ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра комп'ютерної інженерії

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА WEB-ДОДАТКУ ДЛЯ ПОСТАНОВКИ ТА КЕРУВАННЯ ЗАДАЧ З РОЗРОБКИ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ ЗА МЕДОЛОГІЄЮ AGILE**»

Виконав: студент 5 курсу, групи ППЗ–52

спеціальності

121 Інженерія програмного забезпечення

Сичов В. М.

Керівник Коваленко Д. С.

Рецензент_____

Київ – 2021

ЗМІСТ

ВСТУП

Гнучким методологіям, також званим як agile-методології, зараз приділяється багато уваги в усьому світі. Впровадження гнучких методологій почалося відносно недавно, але вони стрімко набувають масовий характер.

Сучасний світ неможливо уявити без agile-медологій - команди, що використовують їх у своїй роботі, швидше досягають результатів, ніж ті, хто використовує класичні процеси. Клієнти більш задоволені результатами роботи гнучких команд. Члени цих команд отримують більше задоволення від своєї роботи.

На сьогодні гнучкі медології показали свою ефективність в розробці програмного забезпечення, тому все частіше зустрічаються у якості основних процесів компаній працюючих у сфері інформаційних технологій.

Процеси засновані на гнучких методологіях вирішують низку проблем, в першу чергу це - постійне навчання та адаптація до мінливих чинників.

В той же час команди, які спеціалізуються в першу чергу на розробці програмного забезпечення, та використовували agile-процеси у своїй роботі зіштовхнулися з проблемою - для ефективної роботи у команді за agile-процесами необхідно створити ефективний механізм для постановки та керування задачами.

В процесі дослідження вирішувалися наступні питання:

- Для чого використовуються гнучкі методології?
- Які процеси засновані на гнучких методологіях?
- Які є існуючі інструменти для постановки та керування задачами?

Метою дослідження є створення нового інструменту для постановки та керування задач з розробки програмного забезпечення, який має відповідати основним характеристикам її існуючих аналогів, при цьому бути зручним в інтеграції та мати зручний та інтуїтивно зрозумілий інтерфейс користувача.

1 ДОСЛІДЖЕННЯ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ ЗА МЕТОДОЛОГІЄЮ AGILE

1.1 Історія виникнення agile-методологій

Коріння гнучкого підходу тягнуться углиб століть, до сформульованого в 1620 році Френсісом Беконом (Francis Bacon) наукового методу. Однак логічніше вважати стартовою точкою розвитку Agile 30-ті роки XX століття, коли фізик і статистик Уолтер Шухарт (Walter Shewhart) з Лабораторії Белла (Bell Labs) почав застосовувати цикли "Плануй-Роби-Вивчай-Дій" (Plan-Do-Study-Act) для поліпшення продуктів і процесів. Шухарт передав основи такої ітеративно-інкрементальної розробки своєму учню, Вільяму Едвардсу Демінгу (W. Edwards Deming), який популяризував цей метод під час відновлення Японії після Другої світової війни.

У 1986, Хіротака Такеучі (Takeuchi Hirotaka), спільно зі своїм партнером Ікуджіро Нонака (Ikujiro Nonaka) опублікували в журналі «Harvard Business Review» статтю під назвою «The New New Product Development Game». Вивчаючи компанії, які були лідерами на ринку інновацій і випереджали своїх конкурентів, автори виявили командно-орієнтований підхід, який повністю змінював класичний процес розробки продукту. У цьому підході, замість класичної «естафети» про передачу продукту за етапами від одного функціонального фахівця до іншого, застосовувався підхід , схожий на гру в «регбі», коли команда просувається по дистанції як єдине ціле, постійно передаючи м'яч один одному.

У 1993 році Джефф Сазерленд зіткнувся з нездійсненним завданням: Компанія «Easel Corporation», яка займалася розробкою софту, повинні були менш ніж за шість місяців встигнути розробити заміну своєму найголовнішому продукту. У Джеффа вже був досвід роботи з нестандартними методологіями, такими як швидка розробка додатків, об'єктно-орієнтована розробка і Цикл Демінга (PDSA, Plan-Do-Study-Act), а також роботи з автономними креативними дослідницькими групами

("skunkworks"). Він прочитав сотні статей, поговорив з десятками провідних менеджерів продуктів. В ході цього дослідження його зацікавили кілька оригінальних ідей.

Одна з цих ідей прийшла з Лабораторії Белла, тієї самої, де народився Цикл Демінга (PDCA). У статті про роботу команди Borland Quattro Pro розповідалося про користь коротких щоденних зустрічей команди. Автори стверджували, що це покращує синхронізацію і підвищує продуктивність команди. Але ключова концепція Сазерлендом була взята зі статті Нонака і Такеучі про «регбійний» підхід, незважаючи на те, що він ставився швидше до виробництва, ніж до сфери IT. Взявши безліч ідей з різних джерел Сазерленд створив новий спосіб розробки софту, давши йому назву «Scrum», на честь аналогії з регбі. Метод Scrum дозволив йому успішно завершити практично неможливий проект в термін, в рамках бюджету і з небувало низькою кількістю багів.

Пізніше він об'єднався зі своїм колегою Кеном Швабером (Ken Schwaber) для формалізації підходу і в 1995 році Scrum був представлений всьому світу. [1]

1.2 Метологія «Scrum». Порівняння переваг та недоліків.

Методика Scrum за своєю суттю є евристична. В її основі лежить постійне навчання та адаптація до мінливих чинників. Scrum команда не знає всього на початку проекту, але розвиваючись буде отримувати уроки з досвіду. У структурі Scrum закладена та свобода, з якою команди пристосовуються до мінливих умов і вимог користувачів. Робочий процес передбачає зміну пріоритетів і короткі цикли релізу, що сприяє постійному навчанню та вдосконаленню команди.

Scrum - це високоефективна, прозора, мотивуюча методологія. В ній закладений підхід, від якого виграє і команда, і замовник.

Розглянемо його переваги:

1) Прозорість. У команді відкритий обмін інформацією, знаннями, проблемами, кожен відчуває себе причетним до спільної мети. Замовник завжди в курсі ходу робіт, вносить правки в процесі, отримує достовірну інформацію про терміни здачі.

2) Автономність команд. Члени команди самі вирішують, як працювати над проектом, свобода дій і відповідальність мотивують. Замовник передає вимоги команді безпосередньо, без зіпсованого телефону.

3) Мотивація результатом. Концепція Scrum дозволяє кожному члену групи бачити свої і загальні досягнення щодня. Замовник отримує приріст функціональності з кожної ітерації.

4) Мінімізація ринкових ризиків. Команда оперативно реагує на зміну вимог до проекту і не робить зайву роботу. Замовник отримує те, що хоче, і що затребуване на ринку.

5) Мінімізація фінансових ризиків. На усунення багів і додавання функціоналу витрачається мало часу і коштів, все вкладається в бюджет.

Явні недоліки Scrum:

1) Не підходить для проектів з туманними вимогами до кінцевого продукту, тому що замовник може нарощувати функціонал до нескінченності.

2) Складно навчитися правильно розставляти пріоритети і оцінювати завдання.

3) Успіх проекту занадто залежить від Scrum -майстра.

4) Scrum складно використовувати у великих проектах, доводиться масштабувати методологію і вводити збори scrum of scrums. У таких мітингах беруть участь представники кількох Scrum -команд, що працюють над пов'язаними продуктами.

Складні завдання можна впорядковувати в легко здійснимі, призначені для користувача історії, а значить, Scrum підійде для складних проектів. Завдяки тому, що ролі і планові заходи чітко розмежовані, протягом усього циклу розробки зберігається прозорість і колективна відповідальність. Частий випуск продуктів мотивує команду і гарантує задоволення користувачів, адже вони бачать, як продукт розвивається протягом короткого відрізка часу.

І все ж, щоб освоїти Scrum, може знадобитися якийсь час, особливо якщо команда розробників звикла до стандартної каскадної моделі. Новій команді належить вибрати Scrum-майстра, освоїтися в світі коротких ітерацій, щоденних Scrum-зборів і оглядів підсумків спринту. Це може стати справжнім струсом основ.

Але переваги в довгостроковій перспективі переважують всі складнощі, пов'язані з освоєнням нових принципів. Scrum успішно застосовується в розробці складного апаратного і програмного забезпечення в самих різних галузях. Це хороший аргумент на користь впровадження методики в рамках організації. [2]

1.2.1 Організація робочого процесу в Scrum командах

Спочатку визначимо три артефакти Scrum. **Артефакт** - це те, що ми створюємо, наприклад інструмент для вирішення проблеми. У Scrum існує три артефакти: беклог продукту, беклог спринту і інкремент з вашими критеріями готовності. Ці три постійні присутні в кожній команді Scrum, ми постійно до них звертаємося і приділяємо їм час.

Беклог продукту - це головний список завдань, які необхідно виконати. Його веде власник або менеджер продукту. Це постійно мінливий перелік функцій, вимог, поліпшень і виправлень, з якого складаються завдання для беклога спринту. У загальному і цілому це список завдань команди. Власник продукту постійно звертається до беклогу продукту, змінює в ньому пріоритети і підтримує його актуальність, бо може з'явитися нова інформація або можуть статися зміни на ринку, через що не буде сенсу виконувати задачі або з'являться нові способи вирішення проблем.

Спринт - це фактичний проміжок часу, протягом якого команда Scrum спільно працює над створенням готового інкремента. Як правило, спринт триває два тижні, хоча деяким командам простіше спланувати обсяг спринту на один тиждень або поставити інкремент, що володіє достатньою цінністю, за місяць Але останнє слово завжди за командою. Можна змінювати тривалість спринту, якщо здається, що він не підходить. Протягом цього періоду власник продукту і команда розробників можуть переглянути обсяг спринту, якщо це необхідно. Це і є ключ до розуміння емпіричної суті Scrum.

Беклог спринту - це список робочих завдань, призначених для користувача історій або виправлень багів, відібраних командою розробників для реалізації в поточному циклі спринту. Перед кожним спринтом проводяться збори з його планування, на якому команда вибирає, які завдання з беклога продукту потрібно виконати в рамках спринту. Беклог спринту може не бути фіксованим і може змінюватися по ходу обговорення. Однак ніщо не повинно заважати досягненню основної мети спринту - того, чого команда хоче добитися за поточний спринт.

Інкремент (або мета спринту) - це готовий до використання кінцевий продукт за підсумками спринту. Слово «інкремент» не так уже й широко зустрічається в повсякденному житті. Його часто визначають як прийняті в команді критерії готовності продукту, контрольну точку, мета спринту або навіть повну версію або поставлений епік. Все залежить від того, якими критеріями готовності керується ваша команда і як вибираються цілі спринту. Наприклад, деякі команди воліють випускати що-небудь для своїх клієнтів в кінці кожного спринту. Для них слово «готове» означає «поставлено». Однак для інших команд це може бути непрактично. Якщо команда працює над серверним продуктом, який можна постачати клієнтам лише раз в три місяці, то як і раніше можна розбивати роботу на двотижневі спринти, але для команди продукт буде «готовий», коли вона закінчить роботу над частиною більшої версії, яку планує поставити цілком. Однак, чим більше часу йде на випуск програмного забезпечення (далі ПЗ), тим менше шансів у цього ПЗ здобути успіх.

Допустимі різні варіанти, навіть коли мова йде про речі, яким команда може надавати ту чи іншу форму. Це показує, чому важливо залишатися відкритими до вдосконалення, зокрема до вдосконалення способу ведення артефактів. Можливо, через прийняті критерії готовності команда відчуває надмірний тиск і потрібно переглянути ці критерії.

Більш відомі такі складові методики Scrum, як послідовні заходи, церемонії чи збори, які регулярно проводять команди Scrum. Саме в підході до зборів найпомітніше проявляються відмінності між командами. Деяким командам в тягар проводити одноманітні збори; в інших робочі зустрічі обов'язкові. Якщо працівники тільки починають знайомство зі Scrum, рекомендується протягом перших двох спринтів провести всі збори, щоб зрозуміти своє ставлення до них. Після цього можна організувати коротку ретроспективу, щоб вирішити, що потрібно скорегувати. [3]

Основні збори, в яких може взяти участь команда Scrum:

1) **Організація беклога**. За цей захід, також відомий як ведення беклога, несе відповідальність власник продукту. У число його основних обов'язків входять приведення продукту у відповідність з його концепцією і постійне відстеження настроїв на ринку і потреб клієнта. Для цього власник продукту і веде список, змінюючи в ньому пріоритети і підтримуючи його в актуальному вигляді на підставі інформації від користувачів і команди розробників, щоб в будь-який час можна було приступити до роботи над внесенням в нього завдань.

2) **Планування спринту**. На цих зборах команда розробників під керівництвом Scrum-майстра планує роботу (обсяг спринту), яку необхідно виконати протягом поточного спринту. На ньому вибирається мета. Потім додаються конкретні призначені для користувача історії з беклога продукту. Ці історії завжди співвідносяться з метою. При цьому команда Scrum погоджує такі історії, які можна буде реалізувати на практиці в ході спринту.

В кінці зборів з планування кожен член команди Scrum повинен чітко уявляти, які завдання можна виконати за спринт і як поставити інкремент.

3) **Спринт**. Всі заходи, від планування до ретроспективи, проводяться протягом спринту. Після того як часовий проміжок визначено, він повинен залишатися незмінним, поки ведеться розробка. Так команда буде отримувати цінні уроки з минулого досвіду і застосовувати висновки до майбутніх спринтів.

4) **Щоденна Scrum-нарада, або стендап**. Це дуже коротке щоденне зібрання, яке для зручності проводиться в один і той же час (зазвичай вранці) і в одному і тому ж місці. Багато команд намагаються вкластися в 15 хвилин, однак це лише рекомендація. Такі збори також називається «щоденним стендапом», що підкреслює його стислість. Щоденна Scrum-нарада проводиться, щоб кожен учасник команди був в курсі того, що відбувається, не відхилявся від мети і отримував план роботи на найближчі 24 години.

Стендап - вдалий час повідомити про все, що заважає вам досягти мети спринту, в тому числі про Блокер.

Найчастіше в рамках стендапа кожному учаснику команди пропонується відповісти на наступні три питання, пов'язані з досягненням мети спринту:

- «Що мені вдалося зробити вчора?»
- «Що я планую зробити сьогодні?»
- «Чи може мені щось завадити?»

5) **Огляд підсумків спринту**. В кінці спринту команда збирається для перегляду демонстрації інкремента (або для його вивчення) в неформальній обстановці. Команда розробників подає заінтересованим сторонам і колегам завершені робочі завдання з беклога, щоб зібрати відгуки. Власник продукту вирішує, чи варто випускати інкремент, хоча в більшості випадків команда отримує зелене світло.

На зборах з огляду підсумків власник продукту також змінює беклог продукту на підставі результатів останнього завершеного спринту. Цей процес може перейти в планування наступного спринту. Якщо спринт триває один місяць, відводять під збори для огляду підсумків не більше чотирьох годин.

6) **Ретроспектива спринту**. Ретроспектива проводиться, щоб команда зафіксувала і обговорила всі успіхи і невдачі спринту, проекту, учасників та їх взаємовідносин, інструментів або навіть певних зборів. Мета ретроспективи - створити умови, щоб команда могла приділити увагу всьому, що вдалося і що потрібно поліпшити в наступний раз, і не зациклювалася на невдачах.

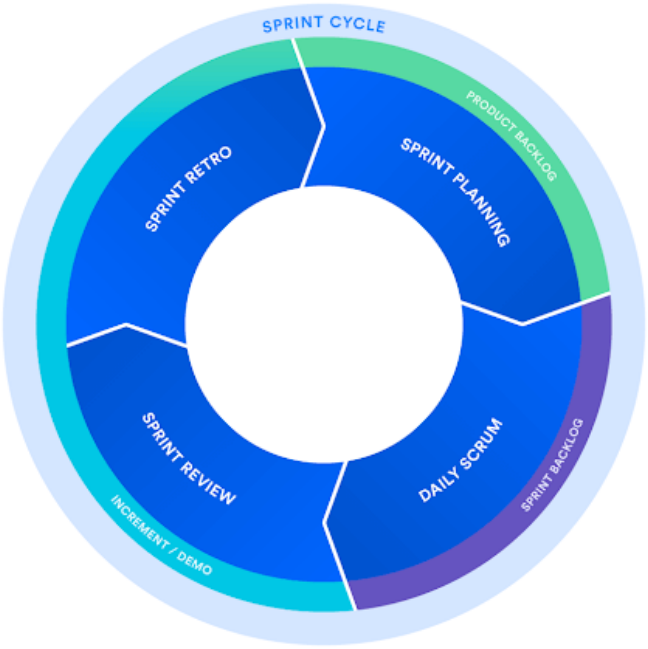

Рисунок 1 – Схема циклу спринту

Саме команда визначає успіх Scrum, він просто не буде працювати там, де люди не хочуть стати краще. Мотивація вже закладена всередину Scrum, а за підтримки керівництва команди збільшують продуктивність в кілька разів.

Scrum одночасно простий і складний, потрібно бути готовим, що вийде не відразу. Головне - не зупинятися, пробувати знову, вчитися по книгах або проходити тренінги, використовувати додатки, щоб стежити за ходом роботи і ефективністю команди.

1.3 Kanban, як сучасний підхід до розробки програмного забезпечення за методологією Agile

Kanban - це один із найпопулярніших підходів розробки ПЗ за методикою Agile і DevOps.

DevOps - низка практик, призначених для пожвавлення взаємодії розробників із фахівцями інформаційно-технологічного обслуговування та зближення їхніх робочих процесів одне з одним. Ґрунтується на думці про тісну взаємозалежність між розробкою та використанням програмного забезпечення і має на меті допомогти організаціям швидше створювати та оновлювати програмні продукти та послуги.[4]

Kanban передбачає обговорення продуктивності в режимі реального часу і повну прозорість робочих процесів. Етапи роботи візуально представлені на Kanban-дошці, що дозволяє членам команди бачити стан кожного завдання в будь-який момент часу.

Ключові принципи методології не втрачають актуальності, їх можна застосувати практично в будь-якій галузі, однак особливим успіхом agile користується серед команд розробників ПЗ. Частково це обумовлено тим, що від них не потрібно ніяких додаткових витрат - потрібно просто вивчити основні принципи методології. Якщо застосовувати Kanban в цехах, потрібно буде змінити фізичні процеси і придбати додаткові матеріали, а командам розробників ПЗ будуть потрібні тільки дошка та картки, та й ті можуть бути віртуальними.

Kanban має такі переваги:

1) **Гнучкість планування.** Kanban-команда концентрується тільки на поточну роботу. По завершенні робочого завдання команда забирає таку задачу з верху беклога. Власник програмного продукту може змінювати пріоритет завдань в беклозі, не заважаючи роботі команди, оскільки зміни відбуваються за межами поточних робочих завдань. Якщо власник стежить, щоб нагорі беклога були найважливіші робочі завдання, команда розробників

буде гарантовано поставляти максимально цінний продукт бізнесу. Таким чином, необхідність в спринтах фіксованої тривалості, використовуваних в методиці Scrum, просто немає.

2) **Скорочення часу циклу**. Тривалість циклу - ключовий показник для Kanban-команд. Під тривалістю циклу розуміється час проходження життєвого циклу робочого завдання, від початку роботи над завданням до його поставки. Оптимізувавши тривалість циклу, в майбутньому команда зможе з упевненістю прогнозувати термін поставки завдань.

Якщо тими чи іншими навичками володіє кілька людей, тривалість циклу скорочується, якщо ж тільки один - в процесі з'являється вузьке місце. Саме тому команди прагнуть ділитися знаннями і впроваджують такі практики, як перевірка коду та наставництво. Завдяки обміну знаннями члени команди можуть виконувати різноманітні завдання, що ще більше оптимізує тривалість циклу. Це також означає, що в разі скупчення роботи вся команда зможе взятися за неї і відновити нормальний перебіг процесу. Наприклад, тестування не завжди виконують тільки інженери з тестування. Розробники теж можуть взяти участь.

3) **Менше вузьких місць**. Багатозадачність вбиває ефективність. Чим більше незавершених завдань, тим частіше доводиться перемикатися між ними, а це позначається на термінах їх завершення. Тому ключовий принцип Kanban полягає в обмеженні обсягу незавершеної роботи (WIP). Ліміти незавершеної роботи дозволяють швидко знаходити в роботі команди вузькі і проблемні місця, викликані браком уваги, людей або навичок.

Наприклад, типова команда розробників ПЗ може використовувати чотири стани процесу розробки: «Заплановано», «В роботі», «Перевірка коду» і «Зроблено». Для стану перевірки коду можна встановити ліміт WIP, рівний 2. Число може здатися маленьким, але на все є свої причини: розробники вважають за краще писати власний код, а не перевіряти чужий. Низький ліміт стимулює команду приділяти особливу увагу завданням в стані перевірки, а також перевіряти чужу роботу, перш ніж створювати свої

завдання на перевірку коду. В кінцевому підсумку це скорочує загальний час циклу.

4) **Наочність**. Одна з основних цінностей - гранична увага до підвищення ефективності команди з кожної робочої ітерації. Графіки - це візуальний засіб, що дозволяє командам не зупинятися на досягнутому. Якщо у всієї команди є доступ до даних, простіше помітити (і усунути) вузькі місця в процесі. Kanban-команди часто використовують два загальних звіти: графіки управління і сукупного потоку.

5) **Безперервна поставка**. Безперервна поставка (CD) передбачає часту поставку релізів продукту клієнтам. Безперервна інтеграція (CI) - це практика інкрементної автоматизації складання і тестування коду протягом дня. Разом вони утворюють конвеєр CI / CD, без якого складно обійтися командам розробників (особливо командам DevOps), якщо вони хочуть швидше випускати якісне ПЗ. [5]

Kanban і CD ідеально доповнюють один одного, оскільки обидві методики засновані на своєчасному (і послідовному) постачанні цінностей. Чим швидше команда зможе випустити інноваційне рішення на ринок, тим більш конкурентоспроможним буде її продукт. Kanban-команди сконцентровані саме на оптимізації процесу поставки продуктів клієнтам.

Канбан має і недоліки:

1) система погано працює з командами чисельністю понад 5 осіб;

2) він не призначений для довгострокового планування;

На практиці система відмінно себе показує в сферах неосновного виробництва:

- Групи підтримки програмного забезпечення або служби підтримки.
- Канбан добре працює при управлінні стартапами без чіткого плану, але де активно просувається розробка.

1.3.1 Використання Kanban підходу у сучасному світі

Робота в Kanban йде за принципом «почніть з того, над чим працюєте прямо зараз». Це означає, що для початку роботи з Kanban не потрібно кидати поточну роботу. Для успішного застосування методики Kanban потрібно дотримати наступні три умови:

1) Ви розумієте поточні процеси в тому вигляді, в якому вони протікають в дійсності, і дотримуєтеся системи поточних ролей, обов'язків і посад.

2) Ви готові практикувати постійне вдосконалення і розвиток через еволюціонування.

3) Ви заохочуєте ініціативність на всіх рівнях, від рядових учасників до керівних осіб.

Робочий процес в Kanban - процес командний, тому в першу чергу команда повинна згуртуватися. Для зручності можна розділити роботу на окремі активності, з яких буде складатися робочий процес (стовпці). Після цього можна вирішити, як і коли додавати на дошку нові завдання (картки). Чи буде працювати служба техпідтримки, через яку клієнти будуть передавати ідеї, або команда буде проводити наради для складання та розміщення нових карток.

Крім того, варто визначити розмір картки і обсяг роботи, який вона покриває. Треба вибрати спосіб оцінки тривалості або складності роботи для всіх карток. Якщо якесь завдання занадто об'ємне або складне, можна розбити його на кілька карток.

Коли точка прийняття зобов'язань і точка поставки продукту визначені, команда починає роботу. Згодом процес буде вдосконалюватися на підставі зауважень команди. Kanban вимагає від учасників усіх рівнів постійно проявляти ініціативу. Ця філософія називається «кайдзен». Повага до людей і постійне вдосконалення в Kanban понад усе. Слідуючи цим цінностям, команда дуже швидко опанує цю методологію.

1.3.2 Різниця між Scrum і Kanban підходами, та їх зв’язок з agile-методологіями

Сенс Agile сформульований в Agile-маніфесті розробки ПЗ: «Люди і взаємодія важливіше процесів та інструментів. Працюючий продукт важливіше вичерпної документації. Співпраця з замовником важливіше узгодження умов контракту. Готовність до змін важливіше проходження попереднім планом ».

До окремих agile-підходів відносяться scrum і kanban.

Scrum - це «підхід структури». Над кожним проектом працює універсальна команда фахівців, до якої приєднується ще двоє людей: власник продукту і scrum-майстер. Перший з'єднує команду з замовником і стежить за розвитком проекту (це не формальний керівник команди, а скоріше куратор). Другий допомагає першому організувати бізнес-процес: проводить загальні збори, вирішує побутові проблеми, мотивує команду і стежить за дотриманням scrum-підходу.

Scrum-підхід поділяє робочий процес на рівні спринту - зазвичай це періоди від тижня до місяця, в залежності від проекту і команди. Перед спринтом формуються задачі на даний спринт, в кінці - обговорюються результати, а команда починає новий спринт. Спринти дуже зручно порівнювати між собою, що дозволяє управляти ефективністю роботи.

Kanban - це «підхід балансу». Його завдання - збалансувати різних фахівців всередині команди і уникнути ситуації, коли дизайнери працюють цілодобово, а розробники скаржаться на відсутність нових завдань.

Вся команда єдина - в kanban немає ролей власника продукту і scrum-майстра. Бізнес-процес поділяється не на універсальні спринти, а на стадії виконання конкретних завдань: «Планується», «Розробляється», «Тестується», «Завершено» і ін.

Головний показник ефективності в kanban - це середній час проходження завдання по дошці. Завдання пройшло швидко - команда

працювала продуктивно і злагоджено. Завдання затягнулися - треба думати, на якому етапі і чому виникли затримки та чию роботу треба оптимізувати.

Для візуалізації agile-підходів використовують дошки: фізичні та електронні. Вони дозволяють зробити робочий процес відкритим і зрозумілим для всіх фахівців, що важливо, коли у команди немає одного формального керівника. [6]

У чому різниця між Scrum і Kanban?

Основу Scrum складають короткі спринти, як правило, 2-3-х тижневі. Перед початком спринту команда сама формує список завдань на ітерацію, далі запускається спринт.

Після закінчення спринту виконані завдання заливаються на продакшн, а невиконані - переносяться в інший спринт. Як правило, завдання, які робляться під час спринту, не змінюються: що було на старті спринту - має бути зроблено до закінчення спринту.

Kanban дає більше гнучкості, якщо під гнучкістю розуміти частоту зміни пріоритетів. Якщо вчора розробник залив виконане завдання, а сьогодні отримали дані з "передової" і дізналися, що щось не працює так, як було задумано, то дають нові вимоги. Доповнені нові завдання піднімаються вгору і програміст бере цю задачу «зверху», виконує її і, до вечора все працює як треба.

У Scrum завдання прийнято оцінювати в Story points або в годинах. Без оцінки не вийде сформувати спринт: адже потрібно знати, чи встигне команда зробити завдання за 2 тижні. Через 2 тижні вона отримує цінну статистику - скільки годин або Story points команда змогла зробити за спринт. Velocity - це продуктивність команди за один спринт. Цей параметр дозволяє Scrum менеджеру передбачити, де команда буде через 2 тижні.

У Kanban не прийнято робити оцінку. Це опціонально, команда вирішує сама. Тут немає поняття «швидкість роботи команди», вважається тільки середній час на завдання. Час вираховується за допомогою спеціального звіту - Cycle Time.

Cycle Time для завдання дорівнює часу виконання завдання мінус час початку роботи над завданням. Наприклад, є колонки: to do, reopened, developing, testing, stage testing, deployed. Cycle time для завдання буде дорівнює deployed-developing, тобто скільки часу пройшло від моменту, коли завдання почали робити до моменту поки вона потрапила в deployed.

Отже, в Scrum наша мета - закінчити спринт, в Kanban - завдання.

1.3.3 Визначення Kanban-дошки

Дошка Kanban - це інструмент управління Agile-проектами, який допомагає наочно уявити завдання, обмежити обсяг незавершеної роботи і домогтися максимальної ефективності (або швидкості). Вона може допомогти командам Agile і DevOps впорядкувати повсякденну роботу. За допомогою карток і стовпців на дошці Kanban команди з технічних питань і сервісні команди можуть зрозуміти, який обсяг роботи слід взяти на себе, і виконати цей обсяг, дотримуючись принципів безперервного вдосконалення. [7]

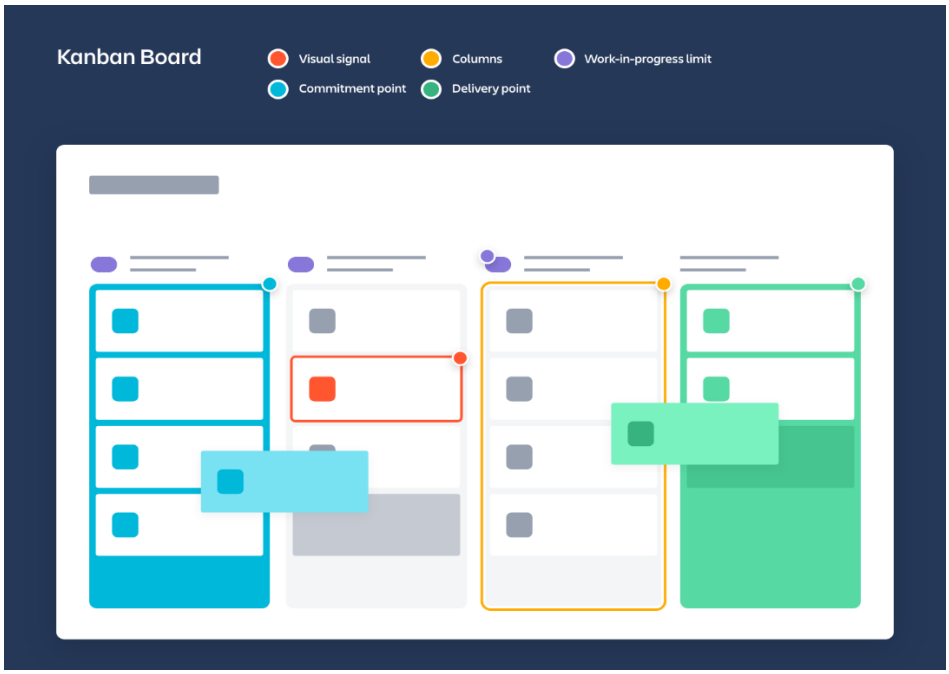

Рисунок 2 – Схематичний приклад Kanban дошки

Методика Kanban пройшла довгий шлях від своїх витоків у сфері бережливого виробництва, за що варто подякувати невеликій, але ефективній групі її прихильників. Праця Девіда Андерсона, в якій були позначені принципи методики Kanban, сприяла проникненню Kanban в світ розробки ПЗ і обслуговування, а книга Джима Бенсона і Тоніан Де Маріа Personal Kanban (Kanban в особистих цілях) допомогла поширенню Kanban в самих різних областях.

Девід Андерсон виділяє п'ять складових дошок Kanban: видимі сигнали, стовпці, ліміти незавершеної роботи, точка прийняття зобов'язань і точка поставки продукту.

1. **Видимі сигнали**. Першими на дошці Kanban кидаються в очі картки (стікери, листки та ін.). Kanban-команди виносять записи про всі проекти і робочі завдання на картки; одна картка, як правило, відповідає одному проекту або робочій задачі. Для Agile-команд кожна картка позначає одну призначену для користувача історію. Побачивши ці сигнали на дошці, учасники команди і зацікавлені сторони зможуть без проблем зрозуміти, над чим працює команда.

2. **Стовпці**. Ще одною відмінною ознакою дошки Kanban є стовпці. Вони символізують конкретні дії, які в сукупності складають «робочий процес». Картки переміщаються по робочому процесу до стадії завершення. Робочі процеси можуть бути простими і складатися лише з стовпців «Має бути зроблено», «В процесі» і «Завершено», а можуть бути набагато складнішими.

3. **Обмеження незавершеної роботи (WIP).** Обмеження WIP - це максимальна кількість карток, що може перебувати в одному стовпці одночасно. Якщо для стовпця вибрано обмеження WIP, що дорівнює 3, то в ньому не може бути більше трьох карток. Коли кількість карток в стовпці досягає максимуму, команда повинна зосередити зусилля на цих картках і передати їх далі, щоб на цю стадію робочого процесу могли вчинити нові картки. Обмеження WIP потрібні, щоб виявляти проблемні місця в робочому процесі і домогтися максимальної швидкості роботи. Обмеження WIP допомагають на ранніх етапах зрозуміти, чи не взяла команда на себе занадто багато завдань.

4. **Точка прийняття зобов'язань**. На дошці у Kanban-команд часто присутній беклог. Клієнти і учасники команди вносять в нього ідеї щодо проектів, до яких команда може звернутися, коли буде готова. У точці

прийняття зобов'язань команда вибирає ту чи іншу ідею, після чого починається робота над проектом.

5. **Точка поставки продукту**. Точка поставки продукту знаменує завершення робочого процесу команди Kanban. Багато команд беруть за точку поставки продукту момент, коли продукт або сервіс передаються в розпорядження клієнта. Мета команди - якнайшвидше перенести картки з точки прийняття зобов'язань в точку поставки продукту. Час, за який картка проходить з однієї точки в іншу, називається часом виконання. Kanban-команди постійно вдосконалюються, прагнучи звести час виконання до мінімуму.

1.3.4 Види канбан-дошок

Дошки Kanban можна застосовувати в багатьох сферах, від виробництва до управління персоналом та розробки ПЗ з використанням методик Agile і DevOps. Від того, до якої сфери потрібно пристосувати Kanban, часто залежить вибір дошки - цифровий або фізичної.

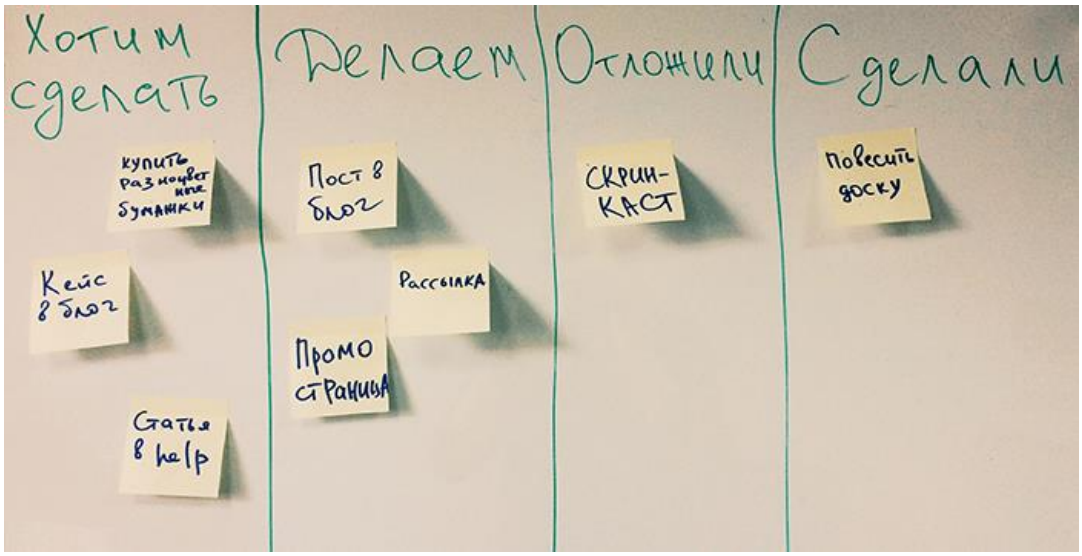

Рисунок 3 – Приклад фізичної Kanban дошки

Одна з переваг реальної дошки полягає в тому, що її не можна «вимкнути». Не можна відкрити нову вкладку на величезній маркерній дошці, що стоїть біля столу. Таку дошку легко підготувати, легко показати іншим, і часто з її допомогою найпростіше доносити інформацію в певних командах. Проте реальні дошки не підходять для віддалених команд.

Можна рекомендувати командам починати з реальної дошкою Kanban, тому що завдяки обговоренням з ранніх пір досягається висока швидкість ітерацій робочого процесу і завдання швидко проходять всі стадії.

Коли система Kanban здобула успіх у команд з розробки ПЗ і технічних команд, дошки Kanban зазнали цифрову трансформацію. Географічно розподілені команди можуть звертатися до цифрових дошок Kanban віддалено і в різний час.

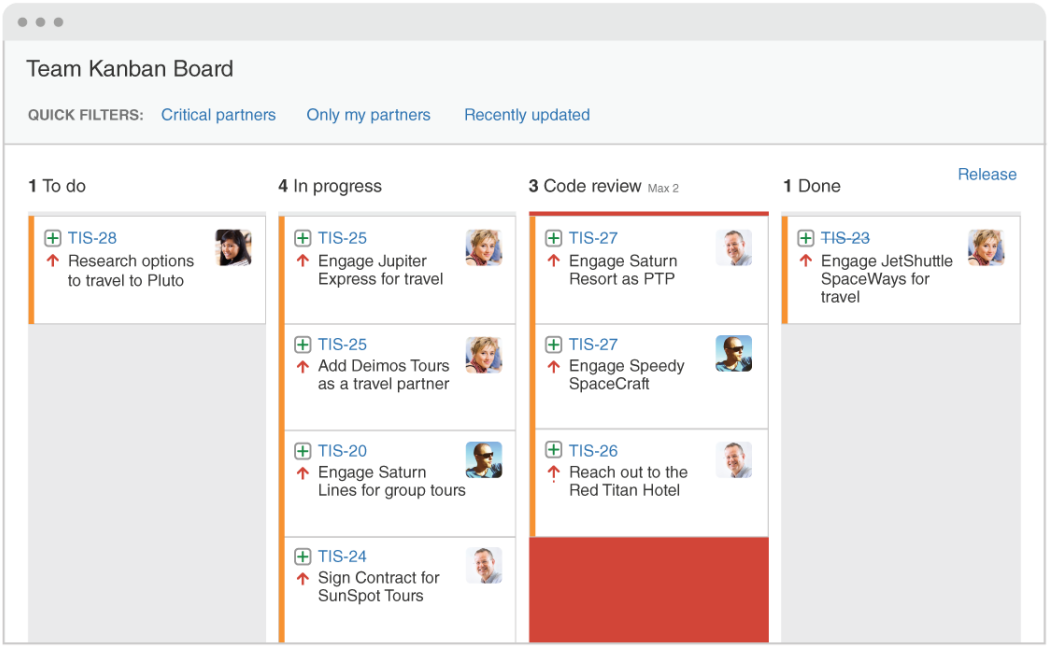

Рисунок 4 – Приклад цифрової Kanban дошки

Розглянемо популярні системи канбан для управління завданнями. Вони відрізняються візуальною привабливістю і зручним інтерфейсом. Користувачі відзначають їх супергнучкість.

Trello – це простий засіб для швидкого створення цифрової дошки Kanban. Всього за кілька натискань можна підготувати дошку з цифровими списками, що символізують стадії Kanban-процесу. Працювати з дошкою і керувати нею може вся команда.

Наприклад, можна створити списки «Беклог», «На черзі», «В процесі» і «Готово». Кожне завдання представлено у вигляді картки, яка переміщається зі списку в список по мірі того, як завдання потрапляє в чергу, над ним працюють і його виконують.

Цифрова дошка Kanban має наступні переваги: швидкість підготовки, простота надання спільного доступу і можливість відстежувати необмежену кількість обговорень і коментарів в будь-який час у мірі реалізації проекту. Звідки і коли б не зверталися учасники команди до дошки Kanban, вони побачать найактуальніший статус проекту.

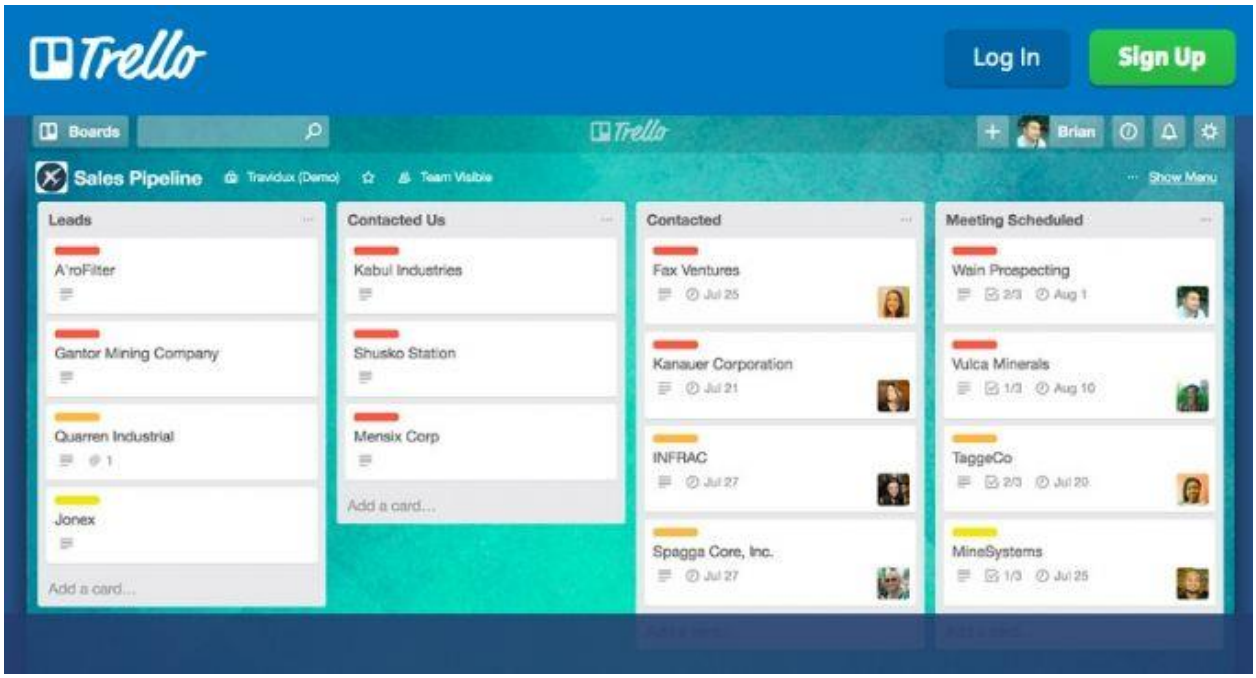

Рисунок 5 – Kanban дошка в додатку Trello

Бувають зовсім прості цифрові дошки Kanban, в той час як деякі більш продумані і передбачають більше можливостей налаштування. Командам, яким потрібні додаткові функції, наприклад обмеження WIP і контрольні графіки, слід вибирати інструмент з більш широкими можливостями, такий як Jira Software. У Jira за замовчуванням доступний шаблон проекту Kanban, щоб команди Kanban могли без зволікання приступити до роботи. Команда може просто створити проект, налаштувати робочий процес і дошку в залежності від потреб, встановити обмеження WIP, створити доріжки swimlane і навіть включити беклог, щоб було зручніше розставляти пріоритети.

Також є досить популярна програма YouTrack - це інструмент управління проектами і баг-трекер, який легко адаптується під ваші процеси. Завдяки йому можна відстежувати завдання, планувати спринти і релізи, створювати робочі процеси, використовуючи agile-дошки, діаграми Ганта, інформаційні панелі і тайм-трекінг.

Agile-дошки в YouTrack розроблені для того, щоб допомогти командам слідувати широкому спектру гнучких процесів управління проектами. Можна створювати дошки:

- Scrum;
- Kanban;
- Scrumban.

Користувачам доступні приватні звіти, та ті що можна налаштувати , які допомагають аналізувати проекти і дії команд і керувати ними. Гнучкість налаштувань допомагає скласти уявлення про те, як працює команда, і яким чином проблеми розподіляються по проекту. Також можна додавати звіти у вигляді готових віджетів або створювати свої власні віджети, якщо потрібно більше даних. [8]

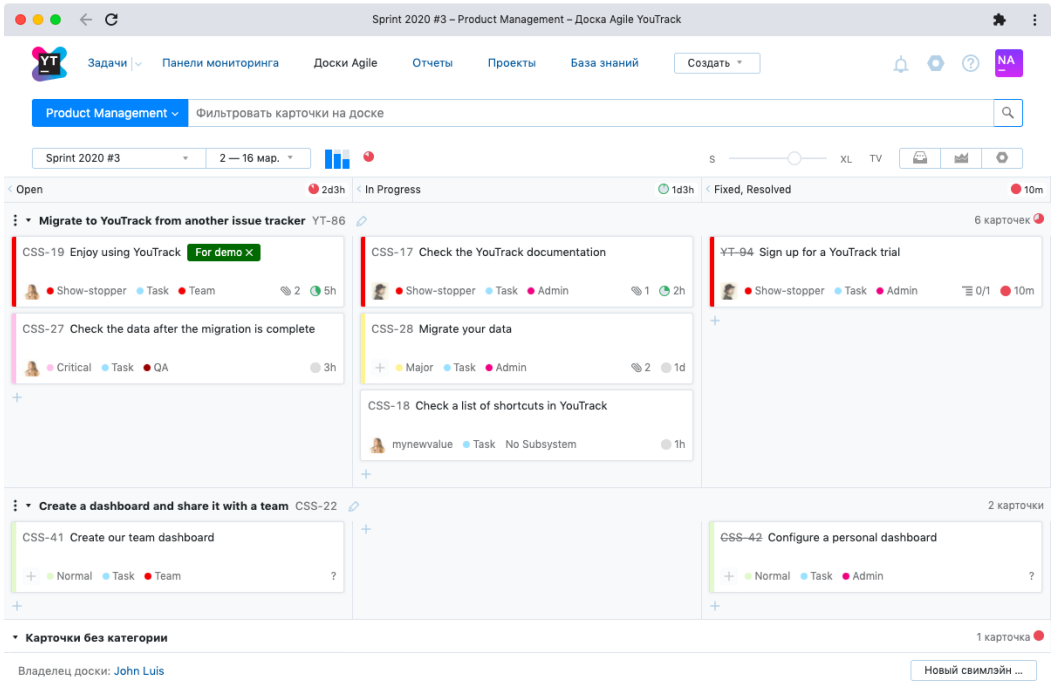

Рисунок 6 - Kanban дошка в додатку YouTrack

Основними перевагами YouTrack є:

1) Функція розумного пошуку (опція автозаповнення дозволяє швидко знаходити потрібну інформацію);
2) Відстеження багів і проблем;
3) Ярлики;
4) Налаштування;

5) Персональна панель управління з можливістю відстеження проекту і команд в реальному часі.

Отже, коріння гнучкого підходу тягнуться углиб століть, але все ж таки буде логічніше вважати стартовою точкою розвитку agile-методології XX століття.

Важко виділити конкретного засновника agile-підходу, тому що кожен вніс свій внесок в розробку концепції agile підходів - спочатку це був фізик і статистик Уолтер Шухарт, потім його учень Вільям Демінг продовжив розвивати цю концепцію яка з часом була названа як Цикл Демінга.

Пізніше у 1986 році Хіротака Такеучі спільно зі своїм партнером Ікуджіро Нонакою опублікували в журналі «Harvard Business Review» статтю під назвою «The New New Product Development Game» в якій автори виділили коцепцію командно-орієнтованого підхіду розробки продукту. Нарешті у 1995 році Джефф Сазерленд разом зі своїм колегою Кеном Швабером змогли об’єднати ці дві концепції та формалізувати підхід давши йому назву «Scrum» і в 1995 році цей підхід був представлений всьому світу.

Пізніше Девід Андерсон створює нову концепцію agile-підходу, якій дав назву Kanban і популярізує її завдяки своїй книзі «Kanban. Альтернативний шлях в Agile».

Основна різниця між Scrum і Kanban - в довжині ітерацій (або спринтів). У Scrum середня тривалість спринту - 2 тижні, в Kanban завдання розробнику можна надавати хоч кожен день. При роботі за методологією Scrum зазвичай планують витрати часу на кожну задачу в спринті, чого не прийнято робити при Kanban-підході, Kanban дає більше гнучкості, якщо під гнучкістю розуміти частоту зміни пріоритетів.

Отже, метою Scrum є завершення спринту, а метою Kanban є завершення задачі, на практиці ці два підходи часто об’єднують, беручи можливість планування Scrum-підходу при цьому зберігаючи гнучкість Kanban.

Для візуалізації agile-підходів використовують канбан-дошки: фізичні та електронні. Вони дозволяють зробити робочий процес відкритим і зрозумілим для всіх фахівців, що важливо, коли у команди немає одного формального керівника.

Також були розглянуті сучасні реалізації канбан-дошок з їх перевагами та недоліками. Метою дипломної роботи було створення канбан-дошки, яка відповідає основним характеристикам її існуючих аналогів.

Після проведення дослідження сучасних реалізацій канбан-дошок можна зробити висновок, що більшість із них - це веб-додатки засновані на клієнт-серверній архітектурі.

2 ДОСЛІДЖЕННЯ ПРОЦЕСУ РОЗРОБКИ СУЧАСНИХ ВЕБ-ДОДАТКІВ

2.1 Поняття веб-додаток

Веб-додаток являє собою клієнт-серверну програму. Це означає, що він має сторону клієнта та сторону сервера. Термін "клієнт" тут стосується програми, яку людина використовує для запуску програми, в більшості випадків це веб-браузер. Веб-додаток являє собою частину середовища клієнт-сервер, де багато комп’ютерів обмінюються інформацією.

Веб-програми можуть розроблятися з різних причин і використовуватися компаніями чи приватними особами. Найчастіше це використовується для полегшення спілкування або придбання речей в Інтернеті. Також співробітники можуть співпрацювати над проектами та працювати над спільними документами з веб-додатками. Вони можуть створювати звіти, файли та обмінюватися інформацією з будь-якого місця та з будь-якого пристрою.

Веб-додаток має багато переваг, серед яких:

- Його не потрібно встановлювати на жорсткий диск кожного користувача.
- Він вимагає меншої підтримки та обслуговування з боку бізнесу та нижчих технічних вимог до комп’ютера користувача.
- Веб-додатки завжди оновлені, оскільки оновлення застосовуються централізовано.
- Усі користувачі можуть отримати доступ до однієї версії, що усуває будь-які проблеми сумісності.
- Ви можете отримати доступ до веб-програм у будь-якому місці за допомогою веб-браузера.
- Поки браузер сумісний, веб-програми можуть працювати на декількох платформах, незалежно від операційної системи або пристрою.

- Веб-програми звільняють розробника від відповідальності за створення клієнта, сумісного з певним типом комп’ютера або певної операційної системи.

Слід зазначити що більшість веб-додатків засновані на архітектурі **клієнт-сервер**, де клієнт вводить інформацію, в той час як сервер отримує та зберігає інформацію.

2.2 Визначення клієнт-серверної архітектури

Клієнт-серверна архітектура являє собою ієрархічну мережу, яка складається з вузлів-клієнтів (їх може бути від одного і до необмеженої кількості) і центрального сервера, через який виконується зберігання і обробка даних, а так само передача їх в обох напрямках. На поточний момент велика частина Інтернету і локальні мережі використовують саме архітектуру Клієнт-Сервер для прийому і передачі даних і медіаконтенту. Найпростіша схема клієнт-серверної архітектури в локальній мережі виглядає так:

Рисунок 7 - схема клієнт-серверної архітектури в локальній мережі

Розташування компонентів на стороні клієнта або сервера визначає наступні основні моделі їх взаємодії в рамках дволанкової архітектури:

- Сервер терміналів - розподілене представлення даних.
- Файл-сервер - доступ до віддаленої бази даних і файлових ресурсів.
- Сервер БД - віддалене уявлення даних.
- Сервер додатків - віддалений додаток.

У більшості випадків клієнтом є програма, за допомогою якої користувач може взаємодіяти з сервером, клієнт дає можливість користувачу бачити дані у вигляді веб-сторінок, а також за допомогою зручного інтерфейсу дає можливість вводити та змінювати дані на сервері за допомогою запитів на сервер (найчастіше це HTTP-запити, рідше - TCP).

Веб-сервер - це сервер, що приймає HTTP-запити від клієнтів і видає їм HTTP-відповіді. Веб-сервером називають як програмне забезпечення, яке виконує функції веб-сервера, так і безпосередньо комп'ютер, на якому це програмне забезпечення працює. [9] Найбільш поширеними видами ПЗ веб-серверів є Apache, IIS і NGINX. На веб-сервері функціонує тестовий додаток,

який може бути реалізовано з застосуванням найрізноманітніших мов програмування: PHP, Python, Ruby, Java, Perl та ін.

Веб-сервер обробляючи запити клієнта, віддає дані клієнту у специфічному форматі (JSON, XML) якщо це API (прикладний програмний інтерфейс), або відразу віддає готову веб-сторінку в форматі HTML у відповіді на клієнтський запит.

Система управління бази даних, або СУБД - це інформаційна модель, що дозволяє упорядковано зберігати дані про об'єкт або групі об'єктів, що володіють набором властивостей, які можна категоризувати. Бази даних функціонують під управлінням так званих систем управління базами даних. Найпопулярнішими СУБД є MySQL, MS SQL Server, PostgreSQL, Oracle - всі є клієнт-серверними.

База даних фактично не є частиною веб-сервера, але більшість програм просто не можуть виконувати всі покладені на них функції без неї, так як саме в базі даних зберігається вся динамічна інформація - додатки (облікові, призначені для користувача дані і ін).

Як і будь-яка інша технологія, клієнт-серверна архітектура має свої переваги і свої недоліки. Розглянемо їх.

Переваги:

- Виконання більшої частини роботи потужної серверної частини при мінімумі навантаження на клієнта.
- Основна частина даних зберігаються на сервері. При цьому, як правило, він краще захищений від різного виду погроз, ніж звичайний клієнтський ПК.
- Можливість більш чіткого розмежування повноважень доступу до різних рівнів інформаційної системи. Кожному клієнту - свій рівень доступу.
- Кросплатформеність. Простіше кажучи, будь-який клієнт може працювати з ресурсами сервера незалежно від операційної системи.

- Зменшення навантаження на мережу з огляду на те, що клієнт в основному передає серверу команди, а той вже їх виконує.

Недоліки:

- Вихід з ладу сервера може привести до непрацездатності всієї системи, що його використовує.
- Висока вартість серверного обладнання та його обслуговування (зокрема, може знадобитися окремий фахівець для обслуговування).
- Високе навантаження на серверне обладнання і канал зв'язку до нього.

2.3 API та його роль у побудові сучасного програмного забезпечення

Абревіатура API розшифровується як «Application Programming Interface» (інтерфейс програмування додатків, прикладний інтерфейс програми). Більшість великих компаній на певному етапі розробляють API для клієнтів або для внутрішнього використання.

API - це, в першу чергу, інтерфейс, який дозволяє розробникам використовувати готові блоки для побудови програми. У випадку з розробкою мобільних додатків в ролі API може виступати бібліотека для роботи з "розумним будинком" - всі нюанси реалізовані в бібліотеці і ви лише звертаєтеся до цього API в своєму коді.

У разі веб-додатків, API може передавати інформацію в відмінному від стандартного HTML форматі, завдяки чому їм зручно користуватися при написанні власних програм. Сторонні загальнодоступні API найчастіше віддають дані в одному з двох форматів: XML або JSON. На випадок, якщо робити API для свого застосування, то JSON набагато більш лаконічний і простий в читанні, ніж XML, а сервіси, що надають доступ до даних в XML-форматі, поступово відмовляються від останнього.

Якщо взяти який небудь додаток - наприклад, Github - має свій API, яким можуть скористатися інші розробники. Те, як вони будуть користуватися ним залежить від можливостей, які надає API і від того, наскільки добре працює фантазія у розробників. API Github дозволяє, наприклад, отримувати інформацію про користувача, його аватар, читачів, репозиторіях і безліч інших корисних і цікавих відомостей. [10]

Існує кілька ситуацій, в яких компанії можуть захотіти створити API для власного написаного додатку:

- **Мобільний додаток**. Безліч мобільних додатків для різних сервісів працюють при використанні API цих самих сервісів. Наприклад розробники описали API, зробили простий мобільний додаток і клієнт зі

смартфоном буде отримувати інформацію в свій пристрій саме через API. Це зручно, це розумно, це має сенс.

- **Опенсорс**. Насправді, якщо у додатку склалася певна аудиторія, яка користується ним, буде логічно обернути це собі на користь, та на користь аудиторії. Можна створити API, за допомогою якого користувачі при бажанні зможуть створити нові клієнти для додатку, нові сервіси на його основі і, можливо, розкриють нові його грані.
- **Максимальне розділення фронтенда і бекенда**. Наприклад, при використанні фронтенд-фреймворків.

Слід відзначити те, що в наш час неможливо створити якісні та корисні сервіси без використання бібліотек API, оскільки вони необхідні як програмістам, для написання програмного забезпечення і додатків, так і різним сервісам, для надання послуг з обслуговування клієнтів і з кожним роком роль і область застосування API тільки розширюється.

2.4 Види веб-API

Веб-API являє собою програмний інтерфейс, що складається з однієї або більше загальнодоступних кінцевих точок до визначеної системи повідомлень запиту-відповіді, як правило, вираженої у форматі JSON або XML, яка виставляється через WEB - найчастіше за допомогою HTTP веб-сервера.

Найвідоміші типи API:

1) Віддалений виклик процедур (Remote Procedure Call - RPC)
2) Простий протокол доступу до об'єктів (Simple Object Access Protocol - SOAP)
3) Передача стану уявлення (Representational State Transfer REST)

SOAP (Simple Object Access Protocol) - Дані передаються в форматі XML.

Переваги:

- галузевий стандарт за версією W3C;
- наявність суворої специфікації;
- широка підтримка в продуктах Microsoft,
- однозначність.

Недоліки:

- складність реалізації;
- складність / ресурсомісткість парсинга XML-даних.

Будь-яке повідомлення в протоколі SOAP - це XML документ, що складається з наступних елементів (тегів):

1) Envelope. Обов'язковий елемент. Визначає початок і закінчення повідомлення.
2) Header. Необов'язковий елемент - заголовок. Містить елементи, необхідні для обробки самого повідомлення. Наприклад, ідентифікатор сесії.

3) Body. Основний елемент, містить основну інформацію повідомлення. Обов'язковий.

4) Fault. Елемент, що містить інформацію про помилки, що виникають в процесі обробки повідомлення. Необов'язковий.

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="https://www.w3.org/2003/05/soap-envelope/"
               soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding/">
    <soap:Body>
      <m:GetPrice xmlns:m="https://www.w3schools.com/prices">
        <m:Item>Apples</m:Item>
      </m:GetPrice>
    </soap:Body>
</soap:Envelope>
```

Рисунок 8 - Приклад SOAP запиту

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="https://www.w3.org/2003/05/soap-envelope/"
               soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding/">
    <soap:Body>
      <m:GetPriceResponse xmlns:m="https://www.w3schools.com/prices">
        <m:Price>1.90</m:Price>
      </m:GetPriceResponse>
    </soap:Body>
</soap:Envelope>
```

Рисунок 9 - Приклад SOAP відповіді

REST (Representational State Transfer) - насправді архітектурний стиль, а не протокол. На відміну від SOAP, REST не підкріплений офіційним стандартом. Фактично, він ґрунтується на угодах. Веб-сервіс, побудований з урахуванням всіх вимог і обмежень архітектурного стилю, можна назвати RESTful веб-сервісом.

REST не використовує конвертацію даних при передачі, дані передаються в початковому вигляді - це знижує навантаження на клієнт веб-

сервісу, але збільшує навантаження на мережу. Управління даними відбувається за допомогою методів HTTP:

- GET - отримати дані;
- POST - додати дані;
- PUT - змінити дані;
- DELETE - видалити дані.

Переваги:

- простота реалізації;
- економічність в плані ресурсів;
- не вимагає програмних надбудов (json_decode є майже в кожній мові).

Недоліки:

- відсутність специфікації;
- неоднозначність методів управління даними.

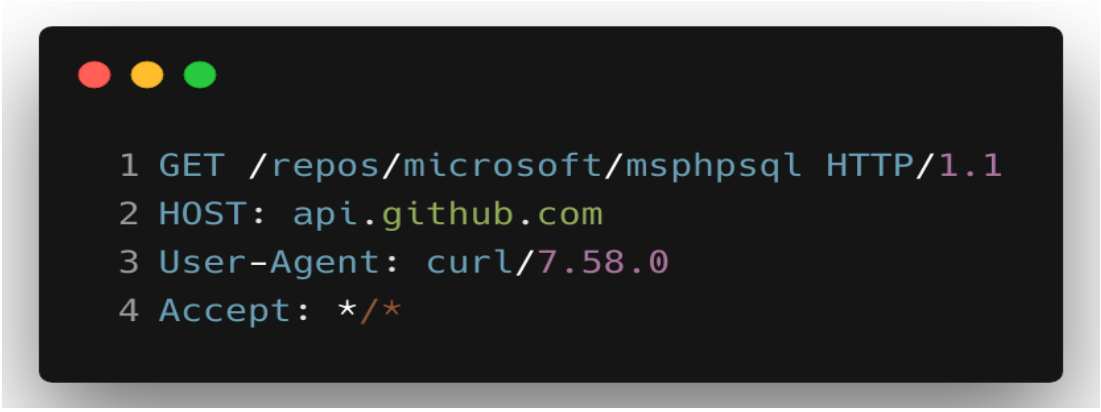

Рисунок 10 - Приклад REST запиту

Рисунок 11 - Приклад REST відповіді

RPC (Remote Procedure Call) - це специфікація, яка дозволяє віддалено виконувати функцію в іншому контексті. RPC розширює поняття локального виклику процедури, але поміщає його в контекст HTTP API.

Проблематичність початкового XML-RPC пов'язана зі складнощами в забезпеченні типів даних для корисних навантажень XML. Пізніше API RPC задіяв більш конкретну специфікацію JSON-RPC, яка вважається більш простою альтернативою SOAP. gRPC - версія RPC, розроблена компанією Google в 2015 році. Завдяки підключенню підтримки балансування навантаження, трасування, перевірки працездатності і аутентифікації gRPC добре підходить для мікросервісів.

Переваги:

- Простота і зрозумілість взаємодій
- Легкість додавання функцій
- Висока продуктивність

Недоліки:

- Щільний зв'язок з базовою системою
- Низька ймовірність виявлення
- Вибух функцій

[11]

```
{
    "jsonrpc": "2.0",
    "method": "user.login",
    "params": {
        "user": "Admin",
        "password": "zabbix"
    },
    "id": 1
}
```

Рисунок 12 - Приклад RPC запиту

Рисунок 13 - Приклад RPC відповіді

Отже, веб-додаток являє собою клієнт-серверну програму. Це означає, що він має сторону клієнта та сторону сервера. Термін "клієнт" тут стосується програми, яку людина використовує для запуску програми, в більшості випадків це веб-браузер. При побудові простих веб-додатків часто сервер може відповідати клієнту готовою веб-сторінкою, але при побудові складних веб-додатків все частіше можна побачити підхід при якому клієнт та сервер обмінюються інформацією через прикладний програмний інтерфейс (API).

Завдяки клієнт-серверній архітектурі клієнтська частина часто буває мультиплатформенною, а точніше - може бути не тільки веб додатком, а ще й мобільним або комп’ютерним додатком, які, в свою чергу, можуть взаємодіяти з одним і тим же сервером по одному і тому же веб-API.

Веб-API - використовується в веб-розробці і містить, як правило, певний набір HTTP-запитів, а також визначення структури HTTP-відповідей, для вираження яких використовують XML- або JSON-формат.

Найвідомішими видами веб-API:

- Віддалений виклик процедур (Remote Procedure Call - RPC)
- Простий протокол доступу до об'єктів (Simple Object Access Protocol - SOAP)
- Передача репрезентативного стану (Representational State Transfer - REST)

При розробці kanban-дошки для даної дипломної роботи було розроблене власне API з використанням стандарту REST, за рахунок його популярності на сьогодні при написанні мультиплатформенних додатків, а також завдяки зручному та зрозумілому синтаксису.

3 ПРОЦЕС РОЗРОБКИ WEB-ДОДАТКУ ДЛЯ ПОСТАНОВКИ ТА КЕРУВАННЯ ЗАДАЧ З РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗА МЕДОЛОГІЄЮ AGILE

3.1 Перелік використаного стеку технологій

В ході дослідження був розроблений web-додаток з використанням наступних технологій:

1. **PHP.** У якості мови програмування для бекенду був використаний PHP версії 8.0 за його зручність при конструюванні веб-додатку.

PHP (рекурсивний акронім словосполучення PHP: Hypertext Preprocessor) - це мова програмування загального призначення з відкритим вихідним кодом. [12]

PHP спеціально сконструйований для веб-розробок та його код може впроваджуватися безпосередньо в HTML. На сьогодні підтримується переважною більшістю хостинг-провайдерів і є одним з лідерів серед мов, що застосовуються для створення динамічних веб-сайтів.

Популярність в області розробки веб-додатків визначається наявністю великого набору вбудованих засобів і додаткових модулів для розробки веб-додатків, а саме:

- автоматичне вилучення POST- і GET-параметрів, а також змінних оточення веб-сервера в спеціальні масиви;
- взаємодія з великою кількістю різних систем управління базами даних через додаткові модулі (MySQL, SQLite, PostgreSQL, Oracle Database, Microsoft SQL Server та інші);
- автоматизована відправка HTTP-заголовків;
- робота з HTTP-авторизацією;
- робота з cookies і сесіями;
- робота з локальними і віддаленими файлами, сокетами;
- обробка файлів, що завантажуються на сервер;
- робота з формами.

Оскільки PHP є спеціально сконструйованою мовою для розробки веб-додатків і має в собі великий набір вбудованих засобів та додаткових модулів які полегшують розробку, тому він ідеально підходить для розробки визначеного продукту.

2. **JavaScript**. При написанні фронтенду була використана мова JavaScript специфікації ECMAScript 6 за рахунок можливості виконання її скриптів клієнтом використовуючи звичайний веб-браузер.

JavaScript - це прототипно-орієнтована, мультипарадигменна мова з динамічною типізацією, яка підтримує об'єктно-орієнтований, імперативний і декларативний (наприклад функціональне програмування) стилі програмування. [13]

JavaScript створювався як скриптова мова для Netscape. Після чого він був відправлений в ECMA International для стандартизації (ECMA - це асоціація, діяльність якої присвячена стандартизації інформаційних і комунікаційних технологій). Це призвело до появи нового мовного стандарту, відомого як ECMAScript.

Подальші версії JavaScript вже були засновані на стандарті ECMAScript. Простіше кажучи, ECMAScript - стандарт, а JavaScript - найпопулярніша реалізація цього стандарту.

3. **Symfony.** При написанні бекенду був використаний фреймворк заснований на мові PHP, це – Symfony версії 5.2.

Symfony - це зручний фреймворк для написання бекенду на мові PHP, він є гнучким тому ніяк не зв’язує розробника у написанні коду, але у той же час має великий набір готових компонентів які вирішують цілу низку задач з якими зіштовхується розробник при написанні бекенду, а саме:

- Написання API
- Взаємодія з базою даних

- Обробка та валідація запитів
- Авторизація користувачів
- Кешування запитів
- Маршрутизація

Отже, Symfony є зручним фреймворком який вирішує майже всі завдання при конструюванні веб-додатку тому він і був використаний при написанні канбан-дошки.

4. Vue.js. При написанні фронтенду для канбан дошки був використаний фреймворк написаний на мові JavaScript – Vue.js версії 2.6.11

Vue.js - це прогресивний фреймворк для створення користувацьких інтерфейсів, він має у собі багато готових компонентів які вирішують основні задачі які стоять перед розробником при конструюванні фронтенду для веб-додатку, а також велику кількість користувацьких бібліотек у вільному доступі. Завдяки Vue.js розробник має змогу розділити бізнес логіку та логіку рендерингу (HTML-розмітку) завдяки розділенню цих логік код написаний на Vue.js легше сприймати і підтримувати при роботі в команді.

Також однією з головних переваг Vue.js є те що він ідеально підходить для створення односторінкових застосунків зі складною бізнес-логікою.

Односторінковий застосунок (англ. single-page application, **SPA**) - це веб-застосунок чи веб-сайт, який вміщується на одній сторінці з метою забезпечити користувачу досвід близький до користування настільною програмою. [14]

В односторінковому застосунку весь необхідний код - HTML, JavaScript, та CSS - завантажується разом зі сторінкою, або динамічно довантажується за потребою, зазвичай у відповідь на дії користувача. Сторінка не оновлюється і не перенаправляє користувача до іншої сторінки у

процесі роботи з нею. Взаємодія з односторінковим застосунком часто включає в себе динамічний зв'язок з веб-сервером.

Канбан-дошка є по своїй суті односторінковим застосунком зі складною бізнес логікою, тому фреймворк Vue.js є інструментом який ідеально підходить для цієї задачі.

5. PostgreSQL. У якості системи керування базами даних для програмного продукту була обрана PostgreSQL версії 12.6.

PostgreSQL - об'єктно-реляційна система керування базами даних (СУБД). Є альтернативою як комерційним СУБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

PostgreSQL є потужною об'єктно-реляційною системою баз даних з відкритим кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають та масштабують найскладніші робочі навантаження на базу даних. Розвиток PostgreSQL бере свій початок в 1986 році в рамках проекту POSTGRES в Університеті Каліфорнії в Берклі і має більш ніж 30 років активного розвитку на базовій платформі, за цей час

PostgreSQL заслужив відмінну репутацію завдяки своїй перевіреній архітектурі, надійності, цілісності даних, надійному набору функцій, розширюваності та відданості спільноти інтузіастів, що стоїть за цим програмним забезпеченням для постійного забезпечення продуктивних та інноваційних рішень.

PostgreSQL має безліч функцій, спрямованих на те, щоб допомогти розробникам створювати додатки, адміністраторам захищати цілісність даних та створювати відмовостійкі середовища, PostgreSQL розповсюджується як безкоштовна СУБД з відкритим вихідним кодом.

Також однією з особливостей PostgreSQL є її розширюваність, а саме - можливість визначати власні типи даних, створювати власні функції, навіть писати код з використанням різних мов програмування, не перекомпілюючи базу даних.

Отже, PostgreSQL була обрана як система керування базами даних завдяки своїй надійності, простоті у використанні та безкоштовній моделі розповсюдження.

6. Doctrine ORM. Для більш ефективної взаємодії з базою даних була використана ORM Doctrine версії 2.8.

Doctrine - об'єктно-реляційний проектор (ORM) для PHP 5.3.0+, який базується на шарі абстракції доступу до БД (DBAL). Однією з ключових можливостей Doctrine є запис запитів до БД на власному об'єктно-орієнтованому діалекті SQL, званий DQL (Doctrine Query Language). [15]

Doctrine базується на паттерні DataMapper для роботи з базою даних.

Наприклад, якщо розробник хоче створити нового користувача в базі даних, він може більше не писати SQL запити, а написати PHP код, позначений на рисунку 14.

```php
$user = new User();
$user->name = "Jack";
$user->password = "adg1we25";
$user->save();
```

Рисунок 14 – приклад коду з використанням Doctrine ORM

Після виконання цього коду новий користувач буде збережений у базі даних, головна перевага такого підходу в тому що програмний код не має чіткої прив’язки до конкретної реалізації СУБД.

Отже, Doctrine це об'єктно-реляційний проектор який полегшує процес взаємодії з базою даних, при цьому він базується на ефективних архітектурних рішеннях які дозволяють не прив’язуватися до конкретної системи керування базою даних та писати більш інтуїтивно-зрозумілий код.

7. Docker. Для більш зручного розгортання інфраструктури була використана технологія Docker.

Docker - це відкрита платформа для розробки, доставки та запуску додатків. Docker дозволяє відокремити програми від інфраструктури, щоб швидко розгортувати програмне забезпечення. За допомогою Docker розробник може керувати інфраструктурою так само, як і програмами. Технологія Docker базується на швидкому розгортанні інфраструктури з використанням docker-контейнерів.

Docker-контейнер - це ізольована віртуальна машина, яка створюється з заздалегідь регламентованим набором конфігурацій, бібліотек, утиліт та іншого програмного забезпечення яке буде використане в подальшому всередині контейнера.

Завдяки використанню Docker для швидкої доставки, тестування та розгортання коду, користувач може скоротити затримку між написанням коду та його запуском у виробництво.

Отже, Docker є зручною платформою для розгортання інфраструктури програмного забезпечення у будь-якому середовищі яке підтримує цю платформу з використанням заздалегідь налаштованих docker-контейнерів.

3.2 Проектування бази даних

Для кожного з інформаційних об’єктів канбан-дошки було створено окрему таблицю в базі даних. У таблицях БД будуть зберігатися реальні дані, що характеризують відповідні об’єкти. Розроблені структури таблиць приводяться нижче:

User. Таблиця user відображає роль користувача у системі, та має наступні поля:

- id – тут і далі поле «id» відіграє роль первинного ключа (primary key)
- email – email користувача у системі
- password – захешований пароль користувача
- roles – json з можливими ролями користувача
- api_token – спеціальне поле яке зберігає api-token сесії користувача

```
user (
    id integer default nextval('user_id_seq'::regclass) not null
        constraint user_pkey
            primary key,
    email varchar(255) not null,
    password varchar(255) not null,
    roles json not null,
    api_token varchar(255)
);
```

Рисунок 15 - Схема таблиці user

Task. Таблиця task використовується у системі для зберігання інформації про задачі, та має наступні поля:

- id – первинний ключ
- title – заголовок задачі
- text – опис задачі
- date_created – відображає дату створення задачі
- priority_id – зовнішній (foreign) ключ на таблицю task_priority
- column_id – зовнішній (foreign) ключ на таблицю task_column
- swimlane_id – зовнішній (foreign) ключ на таблицю task_swimlane

- owner_id – зовнішній (foreign) ключ на таблицю user і відображає інформацію про користувача який створив дану задачу
- executor_id – зовнішній (foreign) ключ на таблицю user і відображає інформацію про користувача який визначений як виконавець даної задачі

```
task (
    id           integer default nextval('task_id_seq'::regclass) not null
        constraint task_pkey
            primary key,
    title        varchar(100) not null,
    text         text,
    date_created timestamp(0) not null,
    priority_id  varchar(255) not null,
    column_id    varchar(255) not null,
    swimlane_id  varchar(255) not null,
    owner_id     varchar(255) not null,
    executor_id  varchar(255) not null
);
```

Рисунок 16 - Схема таблиці task

Task swimlane. Таблиця task_swimlane використовується у системі для зберігання інформації про те, які є дошки з задачами у системі, та має лише два поля:

- id – первинний ключ
- name – назва дошки

```
task_swimlane (
    id integer default nextval('task_swimlane_id_seq'::regclass) not null
        constraint task_swimlane_pkey
            primary key,
    name varchar(255) not null
);
```

Рисунок 17 - Схема таблиці task_swimlane

Task column. Таблиця task_column використовується у системі для зберігання інформації про те, які є стовбці (або статуси) задач для конкретної дошки, та має наступні поля:

- id – первинний ключ
- name – назва стовбця
- position – його позиції відносно інших стовбців для даної дошки

- swimlane_id - зовнішній (foreign) ключ на таблицю task_swimlane, використовується для прив’язки відображення колонки до конкретної дошки

```
task_column (
    id integer default nextval('task_column_id_seq'::regclass) not null
        constraint task_column_pkey
            primary key,
    name varchar(255) not null,
    position integer not null,
    swimlane_id integer constraint fk_46fa03ad761b6b99 references
task_swimlane
);
```

Рисунок 18 - Схема таблиці стовбців с задачами task_comment

Task priority. Таблиця task_priority використовується у системі для відображення інформації про пріорітет задач, та має наступні поля:

- id – первинний ключ
- name – назва пріорітету
- color – використовується для відображення пріорітету задачі спеціальним кольором (наприклад задачі з високим пріорітетом матимуть червоний колір)

```
table task_priority
(
    id integer default nextval('task_priority_id_seq'::regclass) not null
        constraint task_priority_pkey
            primary key,
    name  varchar(255) not null,
    color varchar(100) not null
);
```

Рисунок 19 - Схема таблиці task_priority

Task comment. Таблиця task_comment використовується у системі для зберігання інформації про коментарі до задач які можуть залишати користувачі та має наступні поля:

- id – первинний ключ
- text – текст коментаря
- date_created – дата створення коментаря

- task_id – зовнішній (foreign) ключ який використовується для прив’язки коментаря до конкретної задачі
- user_id - зовнішній (foreign) ключ який використовується для прив’язки коментаря до користувача який його написав

```
task_comment (
    id integer default nextval('task_comment_id_seq'::regclass) not null
        constraint task_comment_pkey
            primary key,
    text         text,
    date_created timestamp(0) not null,
    task_id      varchar(255) not null,
    user_id      varchar(255) not null
);
```

Рисунок 20 - Схема таблиці task_comment

В даній інтерпретації бази даних більшість зв’язків між таблицями зводяться до типу **“many-to-one”** (багатьом записам в одній таблиці може відповідати лише один запис в іншій), нижче наведені конкретні зв’язки між таблицями цього типу:

- task.priority_id – task_priority.id
- task.column_id – task_column.id
- task.swimlane_id – task_swimlane.id
- task_column.swimlane_id – task_swimlane.id
- task_comment.user_id – user.id
- task_comment.task_id – task.id

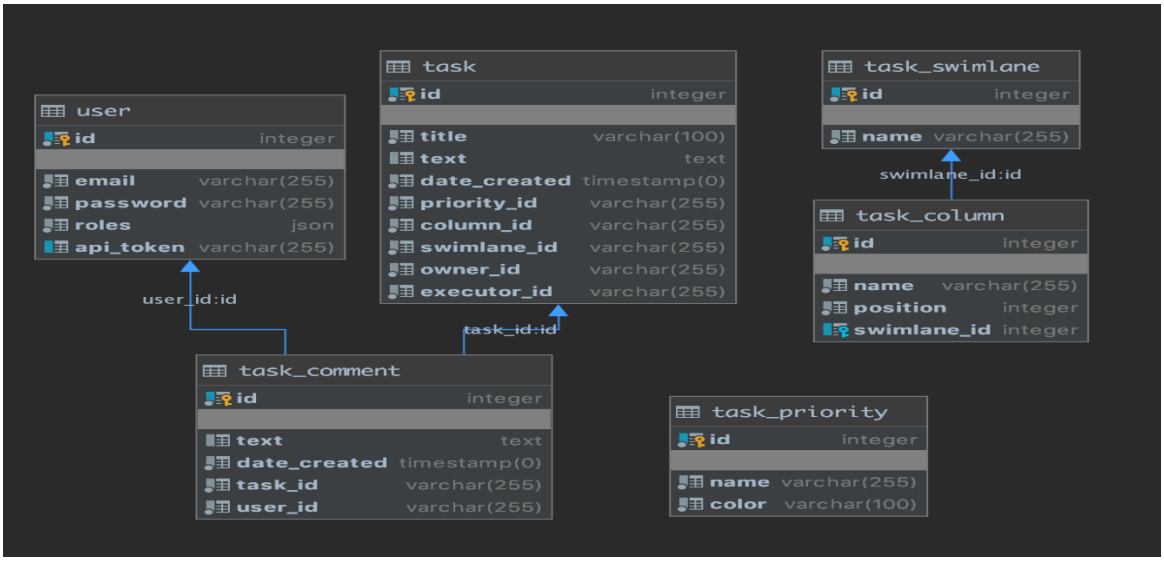

Рисунок 21 - UML-діаграма бази даних

Розроблений прототип бази даних зображений на рисунку 21 був використаний в подальшому при побудові веб-додатку та містить у собі всю необхідну інформацію для коректної роботи додатку.

3.3 Опис розробленого веб-API

При розробці kanban-дошки було розроблене API з використанням стандарту REST, за рахунок його популярності на сьогодні, а також завдяки зручному та зрозумілому синтаксису. Більшість запитів до API є захищеними і мають супроводжуватися спеціальним токеном авторизації.

3.3.1 Авторизація користувача

Авторизація користувача заснована на використанні коцепції JWT. JWT - це стандарт токена доступу на основі JSON, стандартизованого в RFC 7519. Використовується для верифікації тверджень.[14] Якщо конкретніше - при авторизації користувача з використанням email-а та пароля у форматі JSON, у відповідь буде відправлено спеціальний token авторизації, який має бути використаним для подальших запитів на API-сервер, приклад такого запиту зображено на рисунку 22, а приклад відповіді на рисунку 23:

```
POST taskmanager-backend.com/api/login_check
{
    "username": "taskmanager-admin@gmail.com",
    "password": "rmUlterEaDmi"
}
```

Рисунок 22 - Приклад запиту на ***/api/login_check***

```
Status: 200 OK
{
    "token":
    "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE2MjA2Mzg5MzIsImV4cCI6MTYyMTIzO
    DkzMiwicm9sZXMiOlsiUk9MRV9BRE1JTiIsIlJPTEVfVVNFUiJdLCJ1c2VybmFtZSI6InRhc2ttYW5h
    Z2VyLWFkbWluQGdtYWlsLmNvbSJ9.rtmWGdBJsnGoMaaZ4acTJv7A-
    jWkMCYcuIiYiHklLkZRFrF15nKlL8Hk5PVEDhkWKf_ZBIHl6zIX4fXSzUj-
    NkRwp8K7sxYvqCzQILvBaekdrt4i32HI-VeCfNSDd1qLIVU9BZh5fBUKdgtUHz7tYJAZPwQ0Qv-
    54GUCx6SxirsiqTIUxCAFVcdrsyfYwRzuZ-
    g6aBEdmktnD5G0JfbKW2m0tEeZt7rhN0dBghfBIQURlnRfWMuNz5Trk2hG6qrEkdRX_P033dIycuUOp
    gtcrL4Tzh3Bilg9NIdWf-h3QC0B4UcmrLTZtFA94cZY3CF18VoMwltspbcrqYE7mJd7OA"
}
```

Рисунок 23 - Приклад успішної відповіді з ***/api/login_check***

Якщо ж відправлені дані не є дійсними то буде повернено 401 HTTP статус з відповідним текстом, як це зображено на рисунку 24

*Рисунок 24 - Приклад відповіді з **/api/login_check** з недійсними даними*

Отже, авторизація користувача проходить через відправку даних користувача у форматі на url ***/api/login_check*** , у разі успішної авторизації користувач отримує спеціальний токен який може використовуватись у подальшому для відправки запитів.

3.3.2 Приклад роботи API задач

В даному розділі буде розглянуто роботу API на приладі задач, а саме запити на:

1. Створення задачі
2. Редагування задачі
3. Зчитування всіх задач
4. Зчитування конкретної задачі
5. Видалення задачі

1. Створення задачі

На рисунку 25 зображено приклад запиту на створення задачі який супроводжується токеном авторизації та тілом запиту з деталями задачі, а саме:

- **title** – заголовок задачі
- **text** – опис задачі
- **priority_id** – пріорітет задачі
- **column_id** – колонка задачі
- **swimlane_id** – дошка задачі
- **executor_id** – призначений виконавець задачі

Такі поля як id, owner_id, date_created є підставляються автоматично тому їх не потрібно відсилати власноруч.

```
POST taskmanager-backend.com/api/task/?token=eyJ0eXAiOiJKV1Qi…
{
    "title": "Дизайн нового сайту",
    "text": "Потрібно створити макет дизайну нового сайту",
    "priority_id": 1,
    "column_id": 3,
    "swimlane_id": 1,
    "executor_id": 4
}
```

*Рисунок 25 - Приклад запиту на **POST /api/task/***

На рисунку 26 зображено приклад успішного відповіді на запит, в полі **data** знаходиться інформація про нову задачу, у тому числі її ідентифікатор (поле id) який може бути використаний у подальшому для редагування задачі або її видалення.

Status: 201 **Created**


```
{
    "status": 201,
    "data": {
        "id": 9,
        "title": "Дизайн нового сайту",
        "text": "Потрібно створити макет дизайну нового сайту",
        "date_created": "2021-05-10 14:07:22",
        "priority_id": 1,
        "column_id": 3,
        "swimlane_id": 1,
        "owner_id": 1,
        "executor_id": 4,
    },
    "message": "created"
}
```


*Рисунок 26 - Приклад успішної відповіді з **POST /api/task/***

На рисунку 27 зображено приклад відповіді на запит з не коректними даними (пусто поле title), у полі **message** можна побачити конкретні деталі помилки.

Status: 400 **Bad Request**


```
{
    "status": 400,
    "data": "",
    "message": "Text should not be blank"
}
```


*Рисунок 27 - Приклад відповіді на невдалий запит **POST /api/task/***

2. Редагування задачі

Запит API на редагування задачі по своїй суті є дуже схожим на запит на створення нової задачі, відмінність тільки у тому що url запиту потрібно додати конкретний ідентифікатор задачі і метод HTTP запиту змінюється з POST на PUT, на рисунку 28 зображено приклад запиту на редагування

задачі з ідентифікатором 9. Відповідь на коректний та некоректний запити є такими же як і при створенні нової задачі.

PUT taskmanager-backend.com/api/task/9?token=eyJ0eXAiOiJKV1Qi…

```
{
    "title": "Дизайн нового сайту",
    "text": "Потрібно створити макет дизайну нового сайту",
    "priority_id": 1,
    "column_id": 2,
    "swimlane_id": 1,
    "executor_id": 2
}
```

*Рисунок 28 - Приклад запиту на **PUT /api/task/9***

3. Зчитування всіх задач

Запит на зчитування всіх задач у системі відправляється за допомогою HTTP-методу **GET** на url **/api/task/**.

На рисунку 29 зображено відповідь на запит, що зчитує та повертає всі задачі у системі, для даного прикладу він навмисно є зменшеним і відображає лише можливу структуру відповіді.

GET taskmanager-backend.com/api/task/?token=eyJ0eXAiOiJKV1QiLCJhbGci...

```
{
    "status": 200,
    "data": [
        {
            "id": 9,
            "title": "Дизайн нового сайту",
            "text": "Потрібно створити макет дизайну нового сайту",
            "date_created": "2021-04-22 18:07:22",
            "priority_id": 1,
            "column_id": 1,
            "swimlane_id": 1,
            "owner_id": 1,
            "executor_id": 2,
        },
        {…},
    ],
    "message": "success"
}
```

*Рисунок 29 - Приклад відповіді на запит **GET /api/task/***

4. Зчитування конкретної задачі

Запит на зчитування конкретної задачі здійснюється за допомогою ідентифікатора задачі і відправляється за допомогою HTTP-методу **GET** на url **/api/task/[id]**, де замість [id] йде індентифікатор задачі.

На рисунку 30 зображено відповідь на запит, що зчитує та повертає дані про конкретну задачу у системі.

GET taskmanager-backend.com/api/task/9?token=eyJ0eXAiOiJKV1QiLC…


```
{
    "status": 200,
    "data": {
        "id": 9,
        "title": "Дизайн нового сайту",
        "text": "Потрібно створити макет дизайну нового сайту",
        "date_created": "2021-05-10 14:07:22",
        "priority_id": 1,
        "column_id": 2,
        "swimlane_id":1,
        "owner_id": 1,
        "executor_id": 2,
    },
    "message": ""
}
```


*Рисунок 30 - Приклад відповіді на запит **GET /api/task/9***

5. Видалення задачі

Запит на видалення задачі також здійснюється за допомогою ідентифікатора задачі і відправляється за допомогою HTTP-методу **DELETE** на url **/api/task/[id]**, де замість [id] йде індентифікатор задачі.

На рисунку 31 зображено відповідь на запит, що видаляє конкретну задачу у системі, після цього запиту задача буде остаточно видалена із системи.

DELETE taskmanager-backend.com/api/task/9?token=eyJ0eXAiOiJKV1QiLCJhbGciOi…


```
{
    "status": 200,
    "data": [],
    "message": "removed"
}
```


*Рисунок 31 - Приклад відповіді на запит **DELETE /api/task/9***

Отже, у даному розділі було розглянуто роботу API на прикладі задач, процес їх створення, редагування, зчитування та видалення. За цим же принципом працюють і інші сутності при необхідності їх можна детальніше розглянути в Postman-колекції (Додаток Д).

3.4 Огляд кінцевого продукту

У даному розділі буде розглянуто користувацький інтерфейс розробленого програмного продукту, з можливими діями для різних типів користувачів.

Ролі користувачів системи та їх можливості

Так як у системі можуть бути користувачі з різними ролями слід зазначити ці ролі, а також їх можливості:

- Користувач
  - Може авторизуватись
  - Може переглядати задачі
  - Може змінювати статус задачі
  - Може залишати коментарі до задач
- Менеджер
  - Має всі можливості ролі Користувач
  - Може створювати, редагувати та видаляти задачі
- Адміністратор
  - Має всі можливості ролі Менеджер
  - Може створювати, редагувати та видаляти користувачів у системі

Вікно авторизації

Вікно авторизації (Рисунок 32) використовується для входу в систему, користувачу потрібно ввести свій логін та пароль, які їм має видавати адміністратор системи, тому що тільки у нього є права на створення нових користувачів.

У разі успішної авторизації користувач, незалежно від його ролі, потрапляє на сторінку канбан-дошки.

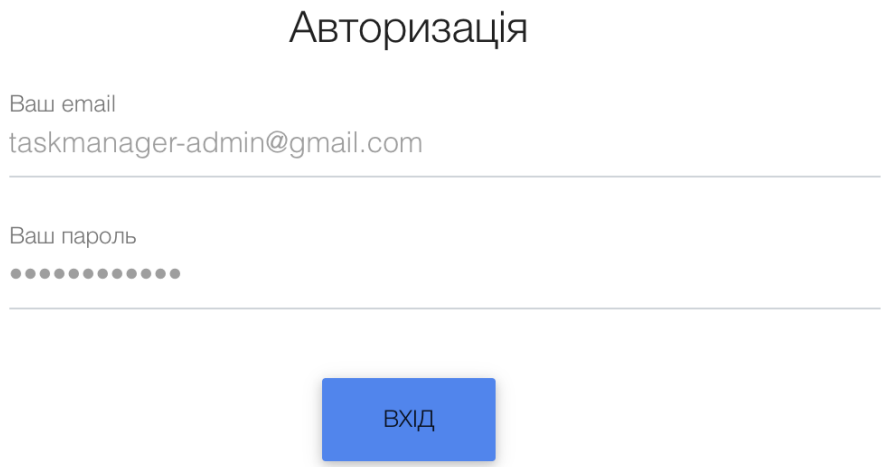

Рисунок 32 - Вікно авторизації

Панель керування задачами, або канбан-дошка

На рисунку 33 зображена канбан-дошка при авторизації користувача який має роль адміністратора.

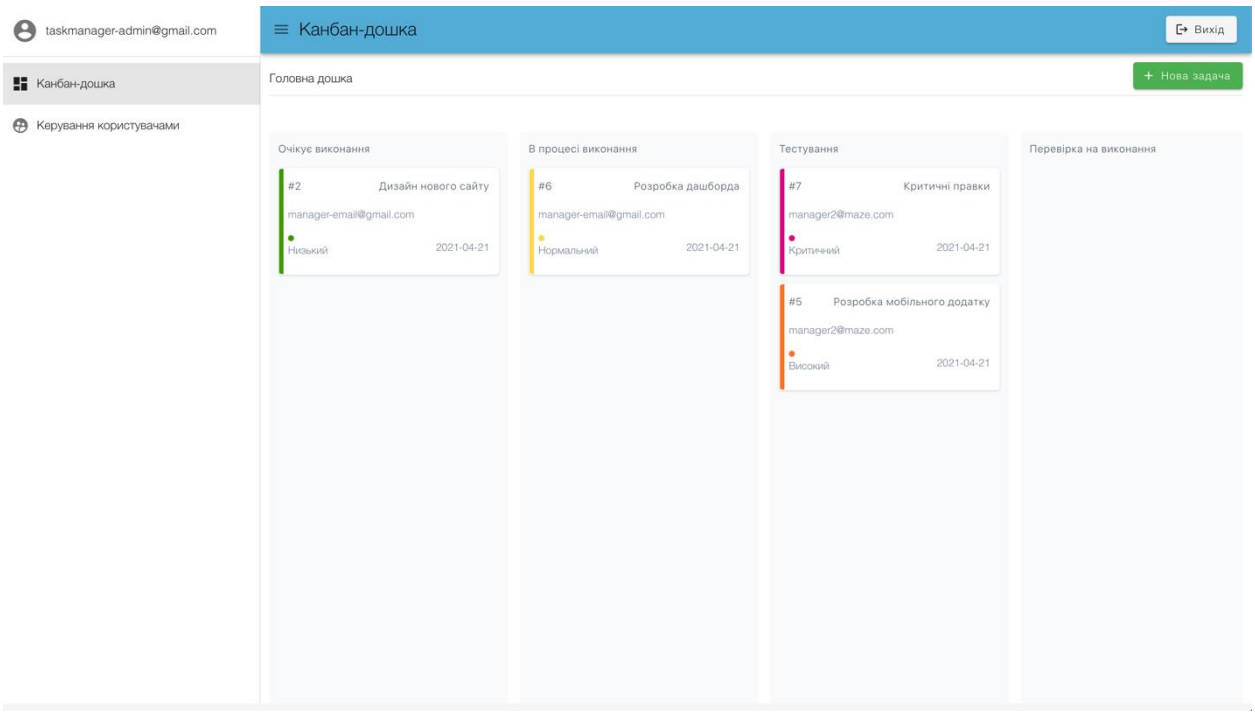

Рисунок 33 - Канбан-дошка

Основну частину сторінки складає власне сама канбан-дошка з її задачами. Канбан-дошка розділена на стовбці які відображають статус задачі,

У разі потреби кількість і назву стовбців можна змінити, в даному прикладі їх всього п'ять:

- Очікує виконання
- В процесі виконання
- Тестування
- Перевірка на виконання
- Готово (видно якщо проскролити далі)

Всередині стовбців знаходяться задачі які представлені у виді карток які, у свою чергу, можна перетягувати між колонками за допомогою лівої клавіші миші тим самим змінюючи їх статус. Кожна картка с задачею на даній дошці має наступні властивості:

- ID задачі
- Назва
- Приорітет, який виділений відповідним кольором
- Виконавець задачі
- Дата створення

У лівій частині знаходиться панель навігації (її можна сховати клікнувши на відповідну іконку) на якій є дві можливі сторінки:

- Канбан-дошка - на якій може знаходитися користувач який має будь-яку роль.
- Керування користувачами – ця сторінка доступна лише користувачу з привілегіями адміністратора системи.

У правій верхній частині екрану є дві кнопки – кнопка «Вихід» відповідає за вихід користувача з системи, а також кнопка «Нова задача» відповідно для створення нової задачі.

Вікно створення нової задачі

Вікно яке зображене на рисунку 34 може побачити лише користувач з роллю менеджера або адміністратора якщо клікнути по відповідній кнопці на

сторінки з канбан-дошкою. Тут можна створити нову задачу за назвою та текстом, визначити її пріорітет, колонку та виконавця.

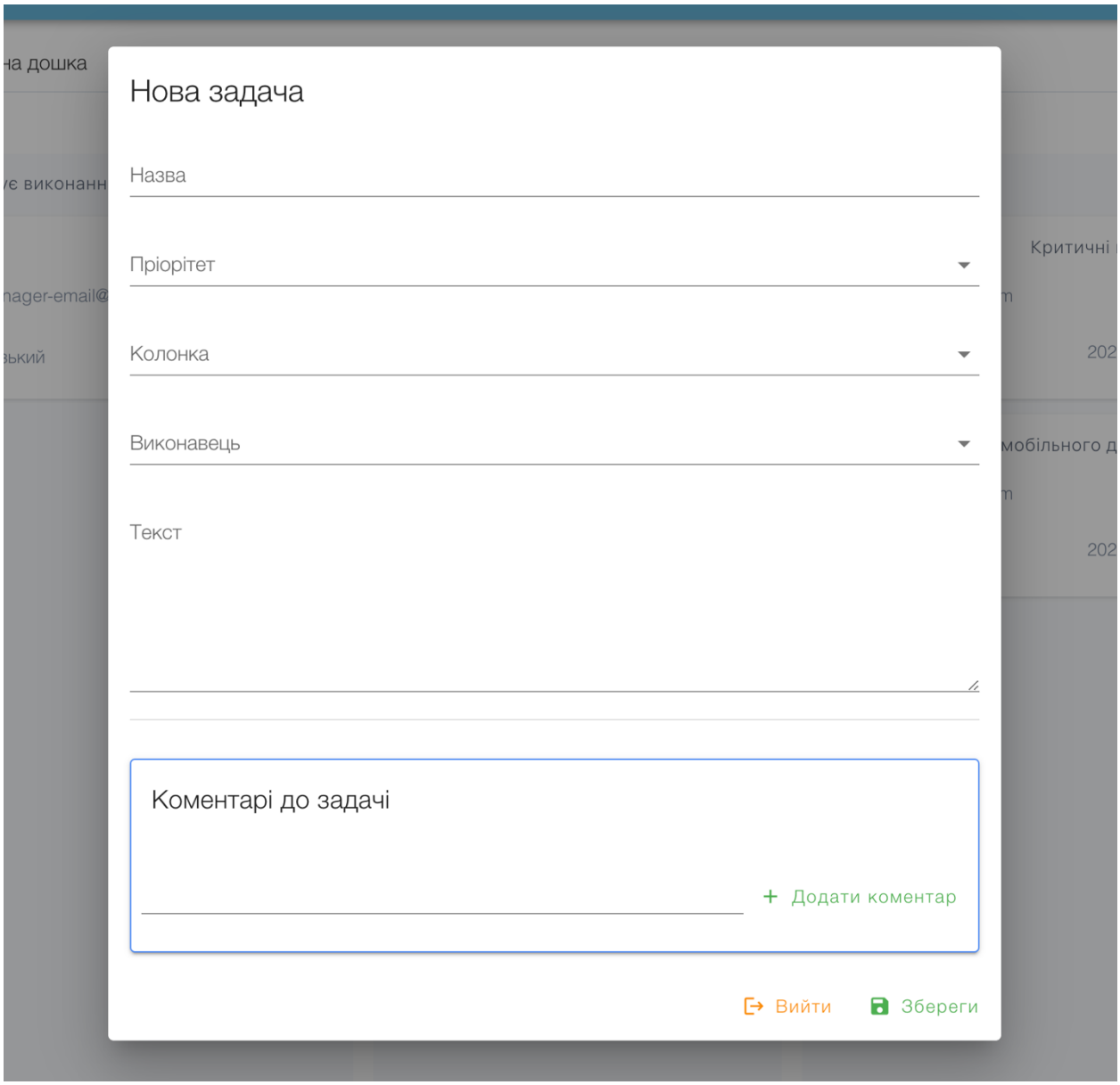

Рисунок 34 - Вікно створення нової задачі

Вікно редагування задачі

Вікно яке зображене на рисунку 35 відкривається при подвійному кліку лівою клавішею миші маючи роль менеджера, або адміністратора. Тут можна внести зміни до існуючої задачі, переглянути її опис, залишити коментар або видалити її.

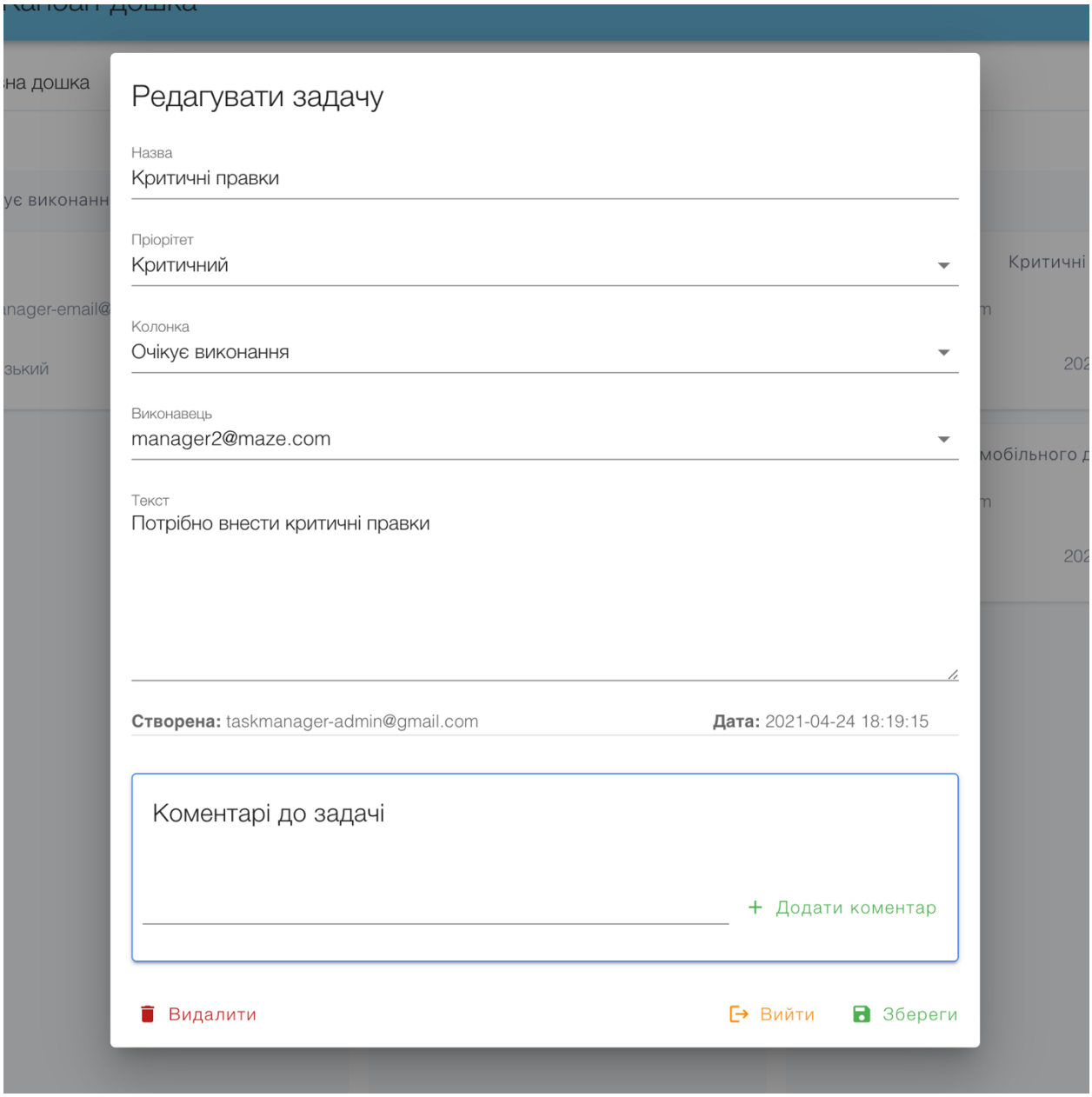

Рисунок 35 - Вікно редагування задачі

Вікно опису задачі

Вікно яке зображене на рисунку 36 відкривається при подвійному кліку лівою клавішею миші маючи роль користувача. Навідміну від менеджера який може редагувати задачу користувач може лише переглянути опис задачі та, при необхідності, залишити коментар.

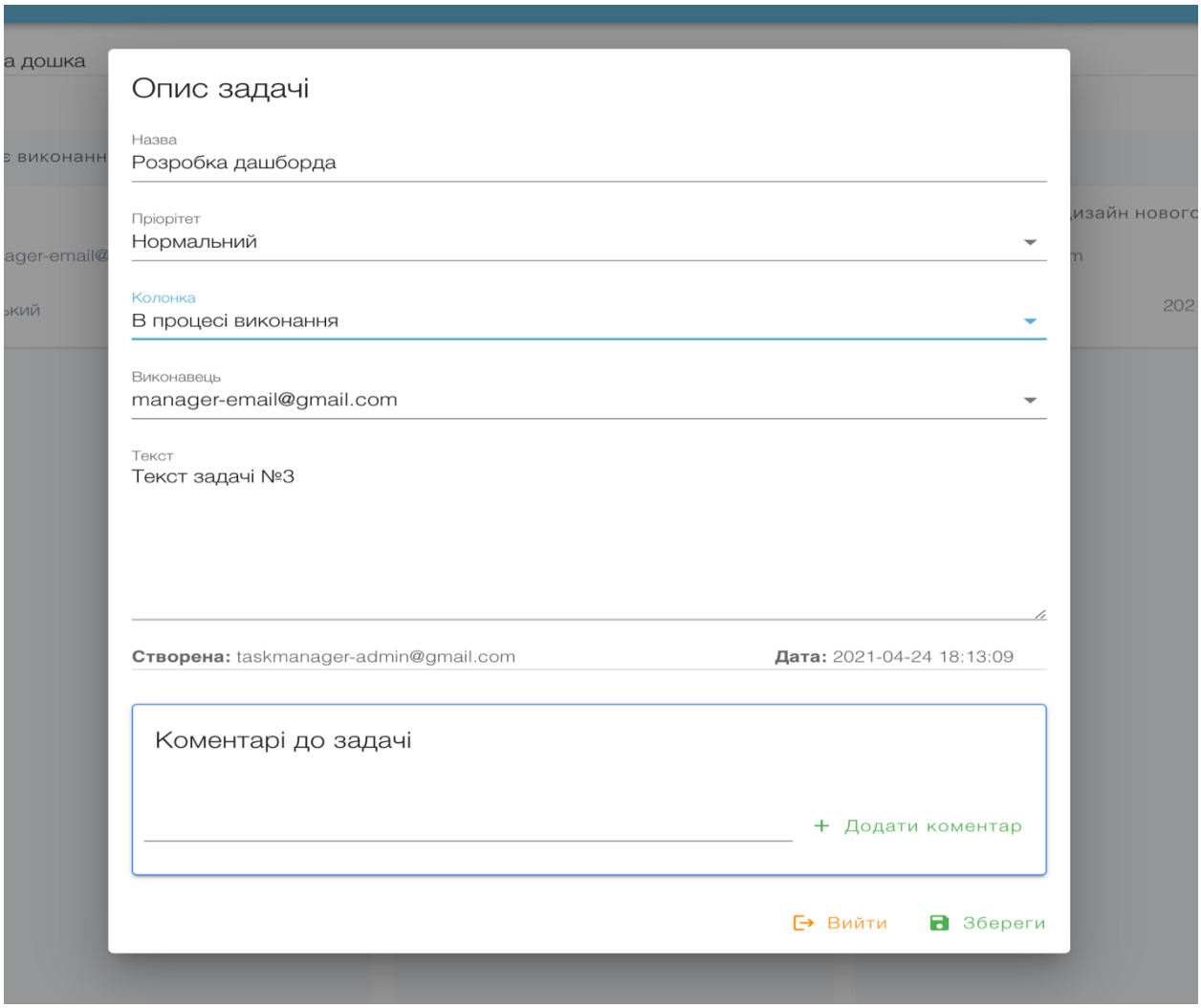

Рисунок 36 - Вікно опису задачі

Панель керування користувачами

Користувач з роллю адміністратора може перейти на сторінку керування користувачами (Рисунок 37). Тут можна додавати нових користувачів у систему, а також їх редагувати та видаляти.

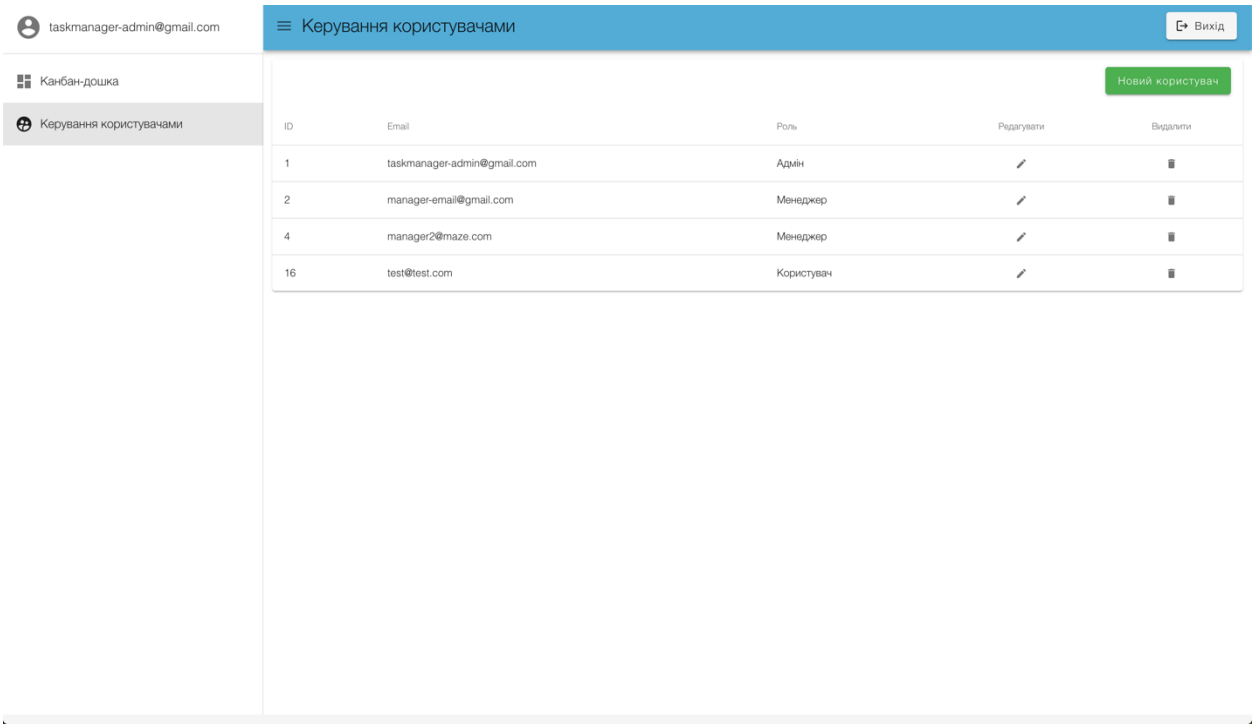

Рисунок 37 - Панель керування користувачами

Вікно створення нового користувача

На рисунку 38 зображена вікно створення нового користувача адміністратором, тут можна вказати email, пароль, а також обрати роль нового користувача. Вікно редагування користувача є аналогічним до вікна створення нового користувача.

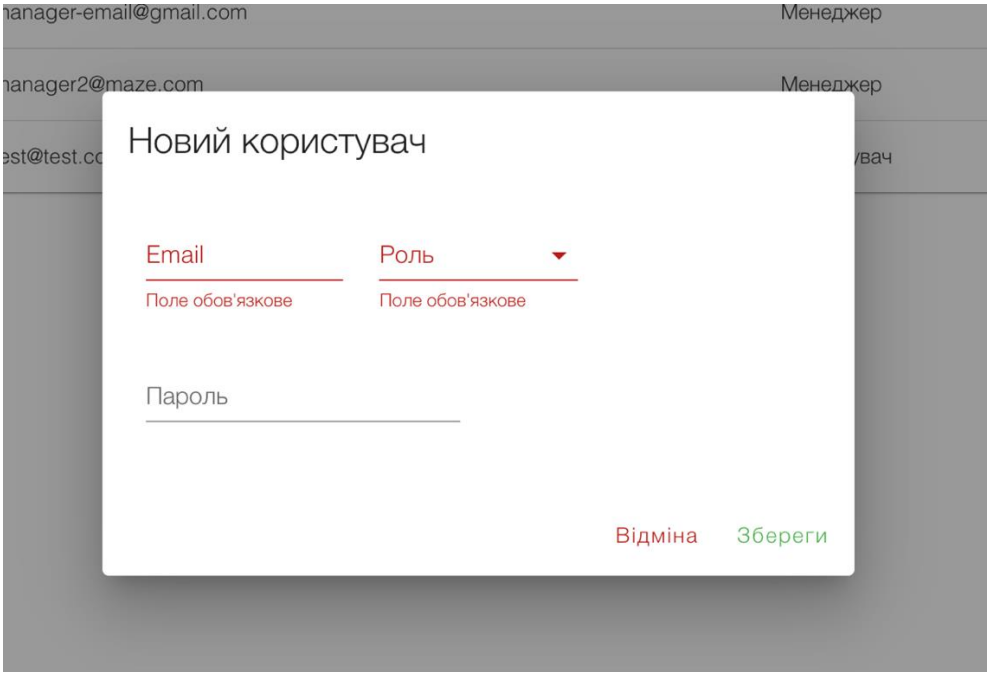

Рисунок 38 - Панель створення нового користувачами

Отже, у даному розділі було розглянуто користувацький інтерфейс розробленого програмного продукту, який має мінімальний набір всіх необхідних характеристик канбан-дошки, а також було представлено систему ролей з можливими діями для користувачів з різними ролями.

ВИСНОВКИ

Сучасний світ неможливо уявити без agile-медологій - компанії, що використовують їх у своїй роботі, швидше досягають результатів, ніж ті, хто використовує класичні процеси. Клієнти більш задоволені результатами роботи гнучких команд. Члени цих команд отримують більше задоволення від своєї роботи.

Засновані на гнучких методологіях процеси вирішують низку проблем, але в першу чергу це - постійна адаптація до мінливих чинників.

На сьогодні гнучкі медології показали свою ефективність в розробці програмного забезпечення, тому все частіше зустрічаються у якості основних процесів компаній працюючих у сфері інформаційних технологій.

Хоча коріння гнучкого підходу тягнуться углиб століть, але все ж таки буде логічніше вважати стартовою точкою розвитку agile-методології XX століття.

У agile-методологій не має конкретного засновника, тому що кожен вніс свой внесок в розробку концепції agile підходів - спочатку це були фізик і статистик Уолтер Шухарт та його учень Вільям Демінг створили першу ітерацію концепції яка з часом була названа як Цикл Демінга.

Пізніше Хіротака Такеучі спільно зі своїм партнером Ікуджіро Нонакою опублікували в журналі «Harvard Business Review» статтю в якій автори виділили коцепцію командно-орієнтованого підхіду розробки продукту. Нарешті році Джефф Сазерленд разом зі своїм колегою Кеном Швабером змогли об’єднати ці концепції та формалізувати підхід давши йому назву «Scrum» і в 1995 році цей підхід був представлений всьому світу.

Методика Scrum за своєю суттю є евристичною - в її основі лежить постійне навчання та адаптація до мінливих чинників. Згідно Scrum, команда не знає всього на початку проекту, але буде розвиватися, отримуючи уроки з досвіду. Робочий процес передбачає зміну пріоритетів і короткі цикли релізу, що сприяє постійному навчанню та вдосконаленню команди.

Пізніше Девід Андерсон створює нову концепцію agile-підходу, якій дав назву Kanban і популярізує її завдяки своїй книзі «Kanban. Альтернативний шлях в Agile».

Основною різницею між Scrum і Kanban є довжина ітерацій (або спринтів). У Scrum середня тривалість спринту - 2 тижні, в Kanban завдання розробнику можна «підсовувати» хоч кожен день. При роботі за методологією Scrum зазвичай планують витрати часу на кожну задачу в спринті, чого не прийнято роботи при Kanban-підході, Kanban дає більше гнучкості, якщо під гнучкістю розуміти частоту зміни пріоритетів.

Отже, метою Scrum є завершення спринту, а метою Kanban є завершення задачі, на практиці ці два підходи часто об’єднують, беручи можливість планування Scrum-підходу при цьому зберігаючи гнучкість Kanban.

Для візуалізації agile-підходів використовують канбан-дошки: фізичні та електронні. Вони дозволяють зробити робочий процес відкритим і зрозумілим для всіх фахівців, що важливо, коли у команди немає одного формального керівника.

Також були розглянуті сучасні реалізації канбан-дошок к з їх перевагами та недоліками. Метою дослідження було створення канбан-дошки, яка відповідає основним характеристикам її існуючих аналогів.

При проведенні дослідження сучасних реалізацій канбан-дошок можна зробити висновок, що більшість із них є веб-додатки заснованими на клієнт-серверній архітектурі.

Далі було проведено дослідження сучасних веб-додатків та їх архітектури.

При побудові простих веб-додатків часто сервер може відповідати клієнту готовою веб-сторінкою, але при побудові веб-додатків зі складною бізнес-логікою все частіше можна побачити підхід при якому клієнт та

сервер обмінюються інформацією через прикладний програмний інтерфейс (API).

Завдяки клієнт-серверній архітектурі клієнтська частина часто буває мультиплатформенною, а точніше - може бути не тільки веб додатком, а ще й мобільним або комп’ютерним додатком, які, в свою чергу, можуть взаємодіяти з одним і тим же сервером по одному і тому же веб-API.

Веб-API - використовується в веб-розробці і містить, як правило, певний набір HTTP-запитів, а також визначення структури HTTP-відповідей, для вираження яких використовують XML- або JSON-формат.

Найвідомішими видами веб-API:

- Віддалений виклик процедур (Remote Procedure Call - RPC)
- Простий протокол доступу до об'єктів (Simple Object Access Protocol - SOAP)
- Передача репрезентативного стану (Representational State Transfer - REST)

При розробці kanban-дошки для даної дипломної роботи було розроблене власне API з використанням стандарту REST, за рахунок його популярності на сьогодні при написанні мультиплатформенних додатків, а також завдяки зручному та зрозумілому синтаксису.

Далі була розглянута розроблена канбан-дошка, а саме:

- Був наведений перелік використаного стеку технологій з їх коротким описом та поясненням задля чого вони були використані.
- Було розглянуто процес проектування бази даних з поясненням використаних типів зв’язків між таблицями, та використанням UML-діаграми спроектованої бази даних.
- Був проведений опис розробленого веб-API з поясненням процесу авторизації користувача, а також наведений приклад роботи розробленого веб-API для задач, з можливістю їх створення, зчитування, редагування та видалення.

- В останньому розділі був проведений огляд кінцевого продукту, а точніше - користувацький інтерфейс розробленого програмного продукту, з можливими діями для різних типів користувачів.

Отже, на основі результатів виконаних досліджень розроблено канбан-дошку у вигляді веб-додатку для постановки та керуванню задачами з розробки програмного забезпечення, яка відповідає основним характеристикам її існуючих аналогів, а саме:

- Створення та постановка нових задач.
- Можливість змінювання опису задач а також їх видалення.
- Створення нових користувачів з різними ролями та можливість їх авторизації.
- Можливість змінювати та видаляти існуючих користувачів у системі, а також можливість надавати їм спеціальні права.
- Можливість змінювати статус задач, пересуваючи їх по kanban-дошці.
- Можливість вказувати пріорітет задач.
- Можливість користувачів бачити коментарі до задачі а також створювати нові.

ПЕРЕЛІК ПОСИЛАНЬ

1. https://hbr.org/2016/04/the-secret-history-of-agile-innovation# - The Secret History of Agile Innovation by Darrell K. Rigby, Jeff Sutherland, and Hirotaka Takeuchi
2. CLAIRE DRUMOND conference
3. Скрам. Революційний метод управління проектами. Джефф Сазерленд. (с. 74-76)
4. https://uk.wikipedia.org/wiki/DevOps
5. https://worksection.com/blog/kanban.html
6. Дэвид Андерсон. Kanban (с. 38-39)
7. https://www.atlassian.com/agile/kanban/boards
8. https://bakunin.com/services/youtrack/
9. https://en.wikipedia.org/wiki/Web_server
10. https://mkdev.me/posts/chto-takoe-api-v-veb-prilozheniyah-i-zachem-on-nuzhen
11. https://nuancesprog.ru/p/11310/
12. https://www.php.net/manual/en/intro-whatis.php
13. https://developer.mozilla.org/ru/docs/Web/JavaScript
14. https://en.wikipedia.org/wiki/Single-page_application
15. https://en.wikipedia.org/wiki/Doctrine_(PHP)
16. https://uk.wikipedia.org/wiki/JSON_Web_Token

https://jiraved.ru/agile/scrum/sprints (рисунок 1)

https://www.atlassian.com/agile/kanban/boards (рисунок 2)

https://pyrus.com/ru/blog/kanban-v-pyrus (рисунок 3)

https://jiraved.ru/agile/kanban (рисунок 4)

https://www.quora.com/Are-there-any-good-project-management-tools-alternatives-to-Trello-etc (рисунок 5)

https://www.jetbrains.com/ru-ru/youtrack/features/agile_project_management.html (рисунок 6)

http://www.immsp.kiev.ua/publications/articles/2016/2016_4/04_2016_Kasim.pdf (рисунок 7)

СЛОВНИК СКОРОЧЕНЬ

Беклог - це документ, який має список вимог до функціональності, які упорядковані згідно зі ступенем важливості. Product backlog представляє список того, що повинно бути реалізовано.

Спринт - це регулярний, повторюваний робочий цикл методології scrum, під час якого робота завершена і готова до огляду.

Безперервна поставка (CD) - передбачає часту поставку релізів продукту клієнтам.

Безперервна інтеграція (CI) - це практика інкрементної автоматизації складання і тестування коду протягом дня.

ПЗ – програмне забезпечення.

Обмеження незавершеної роботи (WIP) - це максимальна кількість карток, що може перебувати в одному стовпці одночасно.

API – прикладний програмний інтерфейс.

Система управління бази даних (СУБД) - це інформаційна модель, що дозволяє упорядковано зберігати дані про об'єкт або групі об'єктів, що володіють набором властивостей, які можна категоризувати.

Doctrine (ORM) — об'єктно-реляційний проектор для PHP 5.3.0+, який базується на шарі абстракції доступу до БД (DBAL).

ДОДАТКИ