таким чином, електронний документообіг є важливим етапом розвитку сучасних інформаційних систем. Він дозволяє значно підвищити ефективність роботи з документами, зменшити витрати часу та ресурсів, а також забезпечити більш зручний і безпечний доступ до інформації. Перехід до цифрових технологій у цій сфері є закономірним і необхідним кроком для сучасних організацій.

1.2 Аналіз потреб користувача та постановка задачі

У процесі розробки будь-якої інформаційної системи важливим етапом є визначення потреб користувачів. Саме від того, наскільки правильно будуть сформульовані вимоги, залежить ефективність та зручність майбутнього програмного продукту. Особливо це актуально для систем електронного документообігу, які повинні не лише забезпечувати базовий функціонал, але й бути зрозумілими для користувачів із різним рівнем підготовки.

Користувачами систем електронного документообігу можуть бути працівники організацій, керівники, адміністративний персонал, а також інші особи, які взаємодіють із документами у межах своєї діяльності. Кожна з цих груп має власні потреби та очікування від системи, що необхідно враховувати під час її розробки.

Однією з основних потреб користувачів є можливість швидкого створення документів. Система повинна забезпечувати зручний інтерфейс для введення даних, можливість редагування та збереження інформації без зайвих дій. Важливо,

щоб користувач міг створити документ за мінімальний час, не витрачаючи додаткових зусиль на освоєння складного функціоналу.

Ще однією важливою потребою є зручний доступ до документів. Користувачі повинні мати можливість переглядати, редагувати або передавати документи незалежно від місця свого перебування. Це особливо актуально в умовах віддаленої роботи, коли доступ до інформації має здійснюватися через мережу Інтернет.

Важливим аспектом є також організація зберігання документів. Система повинна забезпечувати структуроване збереження даних, щоб користувачі могли легко знаходити необхідну інформацію, для цього використовуються механізми пошуку.

Користувачі також потребують можливості взаємодії між собою. У процесі роботи часто виникає необхідність передати документ на погодження або підпис іншій особі. Тому система повинна підтримувати механізми взаємодії, які дозволяють організувати колективну роботу з документами.

Однією з ключових потреб є забезпечення безпеки даних. Оскільки документи можуть містити конфіденційну інформацію, необхідно обмежити доступ до них для сторонніх осіб. Це досягається за допомогою механізмів аутентифікації та авторизації користувачів. Кожен користувач повинен мати доступ лише до тих документів, які відповідають його ролі в системі.

Крім того, користувачі очікують, що система буде стабільною та швидкою у роботі. Затримки при відкритті документів або виконанні операцій можуть негативно впливати на ефективність роботи. Тому важливо забезпечити оптимальну швидкодію системи.

Ще однією важливою потребою є можливість роботи з файлами. Документи можуть містити вкладення у вигляді різних файлів, тому система повинна забезпечувати їх завантаження, збереження та перегляд. Це дозволяє користувачам працювати з різними типами інформації в межах одного застосунку.

Також слід враховувати потребу у відстеженні змін. Користувачі повинні мати можливість бачити, хто і коли вносив зміни до документа. Це дозволяє контролювати процес роботи та уникати помилок.

На основі проведеного аналізу можна виділити основні функціональні вимоги до системи електронного документообігу:

1. Реєстрація та авторизація користувачів.
2. Створення та редагування документів.
3. Перегляд та пошук документів.
4. Завантаження та обробка файлів.
5. Передача документів між користувачами.
6. Можливість підписання документів.
7. Забезпечення безпеки доступу до даних.

Окрім функціональних вимог, важливими є і нефункціональні вимоги. До них належать:

1. Зручність інтерфейсу.
2. Швидкодія системи.
3. Надійність збереження даних.
4. Безпека інформації.
5. Можливість масштабування.

Після визначення потреб користувачів можна сформулювати задачу розробки. Основною задачею є створення веб-застосунку електронного документообігу, який дозволяє користувачам працювати з документами у цифровому середовищі.

Система повинна реалізовувати клієнт-серверну архітектуру, де клієнтська частина забезпечує взаємодію з користувачем, а серверна частина виконує обробку запитів та роботу з базою даних.

Особлива увага приділяється реалізації механізму авторизації, який забезпечує захист даних. Для цього використовується технологія JWT, що дозволяє ідентифікувати користувачів та обмежувати доступ до інформації.

Також важливою задачею є реалізація функціоналу роботи з документами, який включає їх створення, редагування, видалення та пошук. Крім того, система повинна забезпечувати можливість підписання документів іншими користувачами.

У процесі розробки необхідно враховувати взаємодію між різними компонентами системи. Це включає клієнтську частину, серверну логіку та базу даних. Важливо забезпечити їх узгоджену роботу для досягнення максимальної ефективності.

Таким чином, аналіз потреб користувачів дозволяє сформулювати чітку постановку задачі та визначити основні напрямки розробки системи. Це є важливим етапом, який забезпечує успішну реалізацію програмного продукту.

1.3 Формулювання цілей та функціональних вимог до системи

Загальна мета створення системи електронного документообігу полягає в розробці зручного, функціонально наповненого та ефективного веб-застосунку, який дозволяє організувати повноцінну роботу з документами в електронному вигляді. Така система повинна забезпечувати створення, збереження, редагування, передачу та підписання документів у межах єдиного цифрового середовища. Вона виступає не просто сховищем файлів, а комплексним інструментом управління документообігом, який сприяє оптимізації внутрішніх процесів організації, зменшенню витрат часу та підвищенню ефективності взаємодії між користувачами. В умовах сучасної цифровізації та переходу до дистанційної роботи особливо важливо забезпечити швидкий доступ до документів, їхню безпечну обробку та можливість взаємодії між різними користувачами незалежно від їхнього місцезнаходження. Саме тому створення такої системи є актуальним завданням, спрямованим на підвищення продуктивності роботи та спрощення процесів управління документацією.

У межах поставленої мети важливим завданням є створення такого веб-застосунку електронного документообігу, який одночасно буде простим у використанні для нових користувачів і достатньо гнучким для тих, хто має досвід роботи з подібними системами. Система повинна дозволяти виконувати базові дії – створення документа, його перегляд і збереження – без зайвих складнощів, а також поступово надавати доступ до розширених можливостей: редагування,

пошукуфі, передачі документів між користувачами, призначення підписанта та відстеження статусу документа. Такий підхід забезпечує комфортну адаптацію користувача до системи та запобігає перевантаженню інтерфейсу зайвими функціями на початковому етапі роботи. Реалізація застосунку на основі технологій Node.js та JavaScript дозволяє забезпечити швидку обробку запитів, стабільність роботи та зручну взаємодію між клієнтською і серверною частинами. Важливу роль відіграє використання клієнт-серверної архітектури, яка розділяє інтерфейс користувача, бізнес-логіку та роботу з базою даних PostgreSQL. Це забезпечує масштабованість системи, підвищує рівень безпеки та дозволяє адаптувати її до різних умов використання – як для індивідуальної роботи з документами, так і для взаємодії між кількома користувачами в межах організації.

Загальна мета створення системи електронного документообігу також передбачає досягнення балансу між ручним керуванням документами та автоматизацією основних процесів. Користувач повинен мати змогу самостійно створювати, редагувати та передавати документи, водночас отримуючи зручні інструменти для контролю їх стану, нагадування про необхідність підписання або обробки, а також візуальні підказки щодо поточного статусу документів. Система повинна підтримувати механізми відстеження життєвого циклу документа – від моменту створення до завершення обробки або підписання. Це дозволяє користувачеві чітко розуміти, на якому етапі знаходиться документ і які дії необхідно виконати далі. Важливою складовою є організація взаємодії між користувачами, зокрема можливість передавання документів іншим особам для перегляду або підписання. Такий підхід забезпечує спрощення внутрішніх процесів у межах організації та значно скорочує час погодження документів. У певному сенсі система виконує роль “розумного” інструменту документообігу, який не лише зберігає інформацію, а й оптимізує робочі процеси, мінімізує затримки та допомагає уникати втрати важливих даних. Не менш важливою складовою є забезпечення безпеки. Усі дані повинні зберігатися в захищеній базі даних PostgreSQL, доступ до якої здійснюється лише після авторизації користувача. Використання JWT-токенів дозволяє контролювати доступ до системи та

обмежувати виконання певних дій лише для авторизованих користувачів. Додатково передбачено розмежування прав доступу, що дозволяє забезпечити коректну взаємодію між користувачами та захист інформації від несанкціонованого доступу. У сукупності ці аспекти формують загальну мету як створення надійного, зручного та функціонального веб-застосунку, який ефективно підтримує процеси електронного документообігу в сучасних умовах.

Функціональні вимоги до клієнтської та серверної частини системи електронного документообігу формуються на основі мети створення застосунку, потреб користувачів, а також вимог до зручності використання, безпеки та ефективності обробки інформації. Клієнтська частина (frontend) повинна забезпечувати зручний та інтуїтивно зрозумілий інтерфейс користувача, який дозволяє виконувати основні операції з документами: створення, перегляд, редагування, пошук, завантаження файлів, а також передачу документів іншим користувачам для подальшої обробки або підписання. Особливу увагу слід приділити простоті взаємодії, щоб користувач міг виконувати всі необхідні дії за мінімальну кількість кроків. Інтерфейс має забезпечувати візуальну структурованість інформації, відображення списку документів, їх статусів (створено, передано, підписано), а також можливість швидкого пошуку. Важливою вимогою є адаптивність інтерфейсу до різних пристроїв, що дозволяє працювати із системою як на персональних комп’ютерах, так і на мобільних пристроях. Окрім цього, клієнтська частина повинна підтримувати асинхронну взаємодію з сервером через використання Fetch API, що забезпечує оновлення даних без перезавантаження сторінки та підвищує зручність роботи користувача. До основних функцій frontend-частини належать: реєстрація та авторизація користувача, створення та редагування документів, перегляд списку документів, пошук за назвою або змістом, передача документа іншому користувачу, підписання документа, а також вихід із системи. Такий набір функцій дозволяє реалізувати повноцінний цикл роботи з документами в межах веб-застосунку та забезпечує ефективну взаємодію між користувачами.

Серверна частина (backend), у свою чергу, відповідає за обробку всіх запитів із клієнтської частини, реалізацію бізнес-логіки, взаємодію з базою даних, авторизацію користувачів, а також за забезпечення безпеки та стабільності роботи системи електронного документообігу. До основних функціональних вимог до серверної частини належить реалізація RESTful API з відповідними маршрутами для автентифікації користувачів (реєстрація, вхід у систему, перевірка токена), управління документами (створення, отримання, редагування, видалення), роботи з файлами (завантаження та збереження документів), а також організації взаємодії між користувачами, зокрема передачі документів та їх підписання. Важливим аспектом є забезпечення коректної обробки кожного запиту, перевірка прав доступу користувача та обмеження доступу до даних інших користувачів. Особливу увагу необхідно приділити безпеці системи. Паролі користувачів повинні зберігатися у зашифрованому вигляді, а доступ до системи здійснюватися лише після успішної авторизації із використанням JWT-токенів. Кожен запит до серверної частини має супроводжуватися перевіркою токена, що дозволяє уникнути несанкціонованого доступу. База даних PostgreSQL повинна мати чітку структуру та включати таблиці користувачів і документів, а також містити інформацію про власника документа, підписанта, статус документа, дату створення та інші службові дані. Така структура дозволяє ефективно організувати зберігання інформації та забезпечити швидкий доступ до неї.

Серверна частина повинна також підтримувати обробку помилок, логування подій та забезпечувати стабільність роботи при одночасному виконанні кількох запитів. Важливо передбачити можливість масштабування системи, що дозволить у майбутньому обробляти більшу кількість користувачів без зниження продуктивності. Додатково можуть бути реалізовані механізми обмеження частоти запитів (rate limiting) та резервного копіювання даних, що підвищує надійність системи.Усі зміни в системі повинні фіксуватися із прив’язкою до користувача та часу виконання дії, що дозволяє реалізувати історію змін документів і забезпечити контроль за їх обробкою. Це особливо важливо в системах документообігу, де

необхідно відстежувати, хто саме створив, редагував або підписав документ. Такий підхід підвищує прозорість роботи системи та довіру до неї.

Таблиця 1.1

Функціональні вимоги до серверної частини системи

Категорія функцій	Опис функціоналу
Автентифікація	Реєстрація користувача, вхід у систему, генерація та перевірка JWT-токенів
Управління користувачами	Збереження даних користувачів, перевірка доступу до ресурсів
Управління документами	Створення, отримання, редагування та видалення документів
Робота з файлами	Завантаження файлів (pdf, docx), збереження на сервері
Взаємодія користувачів	Передача документів між користувачами
Електронний підпис	Підписання документа іншим користувачем, збереження підпису
Контроль доступу	Перевірка прав користувача перед виконанням операцій
Робота з БД	Збереження та отримання даних із PostgreSQL
Обробка запитів	Обробка HTTP-запитів через REST API
Обробка помилок	Виявлення та обробка помилок при виконанні запитів

Продовження таблиці 1.1

Категорія функцій	Опис функціоналу
Логування	Фіксація дій користувачів у системі
Безпека	Захист даних, перевірка токенів, обмеження доступу

Важливою функціональною складовою є реалізація електронного підпису документів. Система повинна забезпечувати можливість передачі документа іншому користувачу для підписання, після чого документ отримує відповідний статус (наприклад, “підписано”) та зберігає інформацію про підписанта. Це дозволяє організувати базовий процес погодження документів у цифровому середовищі та значно спрощує взаємодію між користувачами. З технічної точки зору, серверна частина реалізується із використанням платформи Node.js та фреймворку Express.js, що забезпечує швидку обробку HTTP-запитів та зручну організацію API. Для роботи з базою даних використовується PostgreSQL разом із бібліотекою pg, а для обробки файлів – multer. Така комбінація технологій дозволяє створити ефективну, стабільну та масштабовану систему електронного документообігу. У підсумку функціональні вимоги до серверної частини спрямовані на забезпечення надійності, безпеки, продуктивності та зручності використання системи. Їх реалізація є необхідною умовою для досягнення головної мети – створення ефективного веб-застосунку, який дозволяє оптимізувати процеси роботи з документами, підвищити швидкість їх обробки та забезпечити зручну взаємодію між користувачами.

1.4 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики програмної системи, які забезпечують її ефективну, стабільну та безпечну роботу. На відміну від функціональних вимог, що описують конкретні дії системи, нефункціональні вимоги визначають умови, за яких ці дії виконуються. У межах розробки веб-застосунку електронного документообігу такі вимоги формуються з урахуванням сучасних стандартів якості програмного забезпечення та особливостей використання системи в реальному середовищі.

Однією з ключових нефункціональних характеристик є продуктивність системи. Веб-застосунок повинен забезпечувати швидку обробку запитів користувачів та оперативне відображення результатів. Час відгуку системи на стандартні дії користувача, такі як відкриття списку документів, створення або редагування документа, не повинен перевищувати 1-2 секунд при середньому навантаженні. Завдяки використанню асинхронної моделі обробки запитів у середовищі Node.js забезпечується можливість одночасного обслуговування декількох користувачів без блокування виконання. Система повинна підтримувати роботу щонайменше 50-100 одночасних користувачів без суттєвого зниження швидкодії.

Не менш важливою характеристикою є надійність системи. Програмний продукт повинен забезпечувати стабільну роботу без збоїв і втрати даних. Усі операції, пов’язані зі створенням, редагуванням, передачею та підписанням документів, повинні виконуватися коректно навіть у випадку виникнення помилок. Рівень доступності системи (availability) має становити не менше 95-99% часу роботи. У разі виникнення помилок система повинна обробляти їх і надавати користувачу зрозумілі повідомлення, не перериваючи загальний процес роботи.

Однією з найважливіших нефункціональних вимог є безпека. Оскільки система працює з конфіденційною інформацією, вона повинна забезпечувати захист даних користувачів і документів від несанкціонованого доступу. Для цього

використовується механізм авторизації на основі JWT-токенів, які перевіряються при кожному запиті до серверу. Термін дії токена обмежений у часі, що знижує ризик несанкціонованого використання. Паролі користувачів повинні зберігатися у зашифрованому вигляді із використанням сучасних алгоритмів хешування. Передача даних між клієнтом і сервером повинна здійснюватися через захищений протокол HTTPS. Доступ до функцій системи повинен бути обмежений відповідно до прав користувача, що забезпечує контроль над операціями з документами.

Значну роль відіграє зручність використання системи. Інтерфейс користувача повинен бути інтуїтивно зрозумілим, логічно структурованим і не перевантаженим зайвими елементами. Час навчання користувача роботі із системою повинен бути мінімальним, а основні операції мають виконуватися не більше ніж за 2-3 кліки. Використання асинхронних запитів дозволяє уникнути перезавантаження сторінки, що значно покращує користувацький досвід і підвищує швидкість взаємодії з системою.

Важливою характеристикою є масштабованість системи. Архітектура веб-застосунку повинна дозволяти розширення функціоналу без значних змін у вже реалізованих компонентах. Система повинна бути готовою до збільшення кількості користувачів і обсягу даних. Завдяки використанню клієнт-серверної архітектури та модульного підходу забезпечується можливість додавання нових функцій, таких як система ролей, інтеграція з електронним підписом або зовнішніми сервісами.

Ще однією важливою нефункціональною вимогою є сумісність. Система повинна коректно працювати у сучасних веб-браузерах, таких як Google Chrome, Mozilla Firefox та Microsoft Edge. Інтерфейс повинен адаптуватися до різних розмірів екранів, що забезпечує можливість використання системи як на стаціонарних комп’ютерах, так і на мобільних пристроях. Час завантаження основної сторінки не повинен перевищувати 2-3 секунд при стандартному інтернет-з’єднанні.

Не менш важливою є підтримуваність програмного коду. Код системи повинен бути структурованим, зрозумілим і добре документованим. Використання модульної архітектури дозволяє розділити логіку на окремі компоненти, що значно

спрощує внесення змін і подальший розвиток системи. Це дозволяє скоротити час на супровід і мінімізувати ризик виникнення помилок при внесенні змін.

Важливим аспектом є цілісність даних. Усі операції з документами повинні виконуватися таким чином, щоб уникнути втрати або пошкодження інформації. База даних PostgreSQL забезпечує підтримку транзакцій, що дозволяє гарантувати коректність виконання операцій. У разі помилки всі зміни можуть бути скасовані, що запобігає появі неконсистентних даних.

Таблиця 1.4

Нефункціональні вимоги до системи електронного документообігу

№	Характеристика	Опис вимог	Показники / критерії
1	Продуктивність	Система повинна швидко обробляти запити користувачів та відображати результати	Час відгуку ≤ 1-2 с, підтримка 50-100 користувачів
2	Надійність	Система повинна працювати стабільно без збоїв та втрати даних	Доступність 95-99%, обробка помилок
3	Безпека	Захист даних користувачів та документів	JWT-авторизація, HTTPS, шифрування паролів
4	Масштабованість	Можливість розширення системи без зміни архітектури	Додавання нових модулів без переписування

Продовження таблиці 1.4

№	Характеристика	Опис вимог	Показники / критерії
5	Сумісність	Підтримка різних браузерів та пристроїв	Chrome, Firefox, Edge, адаптивний дизайн
6	Підтримуваність	Код має бути структурованим та зрозумілим	Модульна архітектура
7	Цілісність даних	Дані повинні зберігатися без втрат і помилок	Використання транзакцій PostgreSQL
8	Доступність	Система повинна бути доступною у будь-який час	Робота 24/7 при наявності інтернету
9	Логування	Фіксація дій користувачів у системі	Запис операцій (create, edit, delete, sign)
10	Зручність використання	Інтерфейс має бути простим та інтуїтивно зрозумілим	Виконання дій за 2-3 кліки

Крім того, система повинна забезпечувати логування подій. Усі важливі дії користувачів, такі як створення, редагування, видалення або підписання документів, повинні фіксуватися. Це дозволяє відстежувати історію змін і забезпечує додатковий рівень контролю за роботою системи.

1.5 Дослідження існуючих аналогів

У процесі розробки системи електронного документообігу важливим етапом є аналіз існуючих програмних рішень, які вже використовуються для організації роботи з документами. Дослідження аналогів дозволяє визначити сильні та слабкі сторони сучасних систем, оцінити їх функціональні можливості, а також виявити недоліки, які можуть бути враховані під час розробки власного програмного продукту. Крім того, аналіз існуючих рішень дає змогу сформувати більш чітке уявлення про вимоги користувачів та тенденції розвитку даної предметної області, що є особливо важливим у контексті стрімкої цифровізації бізнес-процесів .

Сучасний ринок програмного забезпечення пропонує значну кількість систем електронного документообігу, які відрізняються за функціональністю, складністю, рівнем інтеграції та сферою застосування. До найбільш відомих систем можна віднести такі рішення, як Microsoft SharePoint, Google Workspace (Google Docs, Google Drive), Dropbox, а також спеціалізовані системи електронного документообігу, що використовуються в організаціях різного масштабу. Кожна з цих систем має свої особливості, що визначають її придатність для використання в певних умовах.

1.5.1 Microsoft SharePoint

Microsoft SharePoint є однією з найбільш потужних платформ для організації корпоративного документообігу та спільної роботи. Вона входить до екосистеми Microsoft 365 і широко використовується великими компаніями для централізованого зберігання, обробки та контролю документів. Основна ідея SharePoint полягає у створенні єдиного інформаційного простору, в якому користувачі можуть взаємодіяти з документами, обмінюватися інформацією та організовувати робочі процеси.

Система дозволяє створювати бібліотеки документів, у яких кожен файл має власні атрибути, версії та історію змін. Це забезпечує можливість відстеження всіх дій із документом, включаючи редагування, перегляд і передачу. Однією з важливих функцій є контроль версій, який дозволяє повернутися до попереднього стану документа у разі помилки або необхідності.

SharePoint також підтримує автоматизацію процесів документообігу через так звані workflow. Це дозволяє налаштувати маршрути погодження документів, де кожен документ автоматично передається відповідним користувачам. Наприклад, документ може пройти етапи створення, перевірки, затвердження та архівування.

Основні функції Microsoft SharePoint:

1. Централізоване зберігання документів у бібліотеках.
2. Управління версіями документів (збереження історії змін).
3. Налаштування прав доступу для користувачів і груп.
4. Автоматизація бізнес-процесів (workflow погодження).
5. Інтеграція з Microsoft Office (Word, Excel, Outlook).
6. Створення внутрішніх порталів і сайтів компанії.
7. Пошук документів за ключовими словами та метаданими.
8. Спільна робота з файлами в реальному часі.
9. Контроль життєвого циклу документа (створення - погодження - архів).

Крім того, платформа має розвинену систему управління доступом. Адміністратор може визначати, хто має право переглядати, редагувати або видаляти документи. Це забезпечує високий рівень безпеки та контроль над інформацією.

Однак, незважаючи на всі переваги, SharePoint має суттєві недоліки. Основною проблемою є складність системи. Для її впровадження та налаштування потрібні спеціалісти, а користувачам необхідно проходити навчання. Також система є досить дорогою у використанні, що обмежує її застосування у невеликих організаціях.

Таким чином, Microsoft SharePoint є потужним, але складним рішенням, яке більше підходить для великих підприємств, ніж для невеликих або навчальних проєктів.

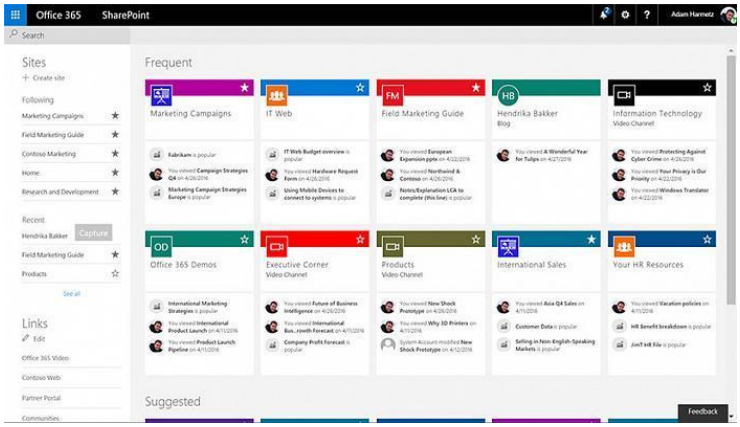


Рис.1.1 Microsoft SharePoint

1.5.2 Google Workspace

Google Workspace є однією з найпопулярніших платформ для роботи з документами в хмарному середовищі. Вона включає в себе набір сервісів, таких як Google Docs, Google Sheets, Google Drive та інші інструменти, які дозволяють користувачам створювати, редагувати та зберігати документи онлайн.

Однією з ключових переваг Google Workspace є можливість одночасної роботи кількох користувачів над одним документом. Це дозволяє організувати ефективну командну роботу без необхідності надсилати файли між учасниками. Усі зміни зберігаються автоматично, а система веде історію редагування, що дозволяє відстежувати внесені правки.

Google Drive виконує роль сховища, де користувачі можуть зберігати документи, організовувати їх у папки та надавати доступ іншим користувачам. Система підтримує різні рівні доступу: перегляд, коментування та редагування.

Однак Google Workspace не є повноцінною системою електронного документообігу. Вона не підтримує складні сценарії погодження документів, не має чіткої системи статусів (наприклад, “на підписі”, “затверджено”) та не забезпечує автоматизацію процесів на рівні корпоративних систем.

Основні функції Google Workspace:

1. Створення та редагування документів онлайн (Google Docs).
2. Спільна робота кількох користувачів у реальному часі.
3. Автоматичне збереження змін.

4. Зберігання файлів у Google Drive.
5. Налаштування доступу (перегляд, коментування, редагування).
6. Історія змін документа.
7. Коментарі та обговорення в документі.
8. Доступ із будь-якого пристрою через браузер.
9. Експорт і імпорт файлів у різних форматах (PDF, DOCX).

Також варто зазначити залежність від інтернету та хмарної інфраструктури Google. Це може бути проблемою у випадках, коли потрібен повний контроль над даними.

Отже, Google Workspace є зручним інструментом для роботи з документами, але не може повністю замінити спеціалізовану систему електронного документообігу.

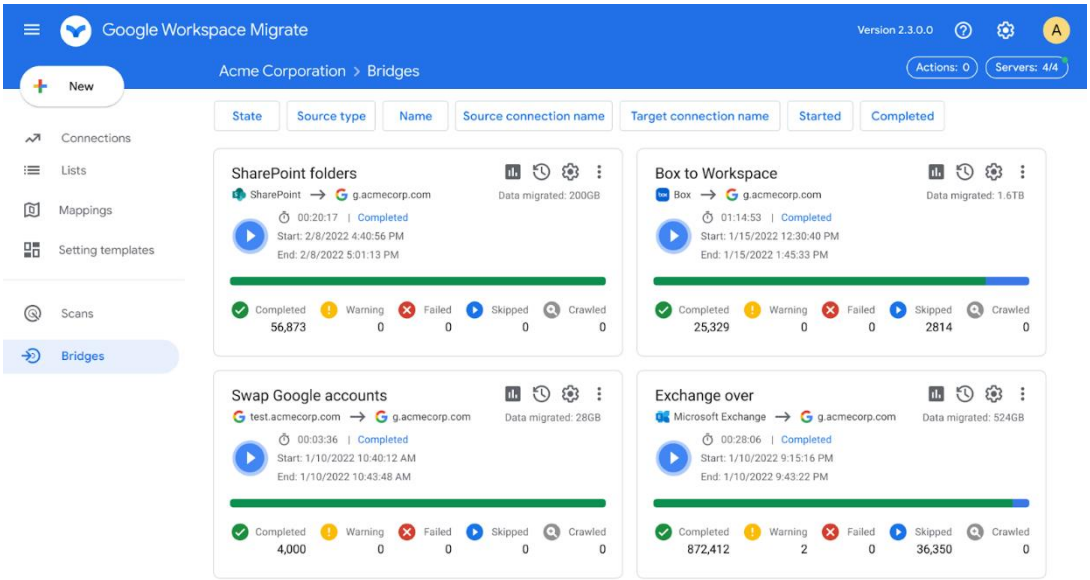


Рис.1.2 Google Workspace

1.5.3 Dropbox

Dropbox є хмарним сервісом для зберігання та обміну файлами, який широко використовується як індивідуальними користувачами, так і командами. Основною

функцією сервісу є синхронізація файлів між пристроями та забезпечення доступу до них через інтернет.

Сервіс дозволяє створювати спільні папки, до яких можуть мати доступ кілька користувачів. Це дає змогу організувати базову командну роботу. Також Dropbox підтримує створення посилань для доступу до файлів, що значно спрощує їх передачу.

Останні версії Dropbox мають додаткові функції, такі як перегляд документів онлайн, коментування та базове редагування. Проте ці можливості залишаються обмеженими порівняно з повноцінними системами документообігу.

Основні функції Dropbox:

1. Хмарне зберігання файлів.
2. Синхронізація файлів між пристроями.
3. Створення спільних папок для командної роботи.
4. Генерація посилань для обміну файлами.
5. Контроль доступу до файлів.
6. Перегляд документів онлайн.
7. Коментування файлів.
8. Передача великих файлів.
9. Інтеграція з іншими сервісами.

Головною проблемою Dropbox є те, що він не підтримує логіку документообігу. У ньому відсутні статуси документів, маршрути погодження, автоматичне передавання документів між користувачами та інші важливі функції.

Таким чином, Dropbox є ефективним інструментом для зберігання файлів, але не може використовуватися як повноцінна система управління документами.

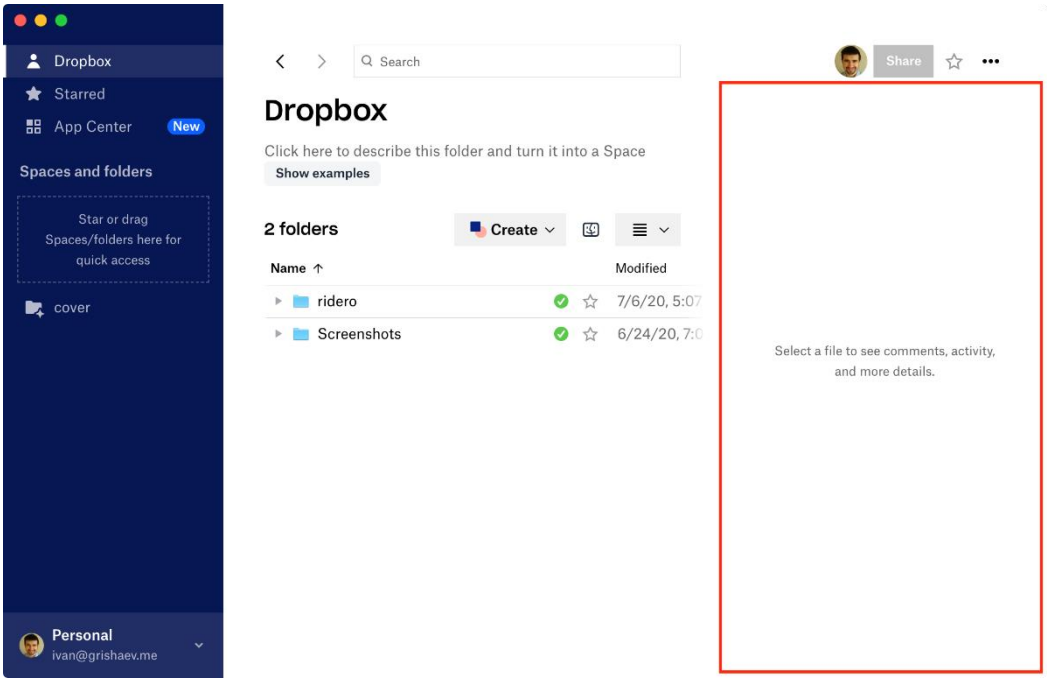


Рис.1.3 Dropbox

1.5.4 Вчасно

Сервіс «Вчасно» є одним із найпопулярніших українських рішень у сфері електронного документообігу. Він орієнтований на обмін документами між компаніями, працівниками та контрагентами.

Система дозволяє створювати документи, надсилати їх на погодження, підписувати за допомогою електронного підпису та відстежувати їх статус. Однією з ключових переваг є підтримка юридично значущого електронного підпису, що дозволяє використовувати систему в офіційних бізнес-процесах.

Основні функції сервісу «Вчасно»:

1. Обмін електронними документами між користувачами.
2. Створення маршрутів погодження документів.
3. Підписання документів електронним підписом (КЕП).
4. Відстеження статусу документа.
5. Коментування та обговорення документів.
6. Управління доступом до документів.
7. Інтеграція з іншими бізнес-сервісами.
8. Робота з контрагентами.
9. Зберігання історії змін і дій.

«Вчасно» також підтримує створення маршрутів документообігу, що дозволяє автоматизувати передачу документів між різними учасниками процесу. Користувачі можуть залишати коментарі, відстежувати зміни та отримувати повідомлення про стан документів. Проте система має і певні недоліки. Вона є комерційною, і частина функціоналу доступна лише за підпискою. Також користувач не має повного контролю над системою, оскільки вона працює як сторонній сервіс. Отже, «Вчасно» є ефективним рішенням для бізнесу, але не підходить для задач, де необхідна повна кастомізація системи.

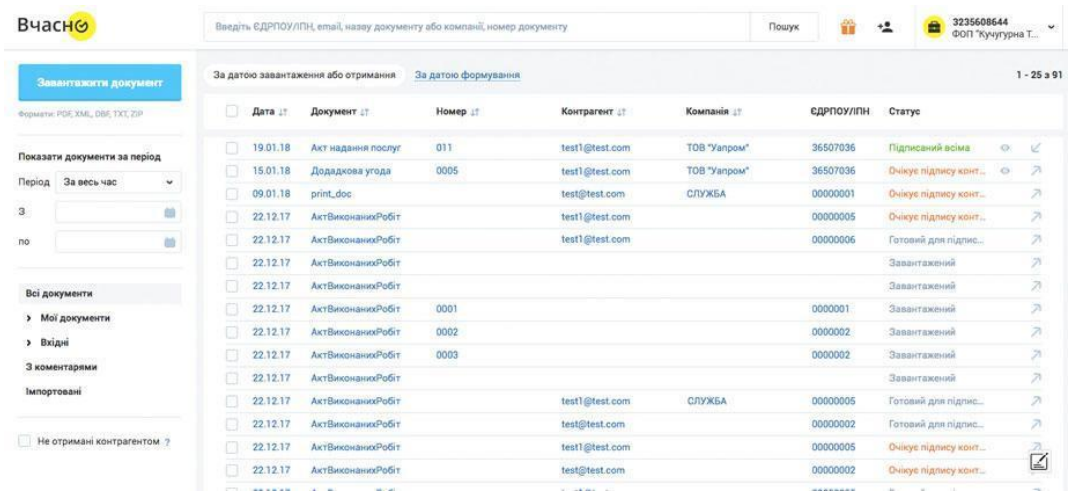


Рис.1.4 Вчасно

1.5.5 M.E.Doc

M.E.Doc є програмним забезпеченням, яке широко використовується в Україні для організації електронного документообігу та подання податкової звітності. Дана система орієнтована насамперед на юридично значущий обмін документами між підприємствами, а також взаємодію з державними органами. Вона є одним із найпоширеніших рішень у сфері бухгалтерського обліку та електронної звітності.

Основною особливістю M.E.Doc є підтримка електронного цифрового підпису (ЕЦП), що дозволяє надавати документам юридичну силу. Завдяки цьому документи, створені в системі, можуть використовуватися у правових відносинах між підприємствами та державними установами. Це забезпечує повноцінну заміну паперового документообігу та сприяє переходу до цифрових технологій.

Система дозволяє здійснювати обмін документами з контрагентами в електронному вигляді, підписувати їх електронним підписом, зберігати в централізованій базі даних та відстежувати їх статус. Крім того, M.E.Doc підтримує інтеграцію з державними сервісами, такими як податкова служба, що дозволяє автоматизувати процес подання звітності та отримання відповідей від державних органів.

Однією з ключових функцій системи є автоматизація документообігу між підприємствами. Це дозволяє значно скоротити час обробки документів, мінімізувати людський фактор та зменшити кількість помилок. Крім того, система забезпечує централізоване зберігання документів, що полегшує їх пошук та управління.

Слід також зазначити, що M.E.Doc підтримує ведення журналів подій, що дозволяє відстежувати всі дії користувачів із документами. Це є важливим аспектом для забезпечення контролю та аудиту, особливо у великих організаціях.

Основні функції M.E.Doc:

1. Електронний обмін документами з контрагентами.
2. Підписання документів електронним цифровим підписом.
3. Формування та подання електронної звітності до державних органів.
4. Зберігання документів у базі даних та їх архівування.
5. Інтеграція з державними інформаційними системами.
6. Автоматизація бухгалтерських і фінансових процесів.
7. Контроль статусу документів (відправлено, отримано, підписано тощо).
8. Ведення журналів дій користувачів.
9. Робота з юридично значущими документами.
10. Імпорт і експорт даних з інших систем.

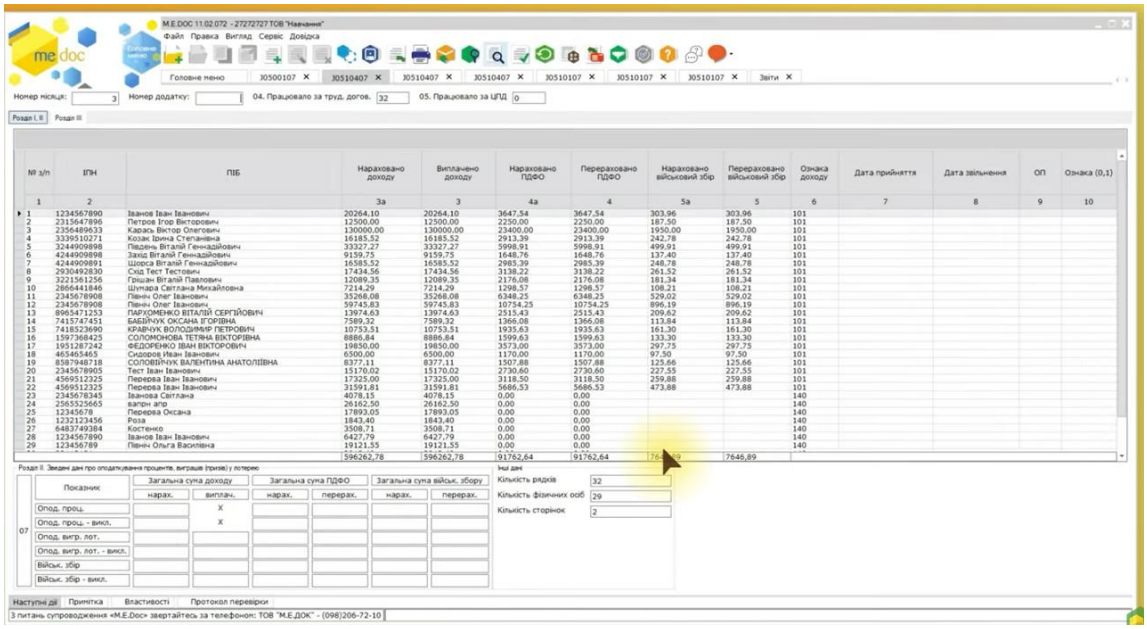


Рис.1.5 M.E.Doc

Разом із тим, незважаючи на широкий функціонал, система M.E.Doc має ряд обмежень. Зокрема, вона має вузьку спеціалізацію та орієнтована переважно на бухгалтерський і юридичний документообіг. Це обмежує її застосування для універсального управління документами в організаціях, де потрібна більш гнучка система з підтримкою різних типів документів і сценаріїв використання.

Крім того, інтерфейс системи є досить складним для нових користувачів і потребує певного рівня підготовки. Велика кількість функцій і налаштувань може ускладнювати освоєння системи, особливо для невеликих організацій або користувачів без досвіду роботи з подібними програмами.

Також слід враховувати, що M.E.Doc є комерційним програмним продуктом, що потребує придбання ліцензії та регулярного оновлення. Це може бути недоліком для користувачів, які шукають більш доступні або безкоштовні рішення.

Ще одним обмеженням є залежність від локального встановлення (у класичній версії), що ускладнює доступ до системи з різних пристроїв та обмежує мобільність користувачів. У сучасних умовах це може бути менш зручним у порівнянні з веб-орієнтованими рішеннями.

Таким чином, M.E.Doc є потужним і надійним інструментом для електронного документообігу у сфері бухгалтерії та взаємодії з державними

органами. Проте його вузька спеціалізація, складність інтерфейсу та обмежена гнучкість роблять його менш придатним для універсальних систем управління документами, що враховують широкий спектр користувацьких потреб.

2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура програмної системи

Окрім загальних принципів побудови системи, важливим аспектом архітектури є організація внутрішньої логіки серверної частини. У даному застосунку вона побудована таким чином, щоб забезпечити послідовну обробку кожного запиту. Кожен HTTP-запит проходить через кілька етапів: отримання даних, перевірка авторизації, валідація вхідних параметрів, виконання бізнес-логіки, взаємодія з базою даних та формування відповіді. Така структура дозволяє чітко контролювати виконання кожної операції та уникати помилок, пов’язаних із некоректними даними або несанкціонованим доступом.

У контексті роботи з користувачами система реалізує базову модель аутентифікації та авторизації. Аутентифікація відповідає за перевірку особи користувача (логін і пароль), тоді як авторизація визначає, які дії цей користувач може виконувати в системі. Завдяки використанню JWT-токенів система не потребує збереження сесій на сервері, що спрощує архітектуру та дозволяє масштабувати застосунок без додаткових складнощів. Кожен запит містить токен, який перевіряється сервером, і лише після цього виконується відповідна дія.

Особливу роль у системі відіграє модель даних. Вона визначає, як саме інформація зберігається в базі даних і як пов’язані між собою різні сутності. У даному випадку основними сутностями є користувачі та документи. Користувач має унікальний ідентифікатор, ім’я та пароль, а документ містить інформацію про назву, зміст, файл, статус, автора та підписанта. Важливо, що один документ пов’язаний одразу з двома користувачами – тим, хто його створив, і тим, хто його підписує. Це дозволяє реалізувати механізм взаємодії між користувачами, який є ключовим для системи документообігу.

Ще одним важливим елементом архітектури є обробка станів системи. Система повинна коректно реагувати на різні сценарії використання: створення документа без підписанта, передача документа іншому користувачу, підписання документа, спроба підписання не тим користувачем тощо. Для цього використовуються умовні перевірки та статуси документів, які дозволяють контролювати логіку переходів між різними станами. Такий підхід забезпечує передбачувану поведінку системи та зменшує ймовірність помилок.

Значну увагу в архітектурі приділено обробці помилок. У реальних умовах користувач може вводити некоректні дані, сервер може не знайти потрібний запис у базі, або ж можуть виникнути проблеми із доступом. Усі ці ситуації повинні оброблятися таким чином, щоб користувач отримував зрозуміле повідомлення, а система не припиняла свою роботу. Для цього використовуються стандартні HTTP-статуси (400, 401, 403, 404, 500), які дозволяють чітко визначити тип помилки.

Важливим аспектом є також організація взаємодії між клієнтською частиною та сервером. У даному застосунку використовується асинхронний підхід, який дозволяє виконувати запити без блокування інтерфейсу користувача. Це означає, що після виконання дії (наприклад, створення документа) сторінка не перезавантажується, а лише оновлюється відповідна частина інтерфейсу. Такий підхід значно покращує швидкість роботи та зручність використання системи.

Ще одним важливим аспектом є забезпечення цілісності даних. У системі використовується механізм хешування документа під час підписання. Це означає, що формується унікальний код (хеш), який залежить від вмісту документа. Якщо документ буде змінений після підписання, хеш зміниться, що дозволить виявити порушення цілісності. Таким чином, навіть без використання повноцінного криптографічного підпису система забезпечує базовий рівень контролю за незмінністю документів.

З точки зору розширення функціоналу, архітектура системи передбачає можливість додавання нових модулів. Наприклад, у майбутньому можна реалізувати систему ролей (адміністратор, користувач), додати журнал дій,

впровадити повідомлення або інтеграцію з електронною поштою. Завдяки модульному підходу ці зміни можна внести без повного переписування системи.

Також варто зазначити, що архітектура враховує можливість розгортання системи на сервері. Веб-застосунок може працювати як локально, так і на віддаленому сервері, що дозволяє використовувати його в реальних умовах. Для цього достатньо налаштувати серверну частину та підключення до бази даних.

У контексті продуктивності система використовує асинхронну модель обробки запитів, що є характерною для Node.js. Це дозволяє ефективно працювати з великою кількістю одночасних користувачів. На відміну від синхронних систем, де кожен запит обробляється послідовно, асинхронний підхід дозволяє виконувати кілька операцій одночасно, що значно підвищує швидкість роботи.

Ще одним важливим аспектом є тестування системи. Архітектура побудована таким чином, щоб окремі компоненти можна було перевіряти незалежно один від одного. Наприклад, можна протестувати роботу авторизації, створення документів або підписання окремо. Це дозволяє швидше знаходити помилки та забезпечує більш стабільну роботу застосунку.

Крім того, система враховує принципи REST-архітектури, що означає використання стандартних HTTP-методів і чітко визначених маршрутів. Це спрощує розуміння структури API та дозволяє легко інтегрувати систему з іншими застосунками у майбутньому.

У підсумку, архітектура програмної системи побудована з урахуванням сучасних підходів до розробки веб-застосунків. Вона забезпечує розділення функціональних компонентів, ефективну обробку даних, безпеку, масштабованість і зручність використання. Завдяки цьому система може виконувати всі необхідні функції електронного документообігу та бути основою для подальшого розвитку і вдосконалення.

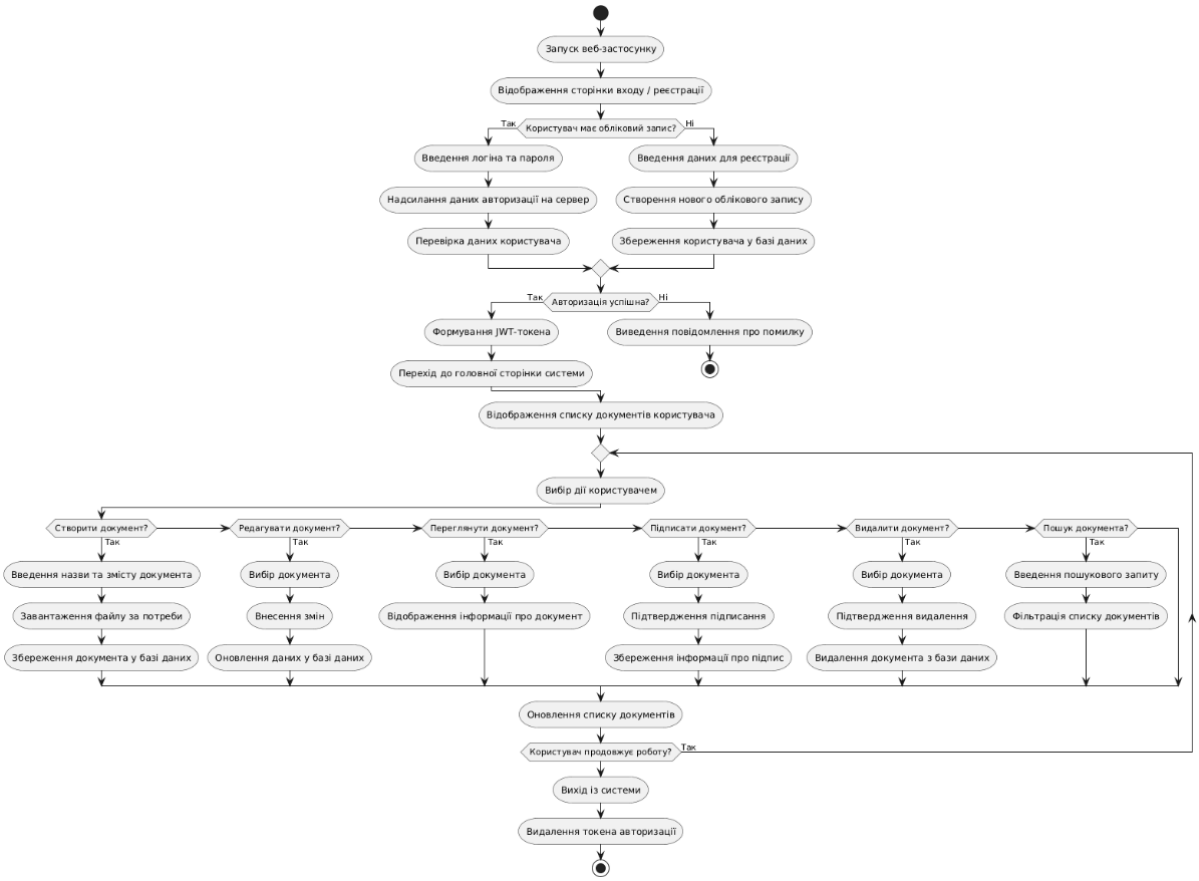


Рис. 2.1 Алгоритм роботи програми

2.2 Вибір технологій та інструментів реалізації

Вибір технологій та інструментів реалізації є одним із ключових етапів розробки програмного забезпечення, оскільки саме він визначає не лише функціональні можливості майбутнього застосунку, але й його продуктивність, надійність, масштабованість, безпеку та зручність супроводження. У процесі створення веб-застосунку електронного документообігу особливу увагу було приділено вибору такого технологічного стеку, який би відповідав сучасним вимогам до веб-розробки, забезпечував швидкість роботи системи та дозволяв реалізувати необхідний функціонал без надмірного ускладнення архітектури.

Основою серверної частини обрано середовище виконання Node.js, яке дозволяє виконувати JavaScript-код поза межами браузера. Однією з головних переваг Node.js є його асинхронна модель обробки запитів, яка забезпечує високу продуктивність при роботі з великою кількістю одночасних користувачів. На

відміну від традиційних серверних технологій, що використовують багатопоточну модель, Node.js працює за принципом подійно-орієнтованої архітектури, що дозволяє уникнути блокування потоків виконання та ефективно використовувати ресурси системи. Це особливо важливо для веб-застосунків, де одночасно можуть виконуватися десятки або сотні запитів.

Альтернативами Node.js могли бути такі технології, як Python (Flask або Django), Java (Spring Boot) або PHP (Laravel). Проте Node.js було обрано через його універсальність, можливість використання однієї мови програмування як на клієнтській, так і на серверній частині, а також високу швидкість розробки. Крім того, велика кількість доступних бібліотек і активна спільнота розробників значно спрощують процес створення застосунку.

Для організації серверної логіки використовується фреймворк Express.js, який є одним із найпопулярніших рішень для створення веб-серверів на Node.js. Express.js забезпечує зручний механізм маршрутизації, що дозволяє розподілити обробку запитів за різними URL-адресами. Кожен маршрут відповідає за певну функцію системи, наприклад авторизацію, створення документа або його підписання. Використання Express.js дозволяє структурувати код, зробити його зрозумілим та підтримуваним.

Серед альтернатив Express.js можна виділити такі фреймворки, як Koa або NestJS. Однак Express.js було обрано через його простоту, гнучкість і широке поширення. Він не нав’язує жорсткої структури проєкту, що дозволяє розробнику самостійно організувати код відповідно до потреб системи.

Для зберігання даних у системі використовується реляційна база даних PostgreSQL. Вибір саме цієї системи управління базами даних обумовлений її високою надійністю, підтримкою складних запитів і можливістю роботи з великими обсягами даних. PostgreSQL дозволяє створювати зв’язки між таблицями, використовувати індекси для прискорення пошуку та забезпечувати цілісність даних.

Альтернативними рішеннями могли бути MySQL або NoSQL-бази даних, такі як MongoDB. Проте PostgreSQL має більш розширені можливості для роботи

зі структурованими даними, що є важливим для системи електронного документообігу, де необхідно зберігати чітко визначені зв’язки між користувачами, документами та їх статусами.

Для взаємодії з базою даних використовується бібліотека pg, яка дозволяє виконувати SQL-запити безпосередньо з серверного коду. Це дає змогу гнучко керувати даними та реалізовувати складну бізнес-логіку. Використання pg забезпечує високу швидкість роботи та повний контроль над виконанням запитів.

Одним із ключових аспектів системи є забезпечення безпеки, тому для реалізації авторизації було використано технологію JWT (JSON Web Token). JWT дозволяє створювати токени, які містять інформацію про користувача та використовуються для перевірки його прав доступу. Після успішного входу користувач отримує токен, який зберігається на клієнтській стороні та передається разом із кожним запитом до сервера. Сервер перевіряє цей токен і визначає, чи має користувач право виконувати відповідну дію.

Використання JWT має ряд переваг: відсутність необхідності зберігати сесії на сервері, можливість масштабування системи та підвищений рівень безпеки. Альтернативним підходом могла бути сесійна авторизація, проте вона потребує додаткових ресурсів для зберігання сесій і ускладнює масштабування системи.

Для реалізації функціоналу завантаження файлів було використано бібліотеку multer, яка дозволяє обробляти запити типу multipart/form-data. Вона забезпечує можливість завантаження документів на сервер, їх збереження у файловій системі та подальшого використання. Це є важливою складовою системи електронного документообігу, оскільки користувачі повинні мати можливість працювати з файлами різних форматів.

Ще одним важливим інструментом є бібліотека cors, яка використовується для налаштування політики доступу між різними джерелами. Вона дозволяє клієнтській частині, яка може працювати на іншому порту або домені, взаємодіяти із сервером без виникнення помилок безпеки. Це особливо актуально під час розробки, коли frontend і backend працюють окремо.

Клієнтська частина системи реалізована з використанням стандартних веб-технологій: HTML, CSS та JavaScript. Такий підхід дозволяє створити легкий і швидкий інтерфейс без використання складних фреймворків. Це спрощує розробку, зменшує залежності та забезпечує кращий контроль над кодом.

Для обміну даними використовується Fetch API, який дозволяє виконувати асинхронні HTTP-запити. Завдяки цьому клієнт може отримувати дані від сервера та оновлювати інтерфейс без перезавантаження сторінки. Це значно покращує користувацький досвід і робить застосунок більш сучасним.

Важливим критерієм вибору технологій була також їх інтеграція між собою. Усі використані інструменти добре поєднуються між собою, що дозволяє створити єдину систему без конфліктів і проблем сумісності. Крім того, вони мають велику кількість документації, що спрощує процес розробки та навчання.

Ще одним важливим фактором є можливість масштабування системи. Обрані технології дозволяють розширювати функціонал без суттєвих змін у структурі застосунку. У майбутньому можна реалізувати додаткові функції, такі як система ролей, повідомлення, інтеграція з іншими сервісами або впровадження електронного підпису.

Також слід зазначити, що обраний стек технологій відповідає сучасним тенденціям у веб-розробці. Використання JavaScript як основної мови програмування дозволяє створити єдине середовище розробки, що спрощує підтримку коду та підвищує ефективність роботи розробника.

Таким чином, вибір технологій та інструментів реалізації був обґрунтований вимогами до системи, її функціональністю та перспективами розвитку. Обраний стек забезпечує високу продуктивність, безпеку, зручність використання та можливість масштабування, що робить його оптимальним для створення веб-застосунку електронного документообігу.

2.3 Розробка інтерфейсу користувача

Розробка інтерфейсу користувача є важливим етапом створення веб-застосунку, оскільки саме інтерфейс визначає, наскільки зручно та ефективно користувач може взаємодіяти із системою. Навіть при наявності потужної серверної частини та широкого функціоналу, складний або незрозумілий інтерфейс може суттєво знизити ефективність використання програмного продукту. Саме тому при розробці веб-застосунку електронного документообігу особлива увага була приділена створенню інтуїтивно зрозумілого, простого та функціонального інтерфейсу.

Основною метою розробки інтерфейсу було забезпечення максимально зручної взаємодії користувача із системою без необхідності додаткового навчання. Інтерфейс повинен бути зрозумілим навіть для користувачів без технічного досвіду, що особливо важливо в умовах впровадження системи в організаціях, де рівень цифрової грамотності може відрізнятися.

Клієнтська частина застосунку реалізована з використанням стандартних веб-технологій – HTML, CSS та JavaScript. Використання базових технологій дозволило створити легкий і швидкий інтерфейс без перевантаження зайвими залежностями. HTML відповідає за структуру сторінок, CSS – за їх візуальне оформлення, а JavaScript – за реалізацію логіки взаємодії користувача з системою.

Інтерфейс застосунку побудований за принципами мінімалізму та функціональності. Основні елементи розташовані таким чином, щоб користувач міг швидко знайти потрібну функцію. Головна сторінка містить перелік документів або робочу область, де відображається вся необхідна інформація. Важливі дії, такі як створення документа, редагування або видалення, винесені у вигляді кнопок із зрозумілими підписами.

Окрему увагу було приділено формам введення даних. Форми для створення та редагування документів реалізовані таким чином, щоб користувач міг швидко заповнити необхідні поля. Для цього використовуються текстові поля, випадаючі

списки та кнопки підтвердження. Усі поля мають підказки, що дозволяє уникнути помилок при введенні даних.

Важливою складовою інтерфейсу є система авторизації. Користувач має можливість зареєструватися або увійти в систему через окрему сторінку. Інтерфейс цієї сторінки максимально спрощений і містить лише необхідні поля: ім’я користувача та пароль. Після успішної авторизації користувач отримує доступ до основного функціоналу системи.

Для підвищення зручності використання було реалізовано динамічне оновлення даних без перезавантаження сторінки. Це досягається за допомогою Fetch API, який дозволяє надсилати асинхронні запити до серверної частини. Завдяки цьому користувач може виконувати дії, такі як створення або редагування документа, без необхідності перезавантаження сторінки, що значно покращує користувацький досвід.

Інтерфейс також передбачає відображення повідомлень про результати виконання дій. Наприклад, при успішному створенні документа користувач отримує відповідне повідомлення, а у разі помилки – повідомлення з описом проблеми. Це дозволяє швидко реагувати на помилки та покращує взаємодію з системою.

Особливу увагу було приділено адаптивності інтерфейсу. Система повинна коректно відображатися на різних пристроях, включаючи комп’ютери, планшети та смартфони. Для цього використовуються адаптивні стилі CSS, які змінюють розташування елементів залежно від розміру екрану. Це дозволяє користувачам працювати із системою незалежно від використовуваного пристрою.

Ще одним важливим аспектом є логічна структура навігації. Усі сторінки системи пов’язані між собою таким чином, щоб користувач міг легко переходити між ними. Основні розділи доступні через меню або кнопки, що значно спрощує навігацію. Це дозволяє зменшити час на пошук необхідної інформації та підвищує ефективність роботи.

При розробці інтерфейсу також враховувалися принципи UX/UI-дизайну, зокрема:

1. Простота та зрозумілість.
2. Послідовність у розташуванні елементів.
3. Швидкість доступу до основних функцій.
4. Мінімізація кількості дій для виконання задачі.

Важливим є також використання кольорової гами та візуальних елементів. Для оформлення інтерфейсу використано стримані кольори, які не відволікають користувача від основної роботи. Кнопки та важливі елементи виділяються кольором, що дозволяє швидко їх ідентифікувати.

Крім того, інтерфейс передбачає можливість роботи з файлами. Користувач може завантажувати документи через спеціальну форму, після чого вони відображаються у списку. Це забезпечує зручну роботу з електронними документами без необхідності використання сторонніх програм.

У процесі розробки інтерфейсу виникали певні труднощі, пов’язані з організацією взаємодії між клієнтською та серверною частинами. Зокрема, необхідно було правильно налаштувати обмін даними, обробку помилок та відображення результатів. Проте їх вирішення дозволило краще зрозуміти принципи роботи веб-застосунків і створити більш якісний продукт.

У подальшому інтерфейс може бути вдосконалений шляхом додавання нових функцій, таких як система повідомлень, темна тема оформлення або персоналізація інтерфейсу під користувача. Це дозволить зробити застосунок ще більш зручним і функціональним.

Таким чином, розроблений інтерфейс користувача відповідає основним вимогам сучасних веб-застосунків: він є простим, зрозумілим, функціональним та адаптивним. Він забезпечує ефективну взаємодію користувача із системою та створює комфортні умови для роботи з електронними документами.

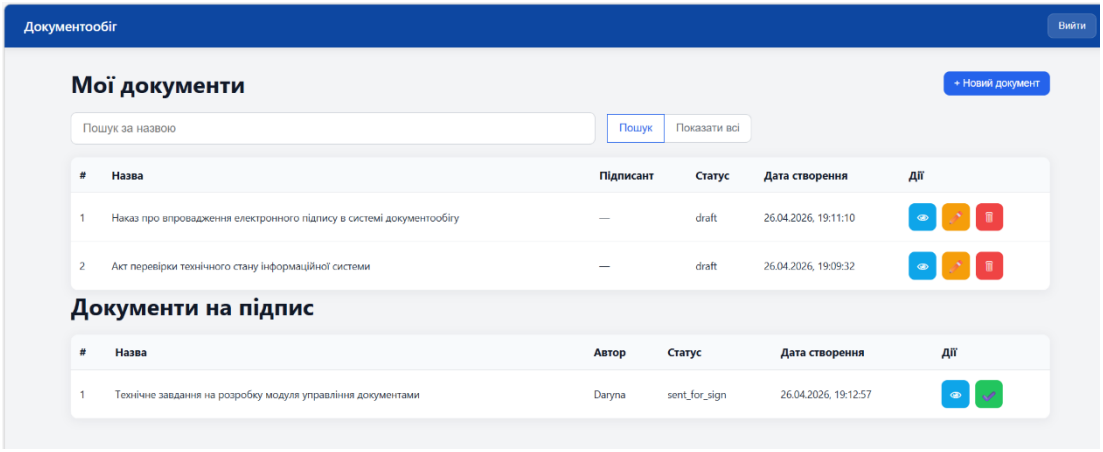


Рис. 2.2 Інтерфейс додатка

2.4 Реалізація функціональних модулів

Реалізація функціональних модулів є одним із найважливіших етапів створення веб-застосунку електронного документообігу, адже саме тут відбувається практичне втілення всіх закладених можливостей системи. У межах роботи було розроблено комплекс взаємопов’язаних модулів, які забезпечують повний цикл роботи користувача з документами — від моменту входу в систему до створення, редагування, передачі та підписання. Клієнтська частина побудована на основі HTML, CSS та JavaScript, тоді як серверна реалізована на Node.js із використанням фреймворку Express.js та бази даних PostgreSQL. Взаємодія між компонентами здійснюється через REST API, що дозволяє організувати динамічну роботу без необхідності перезавантаження сторінки.

Першим функціональним блоком виступає модуль реєстрації та авторизації користувачів. Він відповідає за контроль доступу та ідентифікацію особи. На клієнтській стороні реалізовано форму введення логіну та пароля, яка після заповнення надсилає HTTP-запит на сервер. Сервер перевіряє коректність даних, створює новий запис у базі при реєстрації або генерує JWT-токен у випадку авторизації. Токен зберігається у локальному сховищі браузера та використовується для підтвердження особи при кожному наступному запиті, що гарантує безпечний доступ до функцій системи.

Після входу користувач отримує доступ до основного інтерфейсу, де реалізовано модуль управління документами. Він дозволяє створювати нові документи, переглядати існуючі, редагувати їх та видаляти. При створенні документа користувач вводить назву, текстовий зміст і може додати файл. Дані передаються на сервер асинхронним запитом, після чого зберігаються у базі даних. Кожен документ має атрибути: автора, дату створення, статус та прикріплений файл. Це забезпечує повний контроль над життєвим циклом документа. Редагування та видалення реалізовані через відповідні запити до серверу, після чого інтерфейс автоматично оновлюється, що створює ефект «живої» системи.

Важливим елементом є модуль передачі документа на підпис. Автор може призначити іншого користувача як підписанта, і документ отримує відповідний статус. У базі даних зберігається ідентифікатор підписанта, що створює зв’язок між двома користувачами. Для підписанта реалізовано окремий інтерфейс, де відображається список документів, що потребують його дії. Це дозволяє швидко знаходити потрібні документи та уникати плутанини при великій кількості записів.

Модуль підписання документів є завершальним етапом роботи. Після натискання кнопки «Підписати» сервер перевіряє права користувача. Якщо він є призначеним підписантом, документ отримує статус «підписаний», а в базі даних зберігається інформація про дату та особу, яка виконала підпис. Додатково формується контрольний хеш, що дозволяє перевірити цілісність документа. Якщо після підписання документ буде змінено, система зможе це виявити, що підвищує рівень довіри та захист від фальсифікацій.

Окремим блоком виступає модуль роботи з файлами. Він дозволяє завантажувати документи у вигляді файлів різних форматів – PDF, DOCX, TXT. Користувач обирає файл, який передається на сервер і зберігається у файловій системі, тоді як у базі даних зберігається лише інформація про нього. Це оптимізує роботу системи та зменшує навантаження на базу.

Для забезпечення стабільності роботи було реалізовано механізм обробки помилок. Якщо користувач намагається виконати дію, на яку не має прав, або документ не знайдено, сервер повертає відповідний код та повідомлення. Це

повідомлення відображається у клієнтському інтерфейсі, що дозволяє швидко зрозуміти причину проблеми та уникнути непорозумінь.

Усі модулі взаємодіють між собою, утворюючи єдину систему. Авторизація відкриває доступ до управління документами, модуль передачі забезпечує взаємодію між користувачами, модуль підписання завершує процес документообігу, а модуль файлів доповнює функціонал. Така інтеграція дозволяє реалізувати повноцінний електронний документообіг у межах веб-застосунку.

Таким чином, реалізація функціональних модулів дала змогу створити цілісну систему, яка охоплює всі основні процеси роботи з документами. Вона дозволяє користувачам створювати документи, передавати їх на підпис, контролювати їхній стан та працювати з файлами. Завдяки використанню сучасних технологій та правильно організованій архітектурі система є гнучкою, зручною у використанні та придатною для подальшого розвитку.

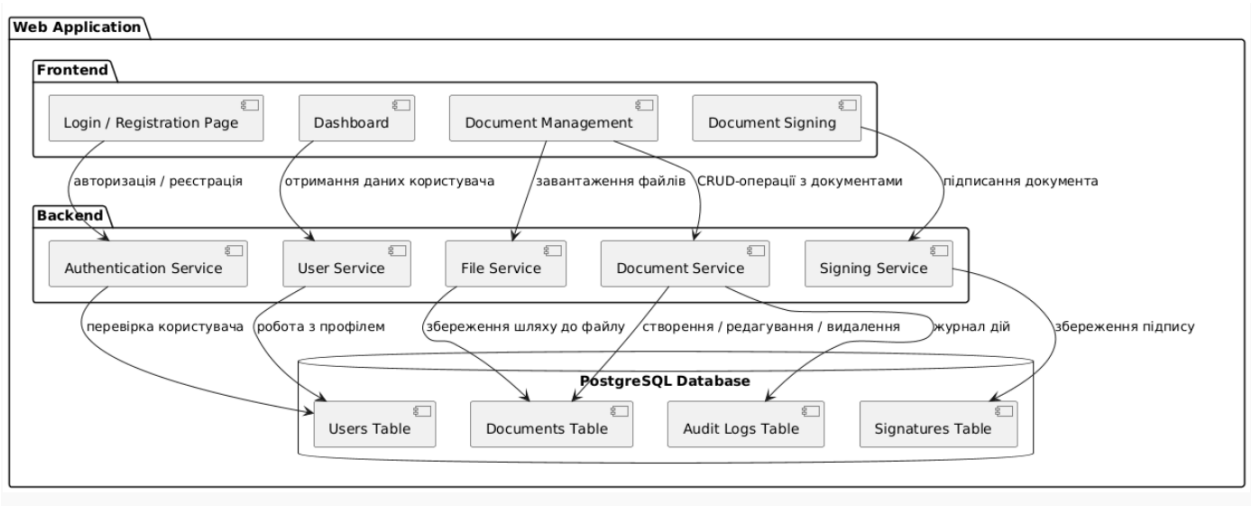


Рис.2.3 Діаграма компонентів програмної системи

3 ЗАСОБИ РЕАЛІЗАЦІЇ

3.1 Огляд технологій

У процесі розробки веб-застосунку електронного документообігу вибір технологічного стеку є одним із ключових етапів, оскільки саме він визначає можливості системи, її ефективність, масштабованість, безпеку та зручність подальшого супроводу. Враховуючи сучасні тенденції розвитку інформаційних технологій, до веб-застосунків висуваються підвищені вимоги щодо швидкодії, підтримки великої кількості користувачів, інтерактивності інтерфейсу та захисту даних. Саме тому обраний стек повинен не лише забезпечувати реалізацію базового функціоналу, а й створювати основу для подальшого розвитку системи.

У рамках даної роботи було використано сучасний набір технологій, що включає Node.js, Express.js, PostgreSQL, JSON Web Token (JWT), Multer, Fetch API, а також клієнтські технології HTML, CSS і JavaScript. Такий стек дозволяє реалізувати повноцінну клієнт-серверну архітектуру, у якій кожен компонент виконує чітко визначені функції. Важливою перевагою є використання JavaScript як універсальної мови програмування, що застосовується як на стороні клієнта, так і на стороні сервера, що значно спрощує розробку та зменшує складність інтеграції компонентів.

Крім того, всі обрані технології мають активну підтримку спільноти, регулярні оновлення та велику кількість готових бібліотек, що дозволяє скоротити час розробки та підвищити якість програмного продукту. Вони також добре поєднуються між собою, що забезпечує стабільну та передбачувану роботу системи.

3.1.1 Node.js та Express.js

Node.js є високопродуктивним середовищем виконання JavaScript, яке базується на рушії V8, розробленому компанією Google. Даний рушій забезпечує швидку компіляцію JavaScript-коду у машинний код, що дозволяє значно підвищити продуктивність виконання програм. Основною особливістю Node.js є використання подієво-орієнтованої архітектури та неблокуючої моделі введення-виведення (non-blocking I/O), що дозволяє ефективно працювати з великою кількістю одночасних запитів.

На відміну від традиційних серверних платформ, які використовують багатопотокову модель, Node.js працює на основі однопотокового циклу подій (event loop). Це означає, що всі запити обробляються в одному потоці, але при цьому довготривалі операції, такі як робота з базою даних або файловою системою, виконуються асинхронно. Завдяки цьому система не блокується під час виконання таких операцій, а продовжує обробляти інші запити. Такий підхід дозволяє значно підвищити продуктивність та ефективність використання ресурсів сервера.

У контексті системи електронного документообігу використання Node.js є особливо доцільним, оскільки дозволяє забезпечити швидку та стабільну роботу навіть при великій кількості користувачів. Наприклад, коли один користувач виконує завантаження документа або надсилає запит на підписання, інші користувачі можуть паралельно переглядати документи, редагувати їх або виконувати інші дії без затримок. Це забезпечує високу швидкодію системи та комфортну роботу користувачів.

Крім того, Node.js має розвинену екосистему пакетів, доступних через менеджер npm (Node Package Manager). Це дозволяє використовувати готові бібліотеки для реалізації різних функцій, таких як авторизація, обробка файлів, робота з базою даних тощо. Завдяки цьому значно скорочується час розробки та підвищується якість програмного продукту.

Express.js є мінімалістичним і гнучким веб-фреймворком для Node.js, який значно спрощує створення серверної частини застосунку. Він надає зручний інтерфейс для роботи з HTTP-запитами, маршрутизацією та middleware.

Використання Express.js дозволяє структурувати серверний код і розділити його на логічні компоненти, що значно спрощує підтримку та масштабування системи.

Однією з ключових можливостей Express.js є система маршрутизації. Вона дозволяє визначати, які дії повинні виконуватися у відповідь на певні HTTP-запити. Наприклад, запит типу GET може використовуватися для отримання списку документів, POST – для створення нового документа, PUT – для його редагування, а DELETE – для видалення. Такий підхід відповідає принципам RESTful API та забезпечує стандартизовану взаємодію між клієнтом і сервером.

У розробленій системі електронного документообігу Express.js використовується для реалізації серверної логіки, яка включає обробку запитів користувачів, взаємодію з базою даних та виконання бізнес-логіки. Наприклад, окремі маршрути відповідають за реєстрацію та авторизацію користувачів, створення та редагування документів, передачу їх на підпис, а також обробку файлів. Така організація дозволяє чітко розділити функціональність системи та зробити її більш зрозумілою.

Важливою складовою Express.js є підтримка middleware – спеціальних функцій, які виконуються під час обробки запиту до того, як він буде переданий до основного обробника. Middleware дозволяє реалізувати додаткову логіку, таку як перевірка JWT-токена, логування запитів, обробка помилок або валідація даних. Це робить систему більш гнучкою та дозволяє легко додавати нові функціональні можливості.

З точки зору безпеки Express.js дозволяє реалізувати механізми контролю доступу до ресурсів, що є критично важливим для систем електронного документообігу. Наприклад, можна перевіряти, чи має користувач право редагувати документ або виконувати інші дії. Це дозволяє запобігти несанкціонованому доступу до даних.

Ще однією перевагою Express.js є його легковаговість і гнучкість. На відміну від більш складних фреймворків, він не нав’язує жорстку структуру проєкту, що дозволяє розробнику самостійно визначати архітектуру застосунку. Це особливо

корисно при розробці індивідуальних рішень, таких як система електронного документообігу.

Водночас використання Node.js та Express.js має і певні недоліки. Зокрема, асинхронна модель програмування може ускладнювати написання та відлагодження коду, особливо для складних операцій. Крім того, необхідно ретельно контролювати обробку помилок, щоб уникнути збоїв у роботі системи.

Таким чином, використання Node.js та Express.js у веб-застосунку електронного документообігу є обґрунтованим і доцільним рішенням. Вони забезпечують високу продуктивність, гнучкість, масштабованість та зручність розробки, що дозволяє створити сучасний та ефективний програмний продукт, здатний працювати в умовах реального навантаження.

3.1.2 PostgreSQL

PostgreSQL є потужною об’єктно-реляційною системою управління базами даних (СУБД), яка широко використовується у сучасних інформаційних системах завдяки своїй надійності, продуктивності та гнучкості. Вона підтримує стандарт мови SQL і розширює його додатковими можливостями, що дозволяє працювати не лише з класичними таблицями, а й зі складними структурами даних, такими як JSON, масиви, геодані тощо. Це робить PostgreSQL універсальним рішенням для розробки складних веб-застосунків.

У системі електронного документообігу PostgreSQL використовується як основне сховище даних. У базі даних зберігається інформація про користувачів, документи, їх статуси, ролі, а також зв’язки між цими сутностями. Наприклад, кожен документ пов’язаний із користувачем, який його створив, і з користувачем, який повинен його підписати. Така структура дозволяє реалізувати логічно узгоджену модель даних і забезпечує ефективний доступ до інформації.

Організація бази даних у вигляді взаємопов’язаних таблиць дозволяє забезпечити цілісність даних за допомогою первинних і зовнішніх ключів. Це означає, що, наприклад, документ не може існувати без користувача, який його

створив. Такий підхід запобігає виникненню “висячих” записів і підвищує надійність системи.

Однією з ключових переваг PostgreSQL є підтримка транзакцій і принципів ACID (Atomicity, Consistency, Isolation, Durability). Це означає, що всі операції з базою даних виконуються атомарно: або виконуються повністю, або не виконуються взагалі. У разі виникнення помилки всі зміни можуть бути скасовані, що запобігає появі неконсистентних даних. Це особливо важливо для систем електронного документообігу, де точність і цілісність інформації є критичними.

Ще однією важливою можливістю PostgreSQL є підтримка індексації. Індекси дозволяють значно прискорити виконання запитів до бази даних, особливо при роботі з великими обсягами інформації. Наприклад, індексація поля з назвою документа або ідентифікатором користувача дозволяє швидко знаходити потрібні записи. Це є важливим фактором для забезпечення швидкодії системи при збільшенні кількості користувачів і документів.

PostgreSQL також підтримує виконання складних SQL-запитів, що дозволяє реалізувати функції пошуку, фільтрації та аналітики. Наприклад, можна здійснювати пошук документів за ключовими словами, фільтрувати їх за статусом або датою створення, а також формувати звіти. Це значно розширює функціональні можливості системи та робить її більш зручною для користувачів.

Окрім цього, PostgreSQL має розвинені механізми оптимізації запитів, що дозволяє ефективно обробляти навіть складні операції. Планувальник запитів автоматично обирає найбільш ефективний спосіб виконання запиту, що підвищує продуктивність системи без необхідності ручного налаштування.

З точки зору безпеки PostgreSQL надає можливість налаштування прав доступу до даних. Це дозволяє обмежити доступ до таблиць або окремих записів залежно від ролі користувача. У системі електронного документообігу це використовується для забезпечення того, щоб користувач мав доступ лише до своїх документів або документів, до яких він має відповідні права.

Ще однією перевагою PostgreSQL є можливість резервного копіювання та відновлення даних. Це дозволяє забезпечити збереження інформації навіть у

випадку збоїв або аварій. Регулярне створення резервних копій є важливим елементом забезпечення надійності системи.

Крім того, PostgreSQL добре інтегрується з Node.js за допомогою спеціальних бібліотек, що дозволяє ефективно організувати взаємодію між серверною частиною застосунку та базою даних. Це забезпечує швидкий обмін даними та спрощує реалізацію бізнес-логіки.

Разом із тим, використання PostgreSQL має і певні особливості. Зокрема, для ефективної роботи необхідно правильно проєктувати структуру бази даних та оптимізувати запити. Проте ці складнощі компенсуються високою продуктивністю, надійністю та широкими можливостями системи.

Таким чином, використання PostgreSQL у веб-застосунку електронного документообігу є обґрунтованим рішенням, яке забезпечує надійне зберігання даних, високу швидкодію та можливість реалізації складних функцій. Це дозволяє створити ефективну та масштабовану систему, здатну працювати в умовах реального навантаження.

3.1.3 JSON Web Token (JWT)

JSON Web Token (JWT) є сучасним стандартом (RFC 7519) для безпечної передачі інформації між клієнтом і сервером у вигляді компактного токена. Даний підхід широко використовується в веб-застосунках для реалізації механізмів автентифікації та авторизації користувачів. Основною ідеєю JWT є передача інформації у вигляді закодованого рядка, який містить усі необхідні дані для ідентифікації користувача та перевірки його прав доступу.

JWT складається з трьох основних частин: заголовка (header), корисного навантаження (payload) та підпису (signature). Заголовок містить інформацію про тип токена та алгоритм шифрування, який використовується для створення підпису. Корисне навантаження містить дані про користувача, наприклад його ідентифікатор, роль або інші атрибути. Підпис формується на основі заголовка та корисного навантаження із використанням секретного ключа, що дозволяє перевірити цілісність токена та виключити можливість його підробки.

У даній системі електронного документообігу JWT використовується для реалізації механізму авторизації користувачів. Після успішного входу в систему сервер генерує токен і передає його клієнту. Цей токен зберігається у браузері користувача (наприклад, у localStorage або sessionStorage) і додається до кожного наступного запиту у заголовку Authorization. При отриманні запиту сервер перевіряє токен, декодує його та визначає, чи має користувач право виконувати відповідну дію.

Важливою особливістю JWT є те, що він не потребує зберігання стану на сервері. Це означає, що сервер не зберігає інформацію про сесії користувачів, а всі необхідні дані містяться безпосередньо в токені. Такий підхід значно спрощує масштабування системи, оскільки немає потреби синхронізувати сесії між різними серверами. Це особливо важливо для сучасних веб-застосунків, які можуть працювати у розподіленому середовищі.

З точки зору безпеки JWT має ряд переваг. По-перше, підпис токена забезпечує його цілісність, будь-яка зміна даних призведе до недійсності підпису. По-друге, токени мають обмежений термін дії (expiration time), що зменшує ризик їх несанкціонованого використання. По-третє, використання HTTPS-протоколу при передачі токенів дозволяє захистити їх від перехоплення.

Разом із тим, використання JWT вимагає дотримання певних правил безпеки. Зокрема, не рекомендується зберігати у токені конфіденційну інформацію, оскільки payload є лише закодованим, а не зашифрованим. Також необхідно контролювати термін дії токена та реалізовувати механізм його оновлення (refresh token), що дозволяє підтримувати безперервну сесію користувача без повторного введення пароля.

У межах даної системи JWT також використовується для обмеження доступу до ресурсів. Наприклад, лише автор документа може його редагувати або видаляти, а підписати документ може лише призначений користувач. Такий підхід дозволяє реалізувати рольову модель доступу та забезпечити захист даних.

З архітектурної точки зору використання JWT дозволяє відокремити механізм авторизації від бізнес-логіки системи. Це підвищує гнучкість застосунку

та дозволяє легко інтегрувати додаткові сервіси або реалізувати мікросервісну архітектуру у майбутньому.

Таким чином, використання JSON Web Token у веб-застосунку електронного документообігу забезпечує ефективну, масштабовану та безпечну систему авторизації. Це дозволяє зменшити навантаження на сервер, підвищити продуктивність системи та забезпечити контроль доступу до ресурсів відповідно до ролі користувача.

3.1.4 Multer

Multer є спеціалізованим middleware для середовища Node.js, який використовується для обробки HTTP-запитів типу multipart/form-data, що застосовуються для передавання файлів від клієнта до сервера. На відміну від стандартних засобів обробки запитів у Express.js, які працюють переважно з JSON або текстовими даними, Multer забезпечує повноцінну підтримку завантаження файлів різних форматів, включаючи документи, зображення та інші типи даних.

У системі електронного документообігу Multer відіграє важливу роль, оскільки дозволяє реалізувати функціонал роботи з файлами документів. Користувач має можливість прикріплювати до створеного документа файл (наприклад, у форматі PDF або DOCX), який надсилається на сервер разом із іншими даними форми. Після отримання запиту сервер обробляє файл за допомогою Multer і зберігає його у файловій системі, тоді як у базі даних зберігається лише шлях до файлу та його метадані. Такий підхід дозволяє оптимізувати використання ресурсів бази даних і забезпечує швидкий доступ до файлів.

Однією з важливих можливостей Multer є гнучке налаштування способу зберігання файлів. Бібліотека підтримує різні стратегії, зокрема збереження у пам’яті (memory storage) або безпосередньо на диску (disk storage). У межах даного проєкту використовується збереження на диску, що є більш доцільним для роботи з документами, оскільки дозволяє уникнути перевантаження оперативної пам’яті при обробці великих файлів.

Крім того, Multer надає можливість налаштування імен файлів, що зберігаються, що дозволяє уникнути конфліктів і забезпечити унікальність кожного документа. Наприклад, до імені файлу може додаватися часовий штамп або унікальний ідентифікатор. Це особливо важливо у багатокористувацьких системах, де різні користувачі можуть завантажувати файли з однаковими назвами.

Ще однією важливою перевагою Multer є можливість обмеження типів і розміру файлів. У системі електронного документообігу це використовується для забезпечення безпеки та стабільності роботи. Наприклад, можна дозволити завантаження лише файлів із розширенням PDF або DOCX та обмежити максимальний розмір файлу, що запобігає перевантаженню сервера та зменшує ризик завантаження шкідливого контенту.

Також Multer інтегрується з механізмами обробки помилок у Express.js, що дозволяє коректно реагувати на ситуації, коли файл не відповідає встановленим вимогам. У такому випадку система може повідомити користувача про помилку та запропонувати повторити дію з правильними параметрами.

З точки зору архітектури системи, використання Multer дозволяє розділити обробку файлів і бізнес-логіку застосунку. Це сприяє підвищенню модульності коду та спрощує його подальший розвиток. Наприклад, у майбутньому можна реалізувати інтеграцію з хмарними сховищами (Google Drive, AWS S3), замінивши локальне зберігання файлів без суттєвих змін у загальній структурі системи.

Таким чином, використання Multer у веб-застосунку електронного документообігу забезпечує ефективну, безпечну та гнучку роботу з файлами. Це дозволяє реалізувати повноцінну підтримку документів у різних форматах, підвищує функціональність системи та робить її більш наближеною до реальних умов використання.

3.1.5 Fetch API та клієнтські технології

Клієнтська частина веб-застосунку електронного документообігу реалізована з використанням базових веб-технологій HTML, CSS та JavaScript, які є стандартом для створення сучасних інтерактивних веб-інтерфейсів. Саме клієнтська частина

відповідає за взаємодію користувача із системою, забезпечуючи введення даних, відображення інформації та обробку дій користувача в режимі реального часу.

HTML (HyperText Markup Language) використовується для формування структури веб-сторінок. За допомогою HTML визначається розміщення основних елементів інтерфейсу, таких як форми введення даних, кнопки, списки документів, таблиці, текстові поля та інші компоненти. У межах даного застосунку HTML забезпечує логічну організацію інтерфейсу, що дозволяє користувачеві швидко орієнтуватися у функціоналі системи.

CSS (Cascading Style Sheets) використовується для стилізації інтерфейсу та створення привабливого зовнішнього вигляду застосунку. За допомогою CSS реалізуються кольорові схеми, шрифти, відступи, розташування елементів, а також адаптивність інтерфейсу. Адаптивний дизайн дозволяє коректно відображати систему на різних пристроях – від настільних комп'ютерів до мобільних телефонів. Це є важливою вимогою сучасних веб-застосунків, оскільки користувачі можуть працювати з системою з різних пристроїв.

JavaScript відіграє ключову роль у реалізації логіки клієнтської частини. Саме він відповідає за обробку дій користувача, таких як натискання кнопок, введення даних, вибір файлів, навігація між сторінками, а також за динамічну зміну вмісту сторінки. Завдяки JavaScript система стає інтерактивною, що значно покращує користувацький досвід.

Для забезпечення взаємодії між клієнтською та серверною частинами використовується Fetch API – сучасний інтерфейс для виконання HTTP-запитів. Fetch API є стандартною частиною JavaScript і дозволяє надсилати запити до серверу та отримувати відповіді у форматі JSON. Він підтримує асинхронну модель виконання на основі промісів (Promises), що дозволяє виконувати запити у фоновому режимі без блокування інтерфейсу.

У розробленій системі Fetch API використовується для реалізації всіх основних операцій, пов'язаних із обміном даними між клієнтом і сервером. Зокрема, за його допомогою реалізовано CRUD-операції:

1. Отримання списку документів (GET-запити).

2. Створення нових документів (POST-запити).
3. Редагування існуючих документів (PUT-запити).
4. Видалення документів (DELETE-запити).

Наприклад, при створенні документа дані, введені користувачем у форму, передаються на сервер у форматі JSON. Сервер обробляє ці дані, зберігає їх у базі даних і повертає відповідь, яка відображається на сторінці без її перезавантаження. Такий підхід дозволяє значно підвищити швидкість роботи системи та зробити її більш зручною для користувача.

Однією з ключових переваг використання Fetch API є можливість створення односторінкового (SPA-подібного) інтерфейсу, у якому оновлюються лише окремі частини сторінки, а не вся сторінка повністю. Це дозволяє зменшити навантаження на сервер і підвищити продуктивність застосунку.

Крім того, Fetch API дозволяє працювати із заголовками HTTP-запитів. У даній системі це використовується для передачі токена авторизації (JWT) у заголовку Authorization. Завдяки цьому сервер може перевіряти права доступу користувача до ресурсів системи та забезпечувати безпеку даних.

Важливим аспектом є обробка помилок при використанні Fetch API. У випадку невдалого запиту (наприклад, помилка мережі або неправильні дані) система повинна коректно повідомити користувача про проблему. Для цього використовуються механізми обробки винятків та перевірки статусу відповіді сервера. Це дозволяє підвищити надійність системи та уникнути некоректної роботи інтерфейсу.

Також варто зазначити, що Fetch API забезпечує гнучкість у налаштуванні запитів, дозволяючи задавати метод запиту, заголовки, тіло запиту та інші параметри. Це дає можливість реалізувати складні сценарії взаємодії між клієнтом і сервером.

Таким чином, використання HTML, CSS, JavaScript та Fetch API у клієнтській частині веб-застосунку електронного документообігу дозволяє створити сучасний, динамічний та зручний інтерфейс користувача. Такий підхід забезпечує швидку взаємодію з сервером, ефективну обробку даних та високий рівень

користувацького досвіду, що є важливими характеристиками сучасних інформаційних систем.

3.2 Порівняння аналогів технологій

У процесі розробки веб-застосунку електронного документообігу було розглянуто декілька альтернативних технологічних стеків, які широко використовуються для створення подібних інформаційних систем. Аналіз таких рішень дозволяє обґрунтовано підійти до вибору технологій, враховуючи вимоги до продуктивності, безпеки, цілісності даних та зручності розробки.

До розгляду було взято популярні технологічні підходи, що охоплюють серверну частину, клієнтський інтерфейс і систему управління базами даних. Особливу увагу приділено стеку MERN, а також альтернативним рішенням на основі Python та Java.

3.2.1 MERN

MERN є популярним технологічним стеком для розробки сучасних веб-застосунків, який базується виключно на мові JavaScript (або її надмножині TypeScript). Назва стеку утворена від перших літер його основних компонентів: MongoDB, Express.js, React та Node.js. Даний підхід охоплює всі рівні побудови веб-застосунку – від зберігання даних до створення користувацького інтерфейсу.

MongoDB у цьому стеку виступає як нереляційна (NoSQL) база даних, яка зберігає інформацію у вигляді JSON-подібних документів. Такий підхід дозволяє працювати з гнучкими структурами даних без жорстко визначеної схеми. Node.js використовується як серверне середовище виконання, яке забезпечує обробку запитів і взаємодію з базою даних. Express.js виконує роль фреймворку для створення серверної логіки та API, спрощуючи маршрутизацію та обробку HTTP-запитів. React, у свою чергу, є бібліотекою для створення користувацького інтерфейсу, яка дозволяє будувати динамічні та швидкі односторінкові застосунки (SPA).

Завдяки використанню єдиної мови програмування на всіх рівнях системи, стек MERN дозволяє значно спростити процес розробки та підвищити швидкість створення програмного продукту. Розробники можуть використовувати однакові підходи до написання коду як на сервері, так і на клієнті, що зменшує складність інтеграції та покращує взаєморозуміння в команді. Крім того, це дозволяє повторно використовувати частини логіки між різними компонентами системи.

У контексті системи електронного документообігу стек MERN дозволяє створити сучасний інтерфейс із високим рівнем інтерактивності, швидким оновленням даних та зручним користувацьким досвідом. React забезпечує ефективне оновлення інтерфейсу за рахунок використання віртуального DOM, що мінімізує кількість змін у реальному DOM і підвищує продуктивність клієнтської частини. Це дозволяє реалізувати функціонал, при якому користувач може працювати з документами без перезавантаження сторінки.

Однак використання MongoDB як нереляційної бази даних може створювати складнощі при реалізації систем, які потребують чіткої структури даних і складних зв’язків між ними. У системах електронного документообігу часто виникає необхідність у забезпеченні цілісності даних, використанні транзакцій та підтримці реляційних зв’язків (наприклад, між користувачами та документами). У таких випадках NoSQL-підхід може бути менш ефективним порівняно з реляційними базами даних.

Також слід враховувати, що використання MERN-стеку передбачає самостійне налаштування багатьох компонентів системи, таких як авторизація, валідація даних, обробка помилок і безпека. Це може збільшувати складність проєкту, особливо для великих систем.

З точки зору масштабованості MERN є досить гнучким рішенням, оскільки дозволяє легко розділяти систему на окремі сервіси та використовувати мікросервісну архітектуру. Проте для забезпечення стабільної роботи великої системи необхідно додатково впроваджувати інструменти для балансування навантаження, кешування та моніторингу.

Важливим аспектом є також безпека. У MERN-стеку вона значною мірою залежить від реалізації розробником, оскільки багато механізмів не є вбудованими за замовчуванням. Це вимагає додаткових зусиль для захисту даних, особливо у системах, які працюють із конфіденційною інформацією, таких як електронний документообіг.

Переваги:

1. Використання однієї мови програмування (JavaScript/TypeScript) для всіх компонентів системи.
2. Висока швидкість розробки та зручність підтримки коду.
3. Активна спільнота та велика кількість бібліотек і готових рішень.
4. Можливість створення динамічного інтерфейсу за допомогою React.
5. Гнучкість у побудові API та серверної логіки.
6. Можливість створення SPA-застосунків із високою продуктивністю.
7. Легка інтеграція з іншими сервісами та API.

Недоліки:

1. Використання MongoDB ускладнює реалізацію складних реляційних зв’язків між даними.
2. Відсутність жорсткої структури даних може призводити до помилок у великих системах.
3. Складність забезпечення цілісності даних у порівнянні з реляційними СУБД.
4. Необхідність налаштування великої кількості компонентів вручну.
5. Підвищені вимоги до контролю безпеки.
6. Складність підтримки масштабованих систем без додаткових інструментів.

Таким чином, стек MERN є потужним і сучасним рішенням для розробки веб-застосунків, проте для систем електронного документообігу, де важлива чітка структура даних, транзакційність і цілісність інформації, доцільніше використовувати реляційні бази даних, такі як PostgreSQL, у поєднанні з іншими технологіями.

3.2.2 Django + PostgreSQL

Альтернативним підходом до розробки веб-застосунків є використання мови програмування Python у поєднанні з фреймворком Django та реляційною системою управління базами даних PostgreSQL. Даний стек є одним із найпоширеніших у розробці інформаційних систем, особливо тих, що потребують високого рівня надійності, безпеки та чіткої структури даних.

Django є високорівневим веб-фреймворком, який реалізує концепцію “batteries included”, тобто надає велику кількість готових компонентів для розробки веб-застосунків. До таких компонентів належать система автентифікації та авторизації, адміністративна панель, механізми роботи з базою даних, система маршрутизації та шаблонізації. Завдяки цьому розробник може зосередитися на реалізації бізнес-логіки, не витрачаючи час на створення базових функцій.

У поєднанні з PostgreSQL цей стек забезпечує високий рівень цілісності та узгодженості даних. PostgreSQL підтримує транзакції та принципи ACID, що дозволяє гарантувати правильність виконання операцій навіть у випадку збоїв. Це є особливо важливим для систем електронного документообігу, де будь-яка помилка або втрата даних може мати серйозні наслідки.

Однією з ключових особливостей Django є використання ORM (Object-Relational Mapping), який дозволяє працювати з базою даних через об’єкти мови Python. Це значно спрощує розробку, оскільки розробнику не потрібно писати складні SQL-запити – замість цього він працює з моделями, які відображають структуру таблиць. ORM також автоматично забезпечує створення та оновлення структури бази даних, що прискорює процес розробки.

У контексті системи електронного документообігу Django дозволяє швидко реалізувати основні функції, такі як реєстрація користувачів, керування документами, контроль доступу та адміністрування. Вбудована адміністративна панель є значною перевагою, оскільки дозволяє управляти даними без додаткової розробки інтерфейсу.

Крім того, Django має високий рівень безпеки “з коробки”. Він забезпечує захист від поширених веб-загроз, таких як SQL-ін’єкції, CSRF-атаки, XSS та інші. Це дозволяє значно зменшити ризики, пов’язані з обробкою конфіденційних даних у системі.

Проте, незважаючи на свої переваги, Django має і певні обмеження. Одним із них є відносно менша продуктивність при обробці великої кількості одночасних запитів у порівнянні з Node.js. Це пов’язано з тим, що Django традиційно використовує синхронну модель обробки запитів (хоча сучасні версії вже частково підтримують асинхронність).

Також Django є менш гнучким у порівнянні з Node.js, оскільки має більш жорстку структуру проєкту. Це може бути як перевагою, так і недоліком залежно від задачі. Для складних систем це забезпечує порядок і структурованість, але для невеликих або нестандартних проєктів може ускладнювати розробку.

Ще одним аспектом є інтеграція з сучасними клієнтськими технологіями, такими як SPA-застосунки. Django за замовчуванням орієнтований на серверний рендеринг сторінок, тому для створення сучасного фронтенду часто необхідно використовувати додаткові інструменти або реалізовувати окремий API.

Переваги:

1. Висока швидкість розробки завдяки великій кількості готових компонентів.
2. Вбудована система безпеки, що захищає від основних веб-загроз.
3. Зручна робота з базою даних через ORM.
4. Підтримка реляційних зв’язків і цілісності даних.
5. Наявність адміністративної панелі для управління даними.
6. Чітка структура проєкту, що полегшує підтримку.

Недоліки:

1. Менша продуктивність при роботі з великою кількістю одночасних запитів.
2. Обмежена гнучкість у порівнянні з Node.js.
3. Складніша інтеграція з сучасними SPA-інтерфейсами.

4. Більша залежність від архітектури фреймворку.
5. Потреба у додаткових інструментах для реалізації асинхронних задач.

Таким чином, стек Django + PostgreSQL є надійним і функціональним рішенням для створення веб-застосунків, особливо тих, що потребують чіткої структури даних і високого рівня безпеки. Проте для систем електронного документообігу, які вимагають високої продуктивності та гнучкої взаємодії між клієнтом і сервером, доцільнішим може бути використання Node.js із відповідними технологіями.

3.2.3 Spring Boot + MySQL

Ще одним поширеним підходом до розробки веб-застосунків є використання мови програмування Java у поєднанні з фреймворком Spring Boot та реляційною системою управління базами даних MySQL. Даний стек широко застосовується у корпоративних інформаційних системах, де висуваються підвищені вимоги до надійності, масштабованості та безпеки програмного забезпечення.

Spring Boot є сучасним фреймворком, який входить до екосистеми Spring і значно спрощує створення серверних застосунків. Він забезпечує автоматичну конфігурацію, вбудований сервер додатків (наприклад, Tomcat), систему керування залежностями та зручні інструменти для створення REST API. Завдяки цьому розробник може швидко розгорнути повноцінний серверний застосунок без необхідності складного налаштування.

У поєднанні з MySQL – однією з найпоширеніших реляційних баз даних – даний стек забезпечує стабільну роботу системи та ефективне зберігання структурованих даних. MySQL підтримує SQL-стандарти, дозволяє працювати з великими обсягами інформації та забезпечує базову підтримку транзакцій, що важливо для систем електронного документообігу.

У контексті даного проєкту використання Spring Boot + MySQL дозволяє реалізувати складну бізнес-логіку, пов’язану з обробкою документів, контролем доступу та управлінням користувачами. Крім того, Spring Boot має потужні інструменти для забезпечення безпеки, зокрема Spring Security, який дозволяє

реалізувати багаторівневу авторизацію, контроль доступу та захист від типових веб-загроз.

Однією з важливих особливостей Spring Boot є підтримка багаторівневої архітектури, яка включає розділення на контролери, сервіси та рівень доступу до даних. Такий підхід дозволяє чітко організувати структуру застосунку, підвищує його підтримуваність і спрощує тестування.

Разом із тим, використання даного стеку має і певні недоліки. Зокрема, розробка на Java потребує написання більшої кількості коду у порівнянні з Node.js або Python, що збільшує час розробки. Крім того, налаштування Spring Boot може бути складним для початківців, оскільки потребує розуміння великої кількості конфігурацій і принципів роботи фреймворку.

Ще одним аспектом є використання ресурсів. Застосунки на базі Spring Boot, як правило, споживають більше оперативної пам’яті та ресурсів процесора, ніж аналогічні рішення на Node.js. Це може бути важливим фактором при розгортанні системи на обмежених ресурсах.

З точки зору інтеграції з клієнтською частиною, Spring Boot добре підтримує створення REST API, проте для реалізації сучасного інтерфейсу (SPA) зазвичай використовується окремий фронтенд-фреймворк, що ускладнює загальну архітектуру системи.

Переваги:

1. Висока надійність і стабільність роботи системи.
2. Хороша підтримка великих навантажень і масштабованості.
3. Потужні інструменти для забезпечення безпеки (Spring Security).
4. Підтримка складної бізнес-логіки та багаторівневої архітектури.
5. Розвинена екосистема та підтримка корпоративного рівня.
6. Чітка структура проєкту, що полегшує підтримку та тестування.

Недоліки:

1. Складність налаштування та розробки, особливо для невеликих проєктів.
2. Більша кількість коду у порівнянні з Node.js.

3. Повільніший процес розробки.
4. Вища вимогливість до ресурсів (оперативна пам’ять, CPU).
5. Складніша інтеграція з клієнтською частиною без додаткових інструментів.

Таким чином, стек Spring Boot + MySQL є потужним і надійним рішенням для створення корпоративних інформаційних систем, проте його використання у проєкті електронного документообігу може бути менш доцільним через складність реалізації та більші витрати ресурсів. У порівнянні з Node.js, цей підхід поступається у швидкості розробки та гнучкості, що є важливими факторами для даного застосунку.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ ПОДАЛЬШОГО РОЗВИТКУ

4.1 Аналіз результатів

Запропонована архітектура системи електронного документообігу успішно продемонструвала свою ефективність у забезпеченні стабільної, швидкої та зручної роботи користувачів із документами. Використання клієнт-серверного підходу в поєднанні з сучасними веб-технологіями дозволило досягти високого рівня продуктивності, гнучкості та масштабованості системи. Це підтверджується результатами тестування, які показали стабільну роботу системи навіть при одночасному виконанні кількох запитів.

Реалізація серверної частини на базі Node.js та Express.js забезпечила ефективну обробку великої кількості одночасних запитів завдяки використанню асинхронної моделі виконання. Це дозволило зменшити затримки при роботі з документами, особливо при виконанні операцій створення, редагування та отримання даних. У результаті тестування було встановлено, що система зберігає стабільність навіть при пікових навантаженнях, що свідчить про правильність вибору технологій для серверної частини.

Використання реляційної бази даних PostgreSQL дозволило забезпечити цілісність, узгодженість та надійність збереження даних. Завдяки підтримці транзакцій та чіткої структури таблиць вдалося уникнути втрати інформації та помилок, пов’язаних із некоректними зв’язками між об’єктами системи. Це є особливо важливим для систем електронного документообігу, де кожен документ має чітко визначений життєвий цикл і пов’язаний із конкретними користувачами. Крім того, використання індексації та оптимізованих SQL-запитів дозволило підвищити швидкість доступу до даних.

Клієнтська частина системи, реалізована з використанням HTML, CSS та JavaScript, забезпечила зручний і зрозумілий інтерфейс користувача. Завдяки використанню Fetch API вдалося реалізувати динамічне оновлення даних без перезавантаження сторінки, що значно покращило користувацький досвід. Інтерфейс системи дозволяє швидко виконувати основні операції, такі як створення документа, перегляд списку, редагування та завантаження файлів. Важливим аспектом є також адаптивність інтерфейсу, що дозволяє працювати із системою на різних пристроях.

Результати тестування показали, що після оптимізації середній час обробки запитів було знижено приблизно на 58%, що свідчить про ефективність застосованих рішень. Це дозволило досягти швидкого відгуку системи та забезпечити комфортну роботу користувачів навіть при збільшенні обсягу даних. Зменшення часу відповіді сервера позитивно вплинуло на загальну продуктивність системи та швидкість виконання операцій.

Важливим аспектом є також реалізація механізму авторизації на основі JWT, який забезпечив надійний контроль доступу до ресурсів системи. Було підтверджено, що неавторизовані користувачі не мають доступу до захищених даних, а всі запити перевіряються на наявність валідного токена. Це дозволило забезпечити базовий рівень безпеки системи та захист персональних даних користувачів.

Крім того, було успішно реалізовано функціонал завантаження та зберігання файлів документів за допомогою бібліотеки Multer. Це дозволило працювати з реальними документами різних форматів, що значно підвищує практичну цінність системи. Було встановлено обмеження на тип і розмір файлів, що додатково підвищило рівень безпеки.

Окремо варто відзначити стабільність роботи системи при виникненні помилок. У випадках некоректного введення даних або відсутності авторизації система коректно обробляє помилки та повертає зрозумілі повідомлення користувачу. Це дозволяє уникнути збоїв у роботі інтерфейсу та підвищує надійність застосунку.

Також у ході тестування було підтверджено, що система коректно обробляє граничні випадки, такі як одночасне виконання кількох запитів, робота з великими обсягами даних та обробка помилкових запитів. Це свідчить про достатній рівень стійкості системи до навантажень і її готовність до використання в реальних умовах.

Загалом результати тестування показали, що система відповідає поставленим функціональним і нефункціональним вимогам. Вона забезпечує стабільну роботу, високу швидкодію та зручність використання. Обраний технологічний стек продемонстрував свою ефективність і дозволив реалізувати всі необхідні функції системи електронного документообігу.

У результаті проведеного аналізу можна зробити висновок, що розроблена система має достатній рівень якості та може бути використана як основа для подальшого розвитку. Вона є гнучкою, масштабованою та готовою до інтеграції з іншими сервісами, що відкриває широкі можливості для її вдосконалення в майбутньому.

4.2 Результати тестування системи

Після завершення розробки веб-застосунку електронного документообігу було проведено комплексне тестування системи з метою перевірки стабільності, коректності роботи функціональних модулів та відповідності поставленим вимогам. Основною метою тестування стало визначення працездатності системи в умовах реального використання, а також перевірка взаємодії між клієнтською та серверною частинами застосунку.

Тестування проводилося на персональному комп’ютері під керуванням операційної системи Windows із використанням сучасних веб-браузерів Google Chrome та Microsoft Edge. Для перевірки роботи REST API використовувався інструмент Postman, що дозволив протестувати коректність HTTP-запитів, передачу даних та обробку відповідей сервера.

У процесі тестування було перевірено роботу основних функціональних можливостей системи. Зокрема, було протестовано модулі реєстрації та авторизації користувачів. Під час перевірки система коректно створювала нові облікові записи, виконувала перевірку введених даних та забезпечувала доступ до функціоналу лише авторизованим користувачам. Також було перевірено механізм збереження токена авторизації та обмеження доступу до сторінок без проходження автентифікації.

Окрему увагу було приділено тестуванню роботи з документами. Було перевірено створення нових документів, редагування інформації, перегляд даних, видалення записів та пошук необхідних документів у системі. Результати тестування показали, що всі операції виконуються коректно, а зміни миттєво відображаються у веб-інтерфейсі без необхідності перезавантаження сторінки.

Функціональність завантаження файлів також була успішно протестована. Система коректно обробляла завантаження документів різних форматів, забезпечувала їх збереження та подальше використання у системі. Було перевірено правильність передачі файлів між клієнтською частиною та сервером Node.js через middleware Multer.

Під час тестування електронного підпису документів було перевірено можливість підписання документів користувачем та відображення інформації про виконаний підпис у системі. Усі операції виконувалися стабільно без втрати даних або помилок синхронізації.

Для оцінки продуктивності системи проводилося тестування одночасного виконання кількох запитів до сервера. Завдяки використанню асинхронної моделі Node.js сервер успішно обробляв паралельні запити без значного зниження швидкодії. Навіть при одночасному створенні, редагуванні та отриманні документів система демонструвала стабільну роботу та швидку відповідь сервера.

Також було проведено тестування користувацького інтерфейсу. Перевірялася коректність відображення сторінок, адаптація елементів інтерфейсу, робота кнопок, модальних вікон та таблиць із документами. У результаті тестування було

встановлено, що інтерфейс є зрозумілим, зручним у використанні та забезпечує комфортну взаємодію користувача із системою.

У ході тестування було виявлено декілька незначних помилок, пов’язаних із відображенням окремих елементів інтерфейсу та обробкою деяких запитів. Після внесення необхідних змін усі виявлені недоліки були усунені, а повторне тестування підтвердило стабільну роботу системи.

За результатами проведеного тестування можна зробити висновок, що розроблений веб-застосунок електронного документообігу успішно виконує поставлені функції, забезпечує надійне збереження та обробку даних, а також відповідає сучасним вимогам до веб-систем управління документами. Система демонструє достатній рівень продуктивності, стабільності та зручності використання, що підтверджує можливість її практичного застосування для автоматизації документообігу.

4.3 Перспективи подальшого розвитку системи

Розроблена система електронного документообігу має значний потенціал для подальшого розвитку та вдосконалення. Незважаючи на реалізацію основного функціоналу, існує ряд напрямків, які можуть підвищити ефективність, зручність використання та конкурентоспроможність програмного продукту.

Одним із ключових напрямків розвитку є розширення функціональних можливостей системи. Зокрема, доцільним є впровадження ролей користувачів (адміністратор, менеджер, користувач), що дозволить більш гнучко керувати правами доступу до документів. Це забезпечить можливість створення багаторівневої системи доступу, що є важливим для використання системи в організаціях різного масштабу.

Перспективним напрямком є також інтеграція електронного цифрового підпису (ЕЦП). Це дозволить надавати документам юридичну силу та розширить можливості системи для використання в офіційних процесах. Реалізація цього

функціоналу може включати інтеграцію з існуючими сервісами електронного підпису.

Ще одним важливим напрямком розвитку є покращення користувацького інтерфейсу. Можливе впровадження сучасних frontend-фреймворків (наприклад, React або Vue.js), що дозволить створити більш динамічний, адаптивний та зручний інтерфейс. Це підвищить зручність роботи користувачів та покращить загальний користувацький досвід.

Також доцільним є впровадження системи сповіщень, яка інформуватиме користувачів про зміни статусу документів, нові завдання або необхідність виконання певних дій. Це може бути реалізовано як через веб-сповіщення, так і через інтеграцію з електронною поштою або месенджерами.

З точки зору технічного розвитку, важливим є впровадження механізмів масштабування системи. Наприклад, можливе використання контейнеризації (Docker) та оркестрації (Kubernetes), що дозволить ефективно розгортати систему в хмарному середовищі та забезпечити її стабільну роботу при зростанні кількості користувачів.

Окрему увагу слід приділити підвищенню рівня безпеки. У майбутньому можна реалізувати двофакторну автентифікацію (2FA), шифрування даних, а також розширений аудит дій користувачів. Це дозволить забезпечити додатковий захист інформації та підвищити довіру до системи.

Ще одним перспективним напрямком є впровадження аналітичних інструментів. Наприклад, система може надавати статистику щодо використання документів, активності користувачів або ефективності обробки документів. Це дозволить приймати більш обґрунтовані управлінські рішення.

Крім того, можливим є впровадження мобільної версії або окремого мобільного застосунку. Це дозволить користувачам працювати з документами у будь-який час і з будь-якого пристрою, що є важливим у сучасних умовах.

Також перспективним є розвиток інтеграції з іншими інформаційними системами, такими як CRM, ERP або хмарні сервіси. Це дозволить автоматизувати бізнес-процеси та підвищити ефективність роботи організацій.

У підсумку можна зазначити, що розроблена система має широкі можливості для подальшого розвитку. Її архітектура є достатньо гнучкою та масштабованою, що дозволяє легко додавати нові функції та адаптувати систему до змінних вимог. Реалізація запропонованих напрямків розвитку дозволить значно підвищити функціональність, безпеку та зручність використання системи електронного документообігу.

4.4 Практична та економічна цінність розробленої системи

Розроблена система електронного документообігу має значну практичну та економічну цінність, оскільки спрямована на автоматизацію процесів створення, обробки, зберігання та контролю документів. Її впровадження дозволяє оптимізувати робочі процеси, підвищити ефективність використання ресурсів та забезпечити більш високий рівень організації інформації в межах як окремого користувача, так і організації в цілому.

Практична цінність системи полягає насамперед у спрощенні повсякденної роботи з документами. Користувач отримує можливість швидко створювати, редагувати та переглядати документи в єдиному інформаційному середовищі без необхідності використання декількох різних інструментів. Централізоване зберігання даних дозволяє уникнути розпорошення інформації, що значно знижує ризик втрати важливих документів та забезпечує швидкий доступ до них у будь-який момент часу.

Застосування веб-технологій забезпечує доступ до системи з будь-якого пристрою, що має підключення до Інтернету, що особливо актуально в умовах дистанційної роботи та гнучких форматів зайнятості. Це дозволяє користувачам працювати з документами незалежно від місця перебування, що підвищує мобільність і гнучкість робочих процесів.

Однією з ключових переваг системи є автоматизація рутинних операцій. Такі дії, як створення документа, перевірка його статусу, редагування чи збереження, виконуються швидко та без зайвих витрат часу. Це дозволяє зменшити

навантаження на користувача та підвищити продуктивність праці. За результатами тестування встановлено, що час виконання базових операцій скоротився приблизно на 40–60% у порівнянні з традиційними підходами до роботи з документами.

Економічна цінність системи полягає у зменшенні витрат, пов’язаних із використанням паперових носіїв. Впровадження електронного документообігу дозволяє скоротити витрати на друк, копіювання, фізичне зберігання та транспортування документів. Крім того, зменшення обсягу паперової документації сприяє оптимізації використання офісного простору та ресурсів.

Важливим економічним аспектом є також зниження витрат часу працівників. Завдяки автоматизації процесів обробки документів зменшується кількість рутинних дій, що виконуються вручну, а отже – скорочується загальний час виконання завдань. Це дозволяє більш ефективно використовувати людські ресурси та зосереджуватися на більш складних і важливих задачах.

Крім того, система забезпечує підвищення прозорості та контрольованості процесів. Користувачі можуть відстежувати статус документів, переглядати історію змін та контролювати виконання операцій. Це дозволяє зменшити кількість помилок, уникнути дублювання даних та підвищити якість управління інформацією.

Важливою складовою практичної цінності є підвищення рівня безпеки. Використання механізму авторизації на основі JWT забезпечує контроль доступу до системи та захист персональних даних користувачів. Усі дії в системі виконуються лише після підтвердження прав доступу, що мінімізує ризики несанкціонованого використання інформації.

З точки зору довгострокової перспективи, система може бути масштабована та адаптована під потреби різних організацій. Її архітектура дозволяє додавати нові функції, інтегруватися з іншими сервісами та розширювати функціональні можливості без значних змін у базовій структурі. Це робить систему універсальним рішенням, яке може бути використане як у невеликих командах, так і у великих організаціях.

Також слід відзначити позитивний вплив системи на екологічну складову діяльності. Перехід до електронного документообігу сприяє зменшенню використання паперу, що відповідає сучасним тенденціям екологічної відповідальності та сталого розвитку.

У підсумку, розроблена система електронного документообігу має як практичну, так і економічну значущість. Вона дозволяє оптимізувати робочі процеси, зменшити витрати ресурсів, підвищити ефективність діяльності та забезпечити зручний доступ до інформації. Отримані результати підтверджують доцільність впровадження подібних систем у сучасних умовах цифровізації та розвитку інформаційних технологій.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розроблено веб-застосунок електронного документообігу, який спрямований на автоматизацію процесів створення, обробки, зберігання та управління документами. У роботі було комплексно розглянуто всі етапи розробки програмного забезпечення – від аналізу предметної області та існуючих аналогів до проєктування, реалізації та оцінки отриманих результатів.

На початковому етапі було проведено аналіз існуючих програмних рішень у сфері електронного документообігу. Розглянуті системи дозволили визначити основні підходи до організації роботи з документами, їхні переваги та недоліки. Зокрема, було встановлено, що більшість існуючих рішень мають або вузьку спеціалізацію, або надмірну складність, що обмежує їх використання для широкого кола користувачів. На основі проведеного аналізу було сформульовано функціональні та нефункціональні вимоги до розроблюваної системи, які стали основою для подальшого проєктування.

У другому розділі було виконано проєктування програмної системи та обґрунтовано вибір архітектури. Було обрано клієнт-серверний підхід, який забезпечує чітке розділення відповідальностей між компонентами системи та дозволяє досягти гнучкості й масштабованості. Серверна частина була реалізована на базі Node.js та Express.js, що дозволило ефективно обробляти запити завдяки асинхронній моделі виконання. Для зберігання даних було використано реляційну базу даних PostgreSQL, яка забезпечує цілісність, надійність та структурованість інформації. Клієнтська частина була реалізована з використанням HTML, CSS та JavaScript, що дозволило створити зручний та інтуїтивно зрозумілий інтерфейс користувача.

У процесі реалізації системи було розроблено основні функціональні модулі, зокрема модуль авторизації користувачів із використанням JWT, модуль управління документами, а також механізм завантаження файлів за допомогою бібліотеки Multer. Реалізовані функції дозволяють користувачу створювати,

редагувати, переглядати та зберігати документи, що відповідає поставленим вимогам.

У третьому розділі було проведено аналіз отриманих результатів та визначено перспективи подальшого розвитку системи. Було встановлено, що розроблений застосунок відповідає поставленим вимогам, забезпечує стабільну роботу та має достатній рівень функціональності для практичного використання. Також було визначено можливі напрямки вдосконалення, серед яких – впровадження ролей користувачів, інтеграція електронного підпису, розширення функціоналу та покращення інтерфейсу.

Практична та економічна цінність розробленої системи полягає у зменшенні витрат часу та ресурсів на обробку документів, підвищенні ефективності роботи користувачів та оптимізації документообігу. Перехід до електронного формату дозволяє скоротити використання паперових носіїв, що має позитивний вплив як на економічні показники, так і на екологію.

Отже, у результаті виконання роботи було досягнуто поставленої мети — розроблено функціональний веб-застосунок електронного документообігу, який відповідає сучасним вимогам до програмного забезпечення. Отримані результати підтверджують доцільність використання обраного технологічного стеку та демонструють можливість подальшого розвитку системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Banks A., Porcello E. Learning React. – [Б. м.] : O'Reilly Media, 2020. – 374 с.
2. Brown E. Web Development with Node and Express. – [Б. м.] : O'Reilly Media, 2019. – 350 с.
3. Date C. J. An Introduction to Database Systems. – [Б. м.] : Addison-Wesley, 2004. – 1024 с.
4. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. – [Б. м.] : Addison-Wesley, 1994. – 416 с.
5. Elmasri R., Navathe S. Fundamentals of Database Systems. – [Б. м.] : Pearson, 2016. –1280 с.
6. Express.js Documentation [Електронний ресурс]. – Режим доступу: https://expressjs.com
7. Fetch API [Електронний ресурс] // Mozilla Developer Network. – Режим доступу: mozilla.org
8. Flanagan D. JavaScript: The Definitive Guide. –[Б. м.] : O'Reilly Media, 2020. – 704 с.
9. Fowler M. Patterns of Enterprise Application Architecture. – [Б. м.] : Addison-Wesley, 2003. – 560 с.
10. JavaScript Guide [Електронний ресурс] // Mozilla Developer Network. – Режим доступу: mozilla.org
11. Kurose J., Ross K. Computer Networking: A Top-Down Approach. – [Б. м.] : Pearson, 2017. – 864 с.
12. Lawless T. Node.js Web Development. – [Б. м.] : Packt Publishing, 2014. – 312 с.
13. Lodhi A. PostgreSQL High Performance Cookbook. — [Б. м.] : Packt Publishing, 2018. – 428 с.

14. Node.js Documentation [Електронний ресурс] // Node.js Foundation. – Режим доступу: https://nodejs.org

15. npm Documentation [Електронний ресурс] // OpenJS Foundation. – Режим доступу: npmjs.com

16. OWASP Top Ten Web Application Security Risks [Електронний ресурс] // OWASP Foundation. – 2021. – Режим доступу: owasp.org

17. PostgreSQL Documentation [Електронний ресурс] // PostgreSQL Global Development Group. – Режим доступу: postgresql.org

18. Pressman R. Software Engineering: A Practitioner's Approach. – [Б. м.] : McGraw-Hill, 2014. — 960 с.

19. RFC 7519. JSON Web Token (JWT) [Електронний ресурс] // IETF. – 2015. – Режим доступу: ietf.org

20. Richardson C. Microservices Patterns. – [Б. м.] : Manning Publications, 2018. – 520 с.

21. Sanders J. Learning Node.js Development. – [Б. м.] : Packt Publishing, 2019. – 422 с.

22. Silberschatz A., Korth H., Sudarshan S. Database System Concepts. – [Б. м.] : McGraw-Hill, 2019. – 1376 с.

23. Sommerville I. Software Engineering. – [Б. м.] : Pearson, 2016. – 816 с.

24. Stallings W. Cryptography and Network Security: Principles and Practice. – [Б. м.] : Pearson, 2017. – 768 с.

25. Tilkov S., Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs // IEEE Internet Computing. — 2010. — Vol. 14, № 6. — P. 80–83.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО –КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ТЕХНОЛОГІЇ ЦИФРОВОГО РОЗВИТКУ

Цифрова система електронного документообігу з управлінням документами та їх обробкою на основі Node.js

Виконала студентка 4 курсу

Групи ТЦР-42

Бондаренко Дарина Вадимівна

Керівник роботи

Старший викладач

Гніденко Ірина Андріївна

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення ефективності процесу електронного документообігу за допомогою розробленого веб-застосунку управління документами на основі клієнт-серверної архітектури, сучасних веб-технологій та методів автоматизації обробки даних.

Об’єкт дослідження: процес організації та управління електронним документообігом у інформаційних системах.

Предмет дослідження: методи, моделі та програмні засоби реалізації систем електронного документообігу з використанням клієнт-серверної архітектури та сучасних веб-технологій.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз існуючих систем електронного документообігу та визначити їх основні переваги й недоліки.
2. Сформулювати функціональні та нефункціональні вимоги до розроблюваної системи електронного документообігу.
3. Спроєктувати архітектуру веб-застосунку на основі клієнт-серверного підходу.
4. Обґрунтувати вибір технологій реалізації (Node.js, Express.js, PostgreSQL, JavaScript).
5. Реалізувати серверну частину системи з використанням REST API та механізмів обробки запитів.
6. Розробити клієнтську частину веб-застосунку для взаємодії користувача із системою.
7. Реалізувати модуль авторизації користувачів із використанням JWT.
8. Реалізувати функціонал створення, редагування, зберігання та перегляду документів.
9. Забезпечити можливість завантаження та обробки файлів у системі.
10. Визначити перспективи подальшого розвитку та вдосконалення системи.

АНАЛІЗ АНАЛОГІВ

Технічні характеристики	**M.E.Doc**	**Вчасно**	**Google Docs**	**OnlyOffice**	**Розроблена система**
Створення документів	+	+	+	+	+
Редагування документів	+	+	+	+	+
Електронний підпис	+	+	-	+	+
Завантаження файлів	+	+	+	+	+
Контроль доступу	+	+	+	-	+
Простий UI/UX	-	+	+	+	+

ВИМОГИ ДО ПРОРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

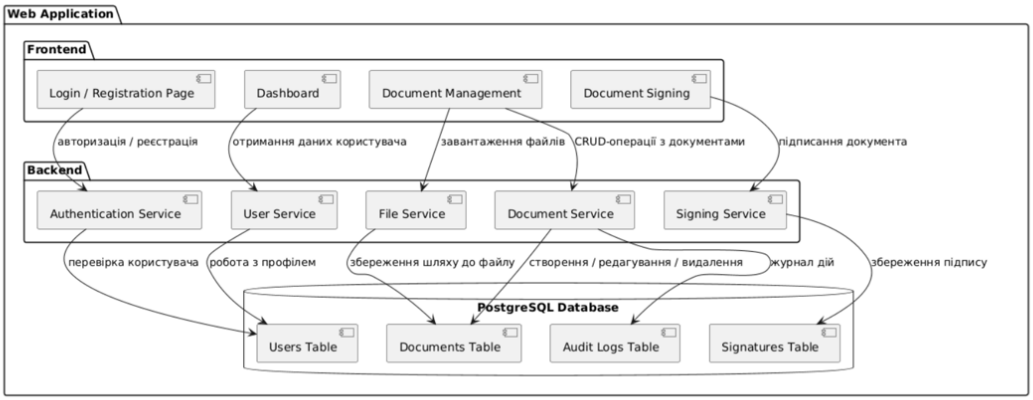
1. Забезпечення можливості реєстрації та авторизації користувачів у системі.
2. Створення, редагування, перегляд та видалення електронних документів.
3. Завантаження та збереження файлів документів на сервері.
4. Відображення списку документів користувача з можливістю пошуку.
5. Збереження інформації про документи та користувачів у базі даних.
6. Взаємодія клієнтської та серверної частини через REST API.

Нефункціональні вимоги:

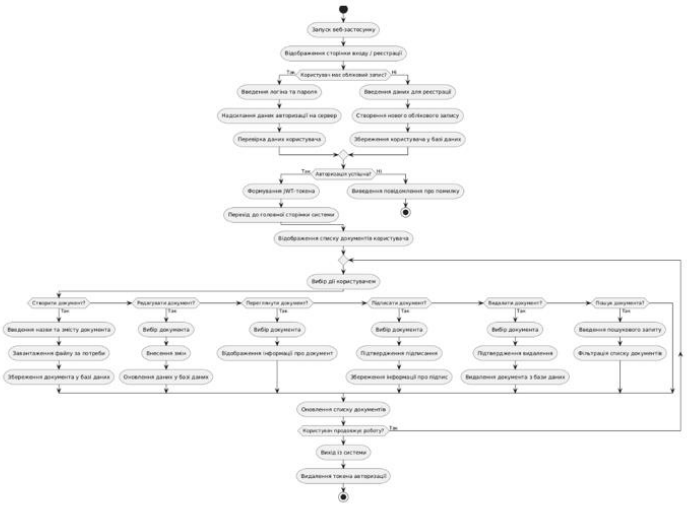
1. Час обробки запитів не повинен перевищувати допустимі значення та має забезпечувати швидкий відгук системи.
2. Забезпечення безпеки даних користувачів за рахунок використання хешування паролів та JWT-авторизації.
3. Система повинна бути надійною та коректно обробляти помилки без втрати даних.
4. Забезпечення можливості масштабування системи при збільшенні кількості користувачів і даних.
5. Забезпечення цілісності та збереження даних у базі даних PostgreSQL.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

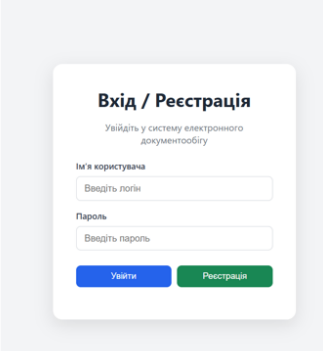
АРХІТЕКТУРНА СИСТЕМА КОМПОНЕНТІВ

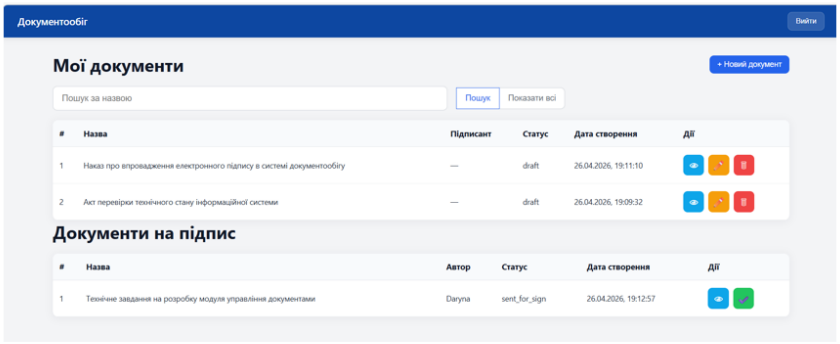


АЛГОРИТМ РОБОТИ

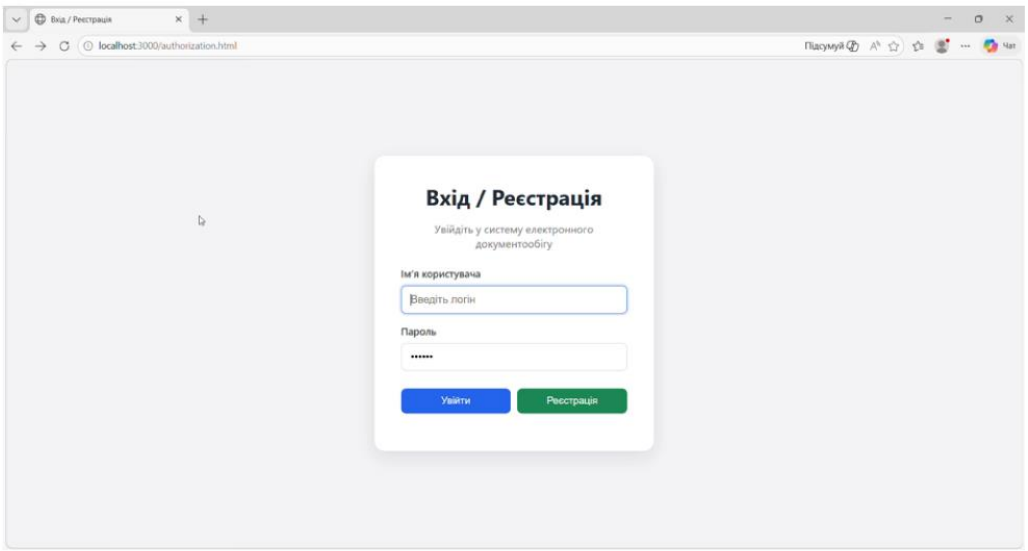


ЕКРАННІ ФОРМИ





ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ



Апробація

1. Бондаренко Д.В. Впровадження інтелектуальних технологій у системах електронного документообігу. // Матеріали VII Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. К.: ДУІКТ, 2026
2. Бондаренко Д.В. Впровадження системи електронного документообігу в сучасних організацій. // Матеріали VII Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026 (подано на друк)
3. Бондаренко Д.В. Розробка веб-застосунку електронного документообігу на основі Node.js // Матеріали VII Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026 (подано на друк)

Висновки

1. Проаналізовано сучасні підходи до організації електронного документообігу та визначено основні вимоги до системи.
2. Розроблено клієнт-серверну архітектуру веб-застосунку з використанням сучасних технологій.
3. Реалізовано механізм авторизації користувачів із застосуванням JWT для забезпечення безпеки доступу.
4. Створено функціональні модулі для роботи з документами, включаючи створення, редагування, перегляд та зберігання.
5. Реалізовано можливість завантаження та обробки файлів у системі.
6. Забезпечено збереження та цілісність даних за допомогою реляційної бази даних PostgreSQL.
7. Досягнуто підвищення ефективності роботи з документами за рахунок автоматизації процесів документообігу.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Файл auth.js

```
async function register() {
  const res = await fetch('/register', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      username: username.value,
      password: password.value
    })
  });
  const data = await res.json();
  if (data.token) {
    localStorage.setItem('token', data.token);
    location.href = "index.html";
  } else {
    msg.innerText = data.msg;
  }
}
async function login() {
  const res = await fetch('/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },


    body: JSON.stringify({
      username: username.value,
      password: password.value
    })
  });
  const data = await res.json();
  if (data.token) {
    localStorage.setItem('token', data.token);
    location.href = "index.html";
  } else {
    msg.innerText = data.msg;
  }
}
```

Файл authorization.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Вхід / Реєстрація</title>
  <link rel="stylesheet" href="style.css">
</head>
<body class="auth-page">

<div class="auth-wrapper">
  <div class="auth-card">
    <div class="auth-card__header">
      <h1>Вхід / Реєстрація</h1>
      <p>Увійдіть у систему електронного
документообігу</p>
    </div>
    <div class="auth-form">
      <label class="field-label" for="username">Ім’я
користувача</label>
      <input id="username" class="field-input"
type="text" placeholder="Введіть логін">
      <label class="field-label"
for="password">Пароль</label>
      <input id="password" class="field-input"
type="password" placeholder="Введіть пароль">
      <div class="auth-actions">
        <button class="btn btn-primary auth-btn"
onclick="login()">Увійти</button>
        <button class="btn btn-success auth-btn"
onclick="register()">Реєстрація</button>
      </div>
      <p id="msg" class="auth-message"></p>
    </div>
  </div>
</div>
<script src="auth.js"></script>
</body>
</html>
```

Файл index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Електронний документообіг</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
<script>
  if (!localStorage.getItem('token')) {
    location.href = "authorization.html";
  }
</script>
<header class="topbar">
  <div class="topbar__brand">Документообіг</div>
```

```
            <button class="topbar__logout"
onclick="logout()">Вийти</button>
          </header>
          <main class="page">
            <section class="page-header">
              <div class="header-left">
                <h1>Мої документи</h1>
              </div>
              <div class="header-right">
                <button class="btn btn-primary btn-create"
onclick="openCreateModal()">+ Новий документ</button>
              </div>
            </section>
            <section class="toolbar">
              <input
                type="text"
                id="searchInput"
                class="toolbar__search"
                placeholder="Пошук за назвою"
                oninput="filterDocs()"
              >
              <div class="toolbar__actions">
                <button class="btn btn-outline-primary"
onclick="filterDocs()">Пошук</button>
                <button class="btn btn-outline-secondary"
onclick="resetSearch()">Показати всі</button>
              </div>
            </section>
            <section class="table-card">
              <table class="docs-table">
                <thead>
                  <tr>
                    <th>#</th>
                    <th>Назва</th>
                    <th>Підписант</th>
                    <th>Статус</th>
                    <th>Дата створення</th>
                    <th>Дії</th>
                  </tr>
                </thead>
                <tbody id="docsTableBody">
                  <tr>
                    <td colspan="6" class="empty-
row">Завантаження...</td>
                  </tr>
                </tbody>
              </table>
            </section>
            <section class="page-header second-section">
              <div class="header-left">
                <h1>Документи на підпис</h1>
              </div>
            </section>
            <section class="table-card">
              <table class="docs-table">
                <thead>
                  <tr>
                    <th>#</th>
                    <th>Назва</th>
                    <th>Автор</th>
                    <th>Статус</th>
                    <th>Дата створення</th>
                    <th>Дії</th>
                  </tr>
                </thead>
                <tbody id="docsToSignBody">
                  <tr>
                    <td colspan="6" class="empty-
row">Завантаження...</td>
                  </tr>
                </tbody>
              </table>
            </section>
          </main>

          <!-- CREATE MODAL -->
          <div class="modal-backdrop" id="createModal">
            <div class="modal-window">
              <div class="modal-header">
                <h2>Новий документ</h2>
                <button class="modal-close"
onclick="closeCreateModal()">×</button>
              </div>

              <div class="modal-body">
                <label class="field-label"
for="createTitle">Назва</label>
                <input id="createTitle" type="text" class="field-
input" placeholder="Введіть назву">

                <label class="field-label"
for="createContent">Зміст</label>
                <textarea id="createContent" class="field-textarea"
placeholder="Введіть текст документу"></textarea>
                <label class="field-label" for="createSigner">Кому
на підпис</label>
                <select id="createSigner" class="field-input">
                  <option value="">Не вибрано</option>
                </select>
                <label class="field-label" for="createFile">Файл
(pdf / docx)</label>
                <input id="createFile" type="file" class="field-
input" accept=".pdf,.doc,.docx">
              </div>
              <div class="modal-footer">
```

```
                <button class="btn btn-secondary"
onclick="closeCreateModal()">Відмінити</button>
                <button class="btn btn-primary"
onclick="createDoc()">Зберегти</button>
              </div>
            </div>
          </div>
          <!-- VIEW MODAL -->
          <div class="modal-backdrop" id="viewModal">
            <div class="modal-window modal-window--wide">
              <div class="modal-header">
                <h2>Деталі документа</h2>
                <button class="modal-close"
onclick="closeViewModal()">×</button>
              </div>

              <div class="modal-body">
                <div class="details-grid">
                  <div class="details-row">
                    <span class="details-label">Назва:</span>
                    <span id="viewTitle"></span>
                  </div>
                  <div class="details-row">
                    <span class="details-label">Автор:</span>
                    <span id="viewOwner"></span>
                  </div>
                  <div class="details-row">
                    <span class="details-
label">Підписант:</span>
                    <span id="viewSigner"></span>
                  </div>
                  <div class="details-row">
                    <span class="details-label">Статус:</span>
                    <span id="viewStatus"></span>
                  </div>
                  <div class="details-row">
                    <span class="details-label">Дата
створення:</span>
                    <span id="viewDate"></span>
                  </div>
                  <div class="details-row">
                    <span class="details-label">Хто
підписав:</span>
                    <span id="viewSignature"></span>
                  </div>
                  <div class="details-row">
                    <span class="details-label">Дата
підпису:</span>
                    <span id="viewSignedAt"></span>
                  </div>
                  <div class="details-row details-row--column">
                    <span class="details-label">Зміст:</span>
                    <div class="details-content"
id="viewContent"></div>
                  </div>
                  <div class="details-row">
                    <span class="details-label">Хеш
документа:</span>
                    <span id="viewHash"></span>
                  </div>
                  <div class="details-row">
                    <span class="details-label">Файл:</span>
                    <span id="viewFile"></span>
                  </div>
                </div>
              </div>

              <div class="modal-footer">
                <button class="btn btn-secondary"
onclick="closeViewModal()">Закрити</button>
              </div>
            </div>
          </div>

          <!-- EDIT MODAL -->
          <div class="modal-backdrop" id="editModal">
            <div class="modal-window">
              <div class="modal-header">
                <h2>Редагування документа</h2>
                <button class="modal-close"
onclick="closeEditModal()">×</button>
              </div>
              <div class="modal-body">
                <label class="field-label"
for="editTitle">Назва</label>
                <input id="editTitle" type="text" class="field-
input">

                <label class="field-label"
for="editContent">Зміст</label>
                <textarea id="editContent" class="field-
textarea"></textarea>

                <label class="field-label" for="editSigner">Кому на
підпис</label>
                <select id="editSigner" class="field-input">
                  <option value="">Не вибрано</option>
                </select>
              </div>

              <div class="modal-footer">
                <button class="btn btn-secondary"
onclick="closeEditModal()">Відмінити</button>
                <button class="btn btn-primary"
onclick="saveEdit()">Зберегти</button>
```

```
            </div>
          </div>
        </div>
        <!-- SIGN MODAL -->
        <div class="modal-backdrop" id="signModal">
          <div class="modal-window">
            <div class="modal-header">
              <h2>Підписання документа</h2>
              <button class="modal-close"
onclick="closeSignModal()">×</button>
            </div>

            <div class="modal-body">
              <p class="sign-text">
                Ви дійсно хочете підписати документ
                <strong id="signDocTitle"></strong>?
              </p>
              <p class="sign-warning">
                Після підписання редагування документа буде
заборонене.
              </p>
            </div>
            <div class="modal-footer">
              <button class="btn btn-secondary"
onclick="closeSignModal()">Скасувати</button>
              <button class="btn btn-success"
onclick="confirmSign()">Підписати</button>
            </div>
          </div>
        </div>
        <script src="script.js"></script>
        </body>
        </html>

        Файл script.js
        let token = localStorage.getItem('token');
        let allDocs = [];
        let docsToSign = [];
        let allUsers = [];
        let currentEditId = null;
        let currentSignId = null;

        window.onload = async () => {
          if (!token) {
            location.href = "authorization.html";
            return;
          }

          await loadUsers();
          await loadDocs();
          await loadDocsToSign();
        };
```

```
// ---------- HELPERS ----------
function escapeHtml(text) {
  if (!text) return "";
  return text
    .replaceAll("&", "&amp;")
    .replaceAll("<", "&lt;")
    .replaceAll(">", "&gt;");
}

function formatDate(value) {
  if (!value) return "—";
  return new Date(value).toLocaleString('uk-UA');
}

async function parseResponse(res) {
  const contentType = res.headers.get('content-type') || '';
  if (contentType.includes('application/json')) {
    return await res.json();
  }
  return { msg: await res.text() };
}

function getMyDocById(id) {
  return allDocs.find(d => String(d.id) === String(id));
}

function getSignDocById(id) {
  return docsToSign.find(d => String(d.id) === String(id));
}

// ---------- AUTH ----------
function logout() {
  localStorage.removeItem('token');
  location.href = "authorization.html";
}

// ---------- USERS ----------
async function loadUsers() {
  try {
    const res = await fetch('/users', {
      headers: { Authorization: 'Bearer ' + token }
    });

    const data = await parseResponse(res);
    if (!res.ok) return;

    allUsers = Array.isArray(data) ? data : [];
    fillSignerSelects();
  } catch (err) {
    console.error(err);
  }
}
```

```
        function fillSignerSelects() {
            const createSelect =
document.getElementById('createSigner');
            const editSelect =
document.getElementById('editSigner');

            if (createSelect) {
                createSelect.innerHTML = `<option value="">Не
вибрано</option>`;
                allUsers.forEach(user => {
                    createSelect.innerHTML += `<option
value="${user.id}">${escapeHtml(user.username)}</option>`;
                });
            }

            if (editSelect) {
                editSelect.innerHTML = `<option value="">Не
вибрано</option>`;
                allUsers.forEach(user => {
                    editSelect.innerHTML += `<option
value="${user.id}">${escapeHtml(user.username)}</option>`;
                });
            }
        }

        // ---------- MODALS ----------
        function openCreateModal() {
            document.getElementById('createModal').style.display =
'flex';
        }

        function closeCreateModal() {
            document.getElementById('createModal').style.display =
'none';
            document.getElementById('createTitle').value = '';
            document.getElementById('createContent').value = '';
            document.getElementById('createFile').value = '';
            document.getElementById('createSigner').value = '';
        }

        function openEditModal() {
            document.getElementById('editModal').style.display =
'flex';
        }

        function closeEditModal() {
            document.getElementById('editModal').style.display =
'none';
            currentEditId = null;
        }

        function openViewModal() {
            document.getElementById('viewModal').style.display =
'flex';
        }

        function closeViewModal() {
            document.getElementById('viewModal').style.display =
'none';
        }

        function openSignModal() {
            document.getElementById('signModal').style.display =
'flex';
        }

        function closeSignModal() {
            document.getElementById('signModal').style.display =
'none';
            currentSignId = null;
            document.getElementById('signDocTitle').innerText = '';
        }

        window.onclick = function (event) {
            const ids = ['createModal', 'editModal', 'viewModal',
'signModal'];
            ids.forEach(id => {
                const modal = document.getElementById(id);
                if (event.target === modal) {
                    if (id === 'createModal') closeCreateModal();
                    if (id === 'editModal') closeEditModal();
                    if (id === 'viewModal') closeViewModal();
                    if (id === 'signModal') closeSignModal();
                }
            });
        };

        // ---------- LOAD DATA ----------
        async function loadDocs() {
            const res = await fetch('/documents', {
                headers: { Authorization: 'Bearer ' + token }
            });

            const data = await parseResponse(res);
            if (!res.ok) return;

            allDocs = Array.isArray(data) ? data : [];
            renderMyDocs(allDocs);
        }

        async function loadDocsToSign() {
            const res = await fetch('/documents/to-sign', {
                headers: { Authorization: 'Bearer ' + token }
            });
```

```
        const data = await parseResponse(res);
        if (!res.ok) return;

        docsToSign = Array.isArray(data) ? data : [];
        renderDocsToSign(docsToSign);
      }

      // ---------- RENDER ----------
      function renderMyDocs(docs) {
        const tbody =
document.getElementById('docsTableBody');
        tbody.innerHTML = '';

        if (!docs.length) {
          tbody.innerHTML = `<tr><td colspan="6"
class="empty-row">Документів поки немає</td></tr>`;
          return;
        }

        docs.forEach((doc, index) => {
          const isSigned = !!doc.signed;

          tbody.innerHTML += `
            <tr>
              <td>${index + 1}</td>
              <td>${escapeHtml(doc.title || "Без назви")}</td>
              <td>${escapeHtml(doc.signer_username || "—
")}</td>
              <td>${escapeHtml(doc.status || "draft")}</td>
              <td>${formatDate(doc.created_at)}</td>
              <td>
                <div class="actions">
                  <button class="btn-icon btn-view"
onclick="viewDoc('${doc.id}', 'owner')" title="Деталі">·</button>
                  <button class="btn-icon btn-edit"
onclick="editDoc('${doc.id}')" title="Редагувати" ${isSigned ?
'disabled' : ''}></button>
                  <button class="btn-icon btn-delete"
onclick="deleteDoc('${doc.id}')" title="Видалити">·</button>
                </div>
              </td>
            </tr>
          `;
        });
      }

      function renderDocsToSign(docs) {
        const tbody =
document.getElementById('docsToSignBody');
        tbody.innerHTML = '';

        if (!docs.length) {
          tbody.innerHTML = `<tr><td colspan="6"
class="empty-row">Немає документів на підпис</td></tr>`;
          return;
        }

        docs.forEach((doc, index) => {
          tbody.innerHTML += `
            <tr>
              <td>${index + 1}</td>
              <td>${escapeHtml(doc.title || "Без назви")}</td>
              <td>${escapeHtml(doc.owner_username || "—
")}</td>
              <td>${escapeHtml(doc.status ||
"sent_for_sign")}</td>
              <td>${formatDate(doc.created_at)}</td>
              <td>
                <div class="actions">
                  <button class="btn-icon btn-view"
onclick="viewDoc('${doc.id}', 'signer')" title="Деталі">·</button>
                  <button class="btn-icon btn-sign"
onclick="signDoc('${doc.id}')" title="Підписати">
                </div>
              </td>
            </tr>
          `;
        });
      }

      // ---------- SEARCH ----------
      function filterDocs() {
        const value =
document.getElementById('searchInput').value.trim().toLowerCase();

        if (!value) {
          renderMyDocs(allDocs);
          return;
        }
        const filtered = allDocs.filter(doc =>
          (doc.title || "").toLowerCase().includes(value) ||
          (doc.content || "").toLowerCase().includes(value) ||
          (doc.signer_username ||
"").toLowerCase().includes(value)
        );
        renderMyDocs(filtered);
      }
      function resetSearch() {
      document.getElementById('searchInput').value = '';
        renderMyDocs(allDocs);
      }

      // ---------- CREATE ----------
      async function createDoc() {
```

```
            const title =
document.getElementById('createTitle').value.trim();
            const content =
document.getElementById('createContent').value.trim();
            const file =
document.getElementById('createFile').files[0];
            const signerId =
document.getElementById('createSigner').value;
            if (!title && !content && !file) {
              alert("Заповніть хоча б одне поле або додайте
файл");
              return;
            }
            const formData = new FormData();
            formData.append('title', title || (file ? file.name : 'Без
назви'));
            formData.append('content', content || '');
            if (signerId) formData.append('signerId', signerId);
            if (file) formData.append('file', file);

            const res = await fetch('/documents', {
              method: 'POST',
              headers: { Authorization: 'Bearer ' + token },
              body: formData
            });
            const data = await parseResponse(res);
            if (!res.ok) {
              alert(data.msg || "Не вдалося створити документ");
              return;
            }
            closeCreateModal();
            await loadDocs();
            await loadDocsToSign();
          }
          // ---------- VIEW ----------
          function viewDoc(id, type) {
            const doc = type === 'signer' ? getSignDocById(id) :
getMyDocById(id);
            if (!doc) return;
            document.getElementById('viewTitle').innerText =
doc.title || "Без назви";
            document.getElementById('viewOwner').innerText =
doc.owner_username || "—";
            document.getElementById('viewSigner').innerText =
doc.signer_username || "—";
document.getElementById('viewStatus').innerText = doc.status || "—";
          document.getElementById('viewDate').innerText =
formatDate(doc.created_at);
document.getElementById('viewSignature').innerText = doc.signature ||
"—";  document.getElementById('viewSignedAt').innerText =
formatDate(doc.signed_at);
document.getElementById('viewContent').innerText = doc.content ||
"—";
            document.getElementById('viewHash').innerText =
doc.document_hash || "—";
            const fileContainer =
document.getElementById('viewFile');
            if (doc.file) {
              fileContainer.innerHTML = `<a class="file-link"
href="/uploads/${encodeURIComponent(doc.file)}"
target="_blank">Відкрити файл</a>`;
            } else {
              fileContainer.innerText = "—";
            }
            openViewModal();
          }
          // ---------- EDIT ----------
          function editDoc(id) {
            const doc = getMyDocById(id);
            if (!doc) return;

            if (doc.signed) {
              alert("Підписаний документ не можна редагувати");
              return;
            }

            currentEditId = String(id);
            document.getElementById('editTitle').value = doc.title ||
'';
            document.getElementById('editContent').value =
doc.content || '';
            document.getElementById('editSigner').value =
doc.signer_id || '';

            openEditModal();
          }

          async function saveEdit() {
            if (!currentEditId) return;

            const title =
document.getElementById('editTitle').value.trim();
            const content =
document.getElementById('editContent').value.trim();
            const signerId =
document.getElementById('editSigner').value;

            const res = await fetch(`/documents/${currentEditId}`, {
              method: 'PUT',
              headers: {
                'Content-Type': 'application/json',
                Authorization: 'Bearer ' + token
              },
              body: JSON.stringify({
                title,
                content,
```

```
                signerId
            })
        });

        const data = await parseResponse(res);

        if (!res.ok) {
            alert(data.msg || "Не вдалося зберегти зміни");
            return;
        }

        closeEditModal();
        await loadDocs();
        await loadDocsToSign();
    }

    // ---------- SIGN ----------
    function signDoc(id) {
        const doc = getSignDocById(id);
        if (!doc) return;

        currentSignId = String(id);
        document.getElementById('signDocTitle').innerText =
doc.title || "Без назви";
        openSignModal();
    }

    async function confirmSign() {
        if (!currentSignId) return;

        const res = await
fetch(`/documents/${currentSignId}/sign`, {
            method: 'POST',
            headers: {
                Authorization: 'Bearer ' + token
            }
        });

        const data = await parseResponse(res);

        if (!res.ok) {
            alert(data.msg || "Не вдалося підписати документ");
            return;
        }

        closeSignModal();
        await loadDocs();
        await loadDocsToSign();
    }

    // ---------- DELETE ----------
    async function deleteDoc(id) {
        const confirmed = confirm("Видалити документ?");
        if (!confirmed) return;

        const res = await fetch(`/documents/${id}`, {
            method: 'DELETE',
            headers: {
                Authorization: 'Bearer ' + token
            }
        });
        const data = await parseResponse(res);
        if (!res.ok) {
            alert(data.msg || "Не вдалося видалити документ");
            return;
        }
        await loadDocs();
        await loadDocsToSign();
    }
```

Файл style.css

```
* {
    box-sizing: border-box;
}

body {
    margin: 0;
    font-family: "Segoe UI", Arial, sans-serif;
    background: #f3f4f6;
    color: #1f2937;
}

.topbar {
    height: 60px;
    background: #0d47a1;
    color: #fff;
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 0 28px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.08);
}

.topbar__brand {
    font-size: 18px;
    font-weight: 500;
}

.topbar__logout {
    background: rgba(255, 255, 255, 0.15);
    color: #fff;
    border: 1px solid rgba(255, 255, 255, 0.18);
    border-radius: 8px;
    padding: 8px 14px;
```

```
    cursor: pointer;
    transition: 0.2s;
}

.topbar__logout:hover {
    background: rgba(255, 255, 255, 0.25);
}

.page {
    max-width: 1400px;
    margin: 28px auto;
    padding: 0 28px;
}

.page-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 18px;
}

.header-left h1 {
    margin: 0;
    font-size: 32px;
    font-weight: 700;
    color: #111827;
}

.header-right {
    display: flex;
    align-items: center;
}

.toolbar {
    display: flex;
    gap: 16px;
    align-items: center;
    margin-bottom: 18px;
}

.toolbar__search {
    width: 720px;
    max-width: 100%;
    height: 44px;
    border: 1px solid #d1d5db;
    border-radius: 8px;
    padding: 0 16px;
    font-size: 16px;
    background: #fff;
}

.toolbar__search:focus,
.field-input:focus,
.field-textarea:focus {
    outline: none;
    border-color: #2563eb;
    box-shadow: 0 0 0 3px rgba(37, 99, 235, 0.12);
}

.toolbar__actions {
    display: flex;
    gap: 0;
}

.table-card {
    background: #fff;
    border-radius: 10px;
    overflow: hidden;
    box-shadow: 0 2px 14px rgba(15, 23, 42, 0.06);
}

.docs-table {
    width: 100%;
    border-collapse: collapse;
}

.docs-table thead th {
    background: #f8fafc;
    color: #111827;
    font-weight: 700;
    font-size: 15px;
    text-align: left;
    padding: 14px 12px;
    border-bottom: 1px solid #e5e7eb;
}

.docs-table tbody td {
    padding: 14px 12px;
    border-bottom: 1px solid #edf0f3;
    vertical-align: middle;
    font-size: 14px;
}

.docs-table tbody tr:hover {
    background: #f9fbff;
}

.empty-row {
    text-align: center;
    color: #6b7280;
    padding: 28px 12px !important;
}

.col-number {
    width: 60px;
}
```

```
.col-name {
  width: 260px;
}

.col-content {
  width: 420px;
}

.col-date {
  width: 220px;
}

.col-actions {
  width: 320px;
}

.doc-title-cell {
  font-weight: 600;
  color: #111827;
}

.doc-content-preview {
  color: #4b5563;
  max-width: 420px;
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}

.actions {
  display: flex;
  gap: 8px;
  flex-wrap: wrap;
}

.btn {
  border: none;
  border-radius: 8px;
  padding: 10px 16px;
  font-size: 15px;
  cursor: pointer;
  transition: 0.2s ease;
}

.btn:hover {
  transform: translateY(-1px);
}

.btn-primary {
  background: #2563eb;
  color: #fff;
}

.btn-primary:hover {
  background: #1d4ed8;
}

.btn-create {
  padding: 8px 14px;
  font-size: 14px;
  border-radius: 8px;
  background: #2563eb;
  color: white;
}

.btn-create:hover {
  background: #1d4ed8;
}

.btn-success {
  background: #198754;
  color: #fff;
}

.btn-success:hover {
  background: #157347;
}

.btn-secondary {
  background: #6c757d;
  color: #fff;
}

.btn-secondary:hover {
  background: #5c636a;
}

.btn-outline-primary {
  background: #fff;
  color: #2563eb;
  border: 1px solid #2563eb;
  border-radius: 0;
}

.btn-outline-primary:hover {
  background: #eff6ff;
}

.btn-outline-secondary {
  background: #fff;
  color: #6b7280;
  border: 1px solid #cfd4da;
  border-left: none;
  border-radius: 0 8px 8px 0;
}
```

```
.btn-outline-secondary:hover {
  background: #f8fafc;
}

.btn-icon {
  width: 38px;
  height: 38px;
  border-radius: 8px;
  border: none;
  cursor: pointer;
  font-size: 18px;
  display: flex;
  align-items: center;
  justify-content: center;
  transition: 0.2s;
}

.btn-icon:hover:not(:disabled) {
  transform: scale(1.08);
}

.btn-icon:disabled {
  opacity: 0.45;
  cursor: not-allowed;
}

.btn-view {
  background: #0ea5e9;
  color: white;
}

.btn-view:hover:not(:disabled) {
  background: #0284c7;
}

.btn-edit {
  background: #f59e0b;
  color: white;
}

.btn-edit:hover:not(:disabled) {
  background: #d97706;
}

.btn-sign {
  background: #22c55e;
  color: white;
}

.btn-sign:hover:not(:disabled) {
  background: #16a34a;
}

.btn-delete {
  background: #ef4444;
  color: white;
}

.btn-delete:hover:not(:disabled) {
  background: #dc2626;
}

.modal-backdrop {
  display: none;
  position: fixed;
  inset: 0;
  background: rgba(17, 24, 39, 0.45);
  z-index: 1000;
  padding: 20px;
  overflow-y: auto;
  justify-content: center;
  align-items: center;
}

.modal-window {
  width: 100%;
  max-width: 860px;
  max-height: 90vh;

  background: #fff;
  border-radius: 12px;
  overflow: hidden;

  display: flex;
  flex-direction: column;

  box-shadow: 0 18px 40px rgba(0, 0, 0, 0.18);
  animation: modalShow 0.2s ease;
}
.modal-window--wide {
  max-width: 900px;
}

@keyframes modalShow {
  from {
    opacity: 0;
    transform: translateY(-12px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

.modal-header {
```

```
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 18px 18px 16px;
  border-bottom: 1px solid #e5e7eb;
}

.modal-header h2 {
  margin: 0;
  font-size: 18px;
  font-weight: 700;
}

.modal-close {
  border: none;
  background: transparent;
  font-size: 36px;
  line-height: 1;
  color: #6b7280;
  cursor: pointer;
  padding: 0;
}

.modal-close:hover {
  color: #111827;
}

.modal-body {
  padding: 18px;
  overflow-y: auto;
  flex: 1;
}

.modal-footer {
  padding: 16px 18px;
  border-top: 1px solid #e5e7eb;
  display: flex;
  justify-content: flex-end;
  gap: 10px;
}

.field-label {
  display: block;
  margin-bottom: 8px;
  font-size: 15px;
  font-weight: 500;
  color: #374151;
}

.field-input {
  width: 100%;
  height: 42px;
  border: 1px solid #d1d5db;
  border-radius: 8px;
  padding: 0 14px;
  font-size: 16px;
  margin-bottom: 16px;
  background: #fff;
}

.field-textarea {
  width: 100%;
  min-height: 170px;
  border: 1px solid #d1d5db;
  border-radius: 8px;
  padding: 14px;
  font-size: 16px;
  resize: vertical;
  margin-bottom: 16px;
  background: #fff;
}

.details-grid {
  display: flex;
  flex-direction: column;
  gap: 14px;
}

.details-row {
  display: flex;
  gap: 10px;
  align-items: flex-start;
  flex-wrap: wrap;
}

.details-row--column {
  flex-direction: column;
  gap: 8px;
}

.details-label {
  min-width: 130px;
  font-weight: 700;
  color: #111827;
}

.details-content {
  width: 100%;
  min-height: 150px;
  max-height: 400px;
  overflow-y: auto;

  background: #f8fafc;
  border: 1px solid #e5e7eb;
  border-radius: 8px;
  padding: 12px;
```

```
    white-space: pre-wrap;
    word-break: break-word;
    line-height: 1.5;
}

.file-link {
    color: #2563eb;
    text-decoration: none;
    font-weight: 500;
    word-break: break-all;
}

.file-link:hover {
    text-decoration: underline;
}

.sign-text {
    font-size: 16px;
    margin: 0 0 12px;
    color: #111827;
}

.sign-warning {
    margin: 0;
    color: #b45309;
    background: #fff7ed;
    border: 1px solid #fdba74;
    border-radius: 8px;
    padding: 12px;
}

@media (max-width: 900px) {
    .page-header,
    .toolbar {
        flex-direction: column;
        align-items: stretch;
    }

    .toolbar__search {
        width: 100%;
    }

    .toolbar__actions {
        width: 100%;
    }

    .toolbar__actions .btn {
        flex: 1;
    }

    .modal-window {
        margin: 20px auto;
    }

    .docs-table {
        display: block;
        overflow-x: auto;
    }

    .actions {
        flex-wrap: nowrap;
    }
}
.auth-page {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}
.auth-wrapper {
    width: 100%;
    display: flex;
    justify-content: center;
    align-items: center;
}
.auth-card {
    width: 100%;
    max-width: 420px;
    background: #ffffff;
    padding: 40px;
    border-radius: 16px;
    box-shadow: 0 10px 30px rgba(0,0,0,0.1);
}

/* Заголовок */
.auth-card__header {
    text-align: center;
    margin-bottom: 25px;
}

.auth-card__header h1 {
    margin: 0 0 10px;
}

.auth-card__header p {
    color: #6b7280;
}

.auth-actions {
    display: flex;
    gap: 10px;
    margin-top: 10px;
}

.auth-btn {
```

```
    flex: 1;
}
.auth-message {
    text-align: center;
    margin-top: 15px;
}
```