на тему: «Дослідження Web-додатків та їх застосування в хмарних сервісах для підвищення ефективності бізнесу»

КНД–41 Волковицький О.В.

1. АНАЛІЗ СПОСОБІВ ЗАСТОСУВАННЯ WEB-ДОДАТКІВ CRM СИСТЕМ В ХМАРНИХ СЕРВІСАХ

1.1 **Аналіз призначення та застосування CRM систем**

Проблема підвищення ефективності в адмініструванні бізнесу за рахунок впровадження WEB-додатків для інформаційного забезпечення організаційної структури підприємства є актуальною. Одним з важливих чинників підвищення ефективності підприємства є поліпшення: управління організацією, операційною діяльністю та процесами підтримки послуги завдяки використанню сучасних WEB-додатків, які розміщені в хмарах та застосовують хмарні технології.

Виконаний аналіз специфіки і характеристик бізнесу дозволяє виділити цілий ряд переваг за рахунок впровадження WEB-додатків в умовах конкуренції, що підвищують стійкість на внутрішньому ринку. Серед таких переваг можна визначити:

- гнучкість (масштабованість, можливості зберігання і контролю, багатий вибір інструментів, засоби захисту);
- ефективність (доступність, швидкість виведення на ринок, безпека даних, економія на обладнанні, краще структура платежів);
- стратегічну цінність (оптимізація процесу роботи, регулярне оновлення, співробітництво).
- Все це надає конкурентні переваги. Але слід зазначити і недоліки, які характерні для WEB-додатків:
- для роботи з «хмарою» потрібне постійне підключення до інтернету;

- користувач не завжди може налаштувати використовується програмне забезпечення під індивідуальні потреби;
- щоб створити власну «хмара» будуть потрібні дуже великі витрати, що недоцільно для нових підприємств;
- «хмара» - сховище даних, до яких, використовуючи вразливості системи, можуть отримати доступ зловмисники.

Одним з важливих чинників підвищення ефективності підприємств бізнесу є поліпшення його управління, що забезпечується завдяки використанню WEB-додатків, які створені на основі штучного інтелекту. Показано, що застосування інтелектуальних WEB-додатків в сфері управління бізнесом націлене на аналіз його бізнес-процесів і управління ними, прогнозування та інформаційно-аналітичну підтримку прийняття управлінських рішень. Тому новим напрямком є застосування інтелектуальних WEB-додатків, саме інтелектуальних WEB- сервісів.

1. Підключення CPM системи. 50 відсодків власників бізнесу визнають, що організаційна роздрібленість негативно впливає на якість обслуговування клієнтів та потенційних клієнтів. Розроблені інформаційні джерела – велика проблема але загальна платформа для управління взаємовідносинами з клієнтами може допомогти. Фоктично 80 відсотків бізнес-лідерів кажуть зо все частіше використовують CPM системи для своєї компанії як єдине джерело правди про своїх клієнтів у різних віділах. За допомогою агальної CPM системи працівники компаній отримують необхідні інструменти та дані для більш ефективного управління відносинами з клієнтами.

2 Внести покращення до прибутку. Було показано що CPM платформи дають реальні результати, та пряме покращення чистого прибутку.

3.Виділити та класифікувати потенційних клієнтів. CPM система може допомогти легко та швидко виявляти та додавати потенційних клієнтів та класифікувати їх. Засоредившись на правильних лідах продажі можуть виділити пріоритетність, що приведуть до заключенню сділок, а маркетинг може ви-

явити потенційних клієнтів, котрим потрібена велика увага, та пробити їх стати якісними.

4. Підвищення життєвої цінності клієнту. Завдяки кращому розумінню своїх клієнтів додаткові продажі стають очевидними, що дає шанс виграти новий бізнес у нових клієнтів. Це допомагає налагодити міцні зв'язки з клієнтами. Завдяки кращій видимості можливо надавати клієнтам якісніші та кращі послуги.

5. Краща підтримка клієнтів. Сучасні клієнти прагнуть швидкої персоналізованої допомоги у любий час.

1.2 Вразливість CPM систем та загрози при їх впровадженні

Загроза - це потенційні или реальні дії, які спричиняють порушення буду функціонування системи правління. Загроза може бути передумовою виникнення порушення усіх или 1-го буква аспектів безпеки.

Уразливий об'єкт – це слабка корова около системі, эна никак не здатна протистояти впливам

Людська уразливість виникає внаслідок психологічних впливів. Технічна — итог виникнення несправності во механізмах управління порядком. Інформаційна уразливість є наслідком непередбачуваного впливу інформації в процес прийняття рішень. Загроза і уразливий об'єкт — це передумови виникнення кризисного буду інфраструктури.

Уразливість в інформаційній системі є подією, с-из-за якої компрометується один либо кілька аспектів безпеки інформації (доступність, конфіденційність, цілісність і достовірність). Наявність уразливого об'єкта створює слабку ланку, що може призвести вплотную вплоть до порушення безпеки інформації. Безусловно, що вплив загрози во технологію управління може викликати критичний отделение слепыш її складових, подобным способом і всієї системи.Уразливість во інформаційній системі є подією, из-за якої компрометується единственный или кілька аспектів безпеки інформації (доступність, конфіден-

ційність, цілісність і достовірність). Наявність уразливого об’єкта створює слабку ланку, що може призвести вплоть до порушення безпеки інформації. Естественно, що вплив загрози в технологію управління може викликати критичний подразделение млекопитающее її складових, таким образом і всієї системи.

Основні загрози:

1) руйнування (або спотворення) даних, що несе из-за собой збій во діяльності організації буква усіма наслідками, що випливають збитками;

2) розкрадання конфіденційних даних організації, наслідки яких очевидні;

3) керогаз сервера до скоєння операцій во обхід положень діяльності організації.

Звідси стає зрозумілим, що, чим глибше CRM система інтегрована в діяльність організації, тим вище вимоги до її надійності. Слабка інтеграція CRM системи, з одного боку, зводить можливий збиток від помилок до мінімуму, але, з іншого боку, не дає ніякої корисної віддачі. На даний момент велика кількість IT-компаній пропонує розробку CRM системи «під ключ». При виборі такої CRM системи інтеграція протікає без відриву співробітників від виробництва і без особливих складнощів. Але, на жаль, ніяка невелика IT-компанія не в змозі створити справжню CRM систему. Справа в тому, що складність і багато компонетність CRM систем вимагає залучення для розробки фахівців із різних галузей.

Тоді можна визначити низку уразливостей під час впровадження CRM системи.

1. Уразливостей IT компанії під время вибору шляху впровадження CRM системи. Утримувати апарат управління с целью IT компанії можна тільки из-за рахунок масового реализацию, інакше бізнес стане нерентабельним ще вплоть до этого времени, млекопитающее CRM концепция если расположена хоча буква отчасти. Причому ця пятьдесят процентов если розроблятися во процесі інтеграції CRM во діяльність організації і всі помилки розробників стануть прямими збитками с целью організації. Естественно, що організація никак не если зацікавлена около фінансуванні налагодження CRM системи из-за свій рахунок

(спочатку никак не подключений во договір), що збільшує реальну вартість інтеграції CRM системи. Подобным чином, набагато вигідніше придбати CRM концепцию загального призначення, наприклад, від знаменитого виробника (естественно, никак не першої версії), і змиритися буква неминучими витратами адаптації бізнес-процесів організації під таку концепцию. Около любому разі, единица CRM концепция вже налагоджена.

2. Уразливостей медперсоналу під время підготовки вплоть до впровадження і во лично впровадженні. В даному етапі використання системи знаходиться під загрозою, этому що во етапі підготовки вплоть до впровадження: все без исключения непередбачуване і сомнительно. Розглянемо вже відомі типові загрозі, но саме: менеджери никак не розуміють, млекопитающее користуватися порядком, или нетерплячість керівництва. Дійсно, около медперсоналу викликають утруднення извини речі під время никак не підготовленості присутствие впровадженні CRM системи, наприклад, никак не відомо млекопитающее здійснювати введення даних, які витрати робочого времени в вбивання вже існуючої, використовуваної інформації эта інші. Нетерплячість керівництва проявляється около этому, що, якщо керівництво никак не бачити довгий время потрібних їм даних во системі, воно починає дивуватися відносно сенс впровадження. Звідси виникає досить просте принцип впровадження CRM системи – це необхідність навчання медперсоналу і керівництва.

3. Уразливостей під час підготовки (адаптації) стандартної програми, що реалізує CRM (бажання впхнути у концепцію все без виключення!). Цілком імовірно, що с целью підготовки (адаптації) стандартної програми, що реалізує CRM, вплоть до мозгов даної фірми будуть найняті фахівці буква боку (или попросту збільшений с целью цих цілей состав співробітників). Цим народам необхідно если во небывало короткі терміни зрозуміти целую боту організації і розробити бізнес-процеси, из-за якими во результаті во результаті впровадження і если здійснюватися бота всієї фірми. Наскільки вірно вони будуть розроблені, настільки спроститься бота співробітників підприємства і підвищиться її ефективність.

Крім этого зачастую замовники задають функції, які никак не потрібні взагалі

4. Уразливость плану CRM від его новизни (опір менеджерів). Відомо, що після впровадження CRM, механізм роботи зміниться - но нове люблять никак не всі, вплоть до нього потрібно звикнути. CRM-концепция починає надавати переваги після накопичення певного обсягу даних. Тоді менеджери во результаті успішного впровадження отримують автоматичний несесер, во якому будуть міститися: інформація относительно клієнтів, находись-які нагадування, пов'язані буква роботою (время дзвінка этому мера іншому покупцеві, важливі дати) - причому руководитель може стати деякою "напоміналкою», що і если робити.

5. Уразливость плану CRM від нечітко визначеної мети впровадження (нерозуміння всіх можливостей системи керівником). Около компанії, яка планує запустити CRM програму, немає чіткого бачення своїх цілей розвитку в найближчий время, около співробітників немає розуміння своїх обов'язків, бізнес-процеси хаотичні і зачасту змінюються. Один одним словом, якщо во компанії хаос, в такому випадку присутність спробі його автоматизувати вийде автоматизований хаос.

1.3 Аналіз призначення та застосування хмарних сервісів

Хмарні сервіси – це інфраструктура, платформи або програмне забезпечення, котре розміщеється сторонніми поставниками та доступні користувачам мережі Інтернет.

Хмарні сервіси полегшують поток користувачів від зовнішніх клієнтів через Інтернет в системи провайдера та назад.

Типи хмарних сервісів. Вся інфраструктура, платформи, програмне забезпечення або технології отримують доступ через Інтернет при необхідності без загрузки додаткового програмного забезпечення.

Інфраструктура як послуга (IaaS) – дає користувачам обчислюванні, мережеві ресурси та ресурси для зберігання.

Платформа як послуга (PaaS) – надають користувачам платформу, на котрій можуть працювати додатки, а також усю IT-інфрастуктуру, що необхідна для праці.

Програмне забезпечення як послуга (SaaS) – надає користувачам хмарний додаток, платформу, ко котрій воно працює та базову інфраструктуру.

Функція як послуга (FaaS) – управляє подіями модель виконування, дозволяє розробникам створювати, запускати та виконувати пакети додатків як функціями без обслуговування та інфраструктури.

Хмарні сервіси надають виконувати хмарні обчислення, що є виконуванням робочих загрузок в хмарному середовищу. Частні хмари у загальному виділяються як хмарне середовище що пристосоване виключно для кінцевого користувача, звично у рамках брандмауера користувача, а іноді й локально.

Публічні хмари - це хмарні середовища, створені з ресурсів, які не належать кінцевому користувачеві, які можна перерозподілити серед інших клієнтів.

Гібридні хмари - це кілька хмарних середовищ з певним ступенем переносності робочих навантажень, оркестровки і управління серед них.

Мультихмари - це IT-системи, котрі включають в себе більше одного хмари - загальнодоступного або приватного, - котрі можуть бути об'єднані в мережу, а можуть і не об'єднані. Гібридні хмари можуть допомогти поліпшити хмарні сервіси

Як працюють хмарні сервіси? Як і всі інші IT-рішення, хмарні сервіси залежать від устаткування і програмного забезпечення. Однак, на відміну від традиційних апаратних і програмних рішень, користувачам не потрібно нічого, крім комп'ютера, підключення до мережі та операційної системи для доступу до хмарних сервісів.

Хмарна інфраструктура. Надаючи користувачам хмарну інфраструктуру, постачальники хмарних послуг відокремлюють обчислювальні можливості від апаратних компонентів, наприклад:

- Обчислювальна потужність центральних процесорів (ЦП)

- Активна пам'ять з мікросхем оперативної пам'яті (RAM)
- Обробка графіки від графічних процесорів (GPU)

Доступність сховища даних з центрів обробки даних або жорстких дисків. Ця абстракція зазвичай досягається за допомогою віртуалізації і віртуальних машин. Після поділу компоненти зберігання, обчислень і мережі надаються користувачам через Інтернет в якості інфраструктури або IaaS. Цей вид хмарних сервісів привів до появи хмарних сховищ, в яких великі дані зберігаються як частина Інтернету речей (IOT). RackSpace - це приклад провайдера IaaS.

Хмарні платформи. Постачальники хмарних послуг також можуть використовувати свої апаратні ресурси для створення хмарних платформ, які представляють собою онлайн-середовища, в яких користувачі можуть розробляти код або запускати додатки. Створення хмарної платформи вимагає більшого, ніж просто абстрагування можливостей комп'ютера від його апаратних компонентів - наприклад, при наданні хмарної інфраструктури. Для надання хмарної платформи потрібні додаткові рівні розробки для включення таких технологій, як контейнеризація, оркестровка, інтерфейси прикладного програмування (API), маршрутизація, безпека, управління і автоматизація. Дизайн користувальницького інтерфейсу (UX) також є важливим фактором для створення зручного для навігації в Інтернеті.

Хмарні платформи - це різновид PaaS. І якщо інфраструктурні компоненти, що підтримують PaaS, добре масштабуються і доступні для спільного використання, це можна розглядати як хмара. Кращі приклади хмар PaaS включають загальнодоступні хмари і керовані приватні хмари.

Постачальники загальнодоступної хмари. Постачальники загальнодоступної хмари абстрагують свою власну інфраструктуру, платформи або додатки від обладнання, яким вони володіють; об'єднати їх в озера даних; і поділіться ними з багатьма орендарями. Вони також можуть пропонувати загальнодоступні хмарні сервіси, такі як управління API, хмарні операційні системи або бібліотеки шаблонів розробки, відомі як фреймворки.

Керовані приватні хмари. Також відомі як постачальники керованих хмар, постачальники приватного хмари обслуговують клієнтів у приватному хмарі, яке розгортається, налаштовується і управляється кимось, крім клієнта. Це варіант хмарної доставки, який допомагає підприємствам або малим підприємствам з недоукомплектованими або недостатньо кваліфікованими IT-командами надавати користувачам більш якісні послуги та інфраструктуру приватного хмари.

Хмарне програмне забезпечення. Останній широко поширений хмарний сервіс, який можуть запропонувати постачальники, - це повноцінне веб-додаток, відоме як хмарне програмне забезпечення або SaaS. Це вимагає великих вкладень в розробку, тому що хмарний провайдер буквально пропонує клієнтам онлайн-додаток. Хмарне програмне забезпечення може бути надано з використанням хмарного підходу, який представляє собою архітектуру програми, яка поєднуватиме невеликі, незалежні і слабкозв'язаного мікросервіси.

2 ДОСЛІДЖЕННЯ ФРЕЙМВОРКІВ ДЛЯ ВЕБ-РОЗРОБКИ CRM СИСТЕМ

2.1 Призначення фрейворк як інструмента для програміста

Сутність фреймворка полягає у тому, що він є основою програм, це каркас програмного середовища спеціального призначення, який використовують для того щоб полегшити об’єднання певних компонентів при створенні програмного забезпечення. Фреймворк об’єднує усі програмні компоненти що використовуються та дозволяє добавити компоненти до програми, залежно від потреб.

Класифікації фреймворків:

- Фреймворки додатків;
- Фреймворки програмних моделей;
- Фреймворки концептуальних моделей.

Використання фреймворків. Якщо стоїть задача створити сайт, то необхідно знайти стратегію роботи. Загалом є три шляхи розробки:

1. Написати вихідний код з нуля. Головною перевагою цього варіанту є його варіативність – практично немає ніяких обмежень, можливо реалізовувати будь-який задуманий функціонал, для цього потрібні лише вміння. Головним мінусом являється праце місткість процесу, його людино-години. Також треба буде прикласти багато зусиль для ретельного тестування отриманого продукту – потрібно буде знайти всі його недоліки та виправляти їх, щоб створити ідеальний веб-проект.

2. Використання готової Content Management System (CMS) – це система управління вмістом - інформаційна система або комп'ютерна програма, яка використовується для забезпечення і організації спільного процесу створення, редагування і управління вмістом, інакше - контентом. Цей варіант є ідеальним

для тих людей, які мало розуміють програмування у сфері веб-розробки. З готовою CMS можливо оперативне створити сайт, який відповідає потребам. Також є можливість вносити необхідні корективи за допомогою панелі адміністратора. Але даний підхід має мінуси. Головний мінус це дуже велика кількість обмежень, саме через це, цей підхід не використовують у великих масштабах.

3. Використання фреймворків, тобто існує основа, в яку потрібно додати потрібну кількість необхідних компонентів. Цей варіант є рентабельним для тих хто розбирається у програмуванні – без відповідного навику програмування виповнити поставлену задачу правильно неможливо.

Виходячи з порівняних цих шляхи розробки можна зрозуміти, що фреймворк являється «золотою серединою» між написанням свого коду, та використанням обмеженою по функціоналу CMS. Використовуючи фреймворк, отримується готовий каркас для проекту, при цьому не втрачаючи гнучкості у плані функціоналу.

Веб-фреймворк —це платформа для створення сайтів та веб додатків які полегшуються розробку та об’єднання компонентів великого прокту. За рахуйнок цих широких можливостей у реалізації бізен логіки та високої продуктивності ця платформа особливо добре знаходить своє місце для стоврення складниз сайтів, та веб-сервісів.

Сучасні веб-фреймворки – це середовища програм, котрі спрощуються створення, підтримку або масштабування додатків. Вони представляють з себе інструменти та бібліотеки, які спрощують загальні задачі веб розробки, включаючи маршрутизацію URL адрес для відповідних обробників, взаємодія з базами даних, підтримку сеансів та авторизацію користувачів, форматування виводів (HTML, JSON, XML) та постійне покращення захисту від несанкціонованих дій.

Основні перваги фреймворків:

1. *Економічна ефективність та цілесообразність використання фреймвоків*. З точки зору бізнесу, розробка на фреймворках завжди економічно вигідна та якісніша по результату, ніж написання проекту на чистій мові програму-

вання без використання платформ. Розробка без використання платформ буде правильним рішенням тільки в двох випадках – або якщо проект зовсім простий та не потребує дальшого розвитку або якщо він згружений та потребує дуже низько рівневої оптимізації. В усіх інших випадках розробка на платформі швидша та якісніша.

Якщо зрівнювати фреймворки з іншими класами платформ - SaaS, CMS або CMF то фреймворки замітко ефективніші у використанні у проектах з складною бізнес логікою та високими вимогами що до швидкості роботи, безпеки та надійності. Але у простих та типових проектах без значних вимог швидкість та вартість розробки на фреймворках буде значно краща ніж на SaaS або CMS.

2. Технічні переваги фреймворків. Одною з головних переваг у використанні фреймворків є те, що фреймворк сам виділяє уніфіковану структуру для побудованих на його основі додатків. Тому додатки на фреймворках значно легше супроводжувати та дороблювати, так як стандартизована структура організацій компонентів знайома усім розробникам на цій платформі, та не потребує довгого розбирання у його архітектурі, для того щоб зрозуміти принцип роботи додатку або знайти місце реалізації того чи іншого функціоналу.

Велика кількість фреймворків для розробки додатків використовує парадигму MVC – це означає що в багатьох фреймворках підхід є ідентичним до організації компонентів додатків та це ще більше спрощує розуміння архітектури на незнайомому розробнику фреймворці.

Для забезпечення більших можливостей у більшій кількості фреймворків використовуються техніки об’єкто-орієнтованого програмування. Наприклад: частини додків можуть унаслідуватись від базових класів фреймворків або інші модулі мужеть бути підключеними як додаток.

Екосистеми веб фреймворків також дуже багаті на готові реалізації багатьох функціональних можливостей. Розробникам при праці над типовими задачами не потрібно придумувати щось своє, того що вони можуть використовувати вже створену суспільством реалізацією. Це не тільки скорочує затрати часу, а ще й дозволяє домогтися більшої стабільності рішення – компонент що вико-

ристовується то дорабляється великою кількістю інших розробників завжди більш якісно реалізований так краще протестований на більшій кількості сценаріїв, ніж рішення яке може у адекватні строки розробляти один розробник або навіть невелика команда розробників.

3. Порівняння з альтернативними. Перед веб розробниками часто виникає питання вибору між фреймворками для реалізації проекту або використання CMS. У кожного з підходів є свої переваги та недоліки.

Переваги фреймворків:

1. Розробка на фреймвоці дозволяє просто супроводжувати проекти.

2. Можлива масштабування та модернізація.

3. Рішення на фреймворках значно швидше та витримують велику загрузку.

4. По рівню безпеки рішення на фреймворках значно кращі ніж самописні системи та можуть порівнюватись з CMS.

Недоліки фреймворків:

1. Строки розробки на фреймворках більше, ніж при використовувані CMS. Фреймворки містять у собі тільки базові компоненти бізнес логіки рівня додатку, саме тому більша кількість функцій реалізовується індивідуально.

2. Для розробки на фреймворці треба мати розуміння бізнес-процесів, які вимагають реалізації.

2.2 Аналіз властивостей фрейворків

Розглянемо самі популярні HTML/CSS, PHP та Python – фреймворки які допоможуть при створенні сайтів. До основних типів фреймворків відносяться:

1. **Тип HTML/CSS-фреймворк**. Його прикладами є:

Bootstrap - цей фреймворк є дуже відомим та затребуваним. Адаптивність це його головна перевага. Він дозволяє створювати проекти з дуже стильним дизайном. Проект буде автоматично налаштовуватись, враховуючи розмір

екрану комп'ютера або мобільного телефону користувача, який дивиться на сайт. До основних переваг також можна віднести велику кількість стилів, шаблонів, посторінковий дизайн. Фреймворк. Bootstrap став популярним завдяки великої кількості переваг та в ньому майже немає недоліків. Це не тільки HTML/CSS-фреймворк в ньому також увімкнені плагіни то готові стилі JS/Jquery. Знання Bootstrap часто є одним з обов'язкових потреб роботодавця.

Semantic UI – використовується для створення переносних інтерфейсів. Цей фреймворк можна назвати молодим, однак слід відмітити його постійний розвиток. У ньому можна знайти багату кількість кнопок та інших елементів, що необхідні у роботі – картинки, іконки, надписи.

Foundation – цей фреймворк є одним з найпопулярніших у фронтенд фреймворков. Останні версії відмінні покращеним функціоналом для теперішніх мобільних телефонів. Завдяки семантичному підходу, є можливість використовування CSS (Cascading Style Sheets) — це код, що використовується для стилізації вашій веб-старинки, написання більш чистого коду в HTML.

Pure by Yahoo! – у цьому фреймворці є декілька невеликих CSS модулів, які добре підходять для будь якого сучасного проекту. Назва фреймворка характеризує його основну особливість – нічого зайвого лише необхідний, не протяжливий програмний каркас, якій добре підходять для створення сайту.

Uikit – фреймворк, який відділяється завдяки модульною легкою структурою. Є декілька особливостей, які допомагають йому буди іншим на фоні інших сучасних фреймворків. Він має властивості markdown – можливості передусім продивитись сторінку сайту у режимі реально часу. Також можна відмітити синтаксичну під світку для HTML.

2.**Тип PHP** – фреймворки Hypertext Preprocessor: Його прикладами є:

Yii – це старий фремворк, що продовжує оновлюватись у наші дні. Відмічається зручним функціоналом – кешування, високою працездатністю, повною обробкою помилок, можливістю переносу існуючих баз даних, використання jQuery та інше. Фреймворк Yii відмічається своєю простотою, можливо швидкого освоєння його основ, немає ніяких складнощів у роботі та використанні

основного функціоналу. Цей PHP - фреймворк часто советуют людям, котрі роблять перші кроки в розумінні PHP.

CodeIgniter фреймворк – це ще один старий фреймворк, серед основних переваг цього фреймворку:

-швидке встановлення

-постата в використанні

Symfony фреймворк – дуже стабільний фреймворк, який спеціалісти рекомендують для створення великих проектів. Значний функціонал, гнучкість у налаштуваннях – популярність цього фреймворку обумовлення його перевагами та є велика кількість корисних, багаторазових компонентів, які можливо використовувати для створення великого сайту. Сюди можна віднести шаблони, безопасність, налаштування форм.

Laravel фреймворк – лідер різноманітних опитів та рейтингів, які присвячені PHP – фреймворкам. Фреймворк легко можна освоїти, являється ідеальним варіантом для невеликих, також середніх проектів.

PHP Phalcon фреймворк – це фреймворк з відкритим кодом, підтримкою практично усіх сучасних ОС. Працемісткість цього фреймворку знаходиться на дуже високому рівні. Багатьма спеціальними тестуваннями як слідство його популярність. Є можливість використання його на власному сервері.

Flask фреймворк – має мінімальну кількість базового функціоналу, але у любий момент можливо додати необхідний функціонал завдяки великій кількості розширень. Підходить для початкових програмістів у якості першого знайомства з python – фреймворкми.

Web2py фреймворк – його основний є концепцією RAD (rapid application development) – швидка розробка додатків. Це дозволяє програмістам оперативне створювати якісні продукти, при цьому не варто витрачати багато сил або часу. Розробники намагались створити фреймворк максимально простим та ефективним. Повністю відкритий код дозволяє створювати любі динамічні сайти на мові Python. Фреймворк відмінний багатим функціоналом та працездатністю.

Django – фреймворк є дуже популярним у цілому, він займає лікуючу позицію завдяки простоті та функціональності. По-перше для початку не треба глибокі знання мови програмування Python. А завдяки DRY (Don`t repeat yourself) – принципу написання коду спрощується – не потрібно повторно вписувати строки, які вже використовувались, фреймворк самостійно робить це. При цьому код буде залишатись лаконічним та ефективним. Має багато шаблонів, а також стандартну структура. Має також систему адміністрування CMS Django.

TurboGears – фреймворк - це довгожитель свого сегменту, структура цього фреймворка зібрана з WSGI – компонентів які дозволяють створювати любі сучасні проекти. Фреймворк є дійстно потужним, відмічається великим функціоналом. Є підтримка різних баз данних, можливість масштабування, недоліків практично не існує.

Tornado фреймворк – у нього головна особливість, яка відмічається на популярності, це можливе рішення проблеми багатьма способами. Завдяки особливостям серверу, цей фремворк може прекрасно справлятись з тисячами одночасних підключень.

2.3 Дослідження застосування фреймворків для розробки Web- додатків в хмарних сервісах

Праця напряму з HTTP – запитами та віддповідями. Як було видно в останній час, веб-сервери та браузери обмінюються даними по протоколу HTTP – сервери очікують HTTP запити з браузеру, а після цього віддають інформацію в HTTP відповідях. Веб-фреймворки дозволяють писати простим синтаксисом, що буде генеруватись за допомогою серевного коду для розробки з цими запитами та відповідями. Це означає що вам буде легше працювати, взаємодіяти з більш простими боками кодів більш високого рівня, а не з селевими примітивами більш низького рівня. Нижще показується як працює фреймворк Django (Python). Кожна функція view отримує HttpRequest, який в собі має інформацію

про запит, та повинен повернути HttpRequest обьект з форматованим виходом (у цьому випадку це строка)

Запити маршрутизатора до відповідного обробника. Велика кількість сайті дозволяють декілька різних ресурсів, що доступні через відповідні URL адреси. Якщо працювати з ними в одній функції, підтримувати буде важким, тому веб фреймворки предтваляють з себе прості механізми для сопоставлення шаблонів URL адрес, використовуємі для доставки відповідної функції, без зміни базового коду. Різні фреймворки використовують для доставки різні механізми для відповідності. Наприклад Flask(Python) додає маршрути для проглядування функцій використовуя декорати. Django очікує що розробники віділяють список відповідностей URL адрес між шаблоном URL адрес та функцій проглдання.

Спрощувати доступ до даних у запиті. Дані можуть бути кодовані у HTTP запиті різними способами. Для отримання файлів або даних з серверу, HTTP запит GET може кодувати деякі данні потребують URL параметру або структуру URL. HTTP запит POST для оновлення ресурсів на сервері замість цього буде вкоючати оновленну інформацію як POST дані усередені тіла запиту. HTTP – запит може також включати у себе інформацію про триваючі сесії або користувача в cookie зі клієнтської сторони.

Спрощення та абстрагування доступу до бази даних . Веб-сайти використовують бази даних користувачів для зберігання інформації користувачі та і самих користувачах. Веб-фреймворки часто представляють з себе бази даних, котрі обстрагують операції читання, записи, запити та видалення бази даних. Цей рівень абстрагування має назву Object-Relational Mapper (ORM).

Використання ORM має переваги:

Ви можете змінювати лежачу у основі базу даних без необхідності змінювати код, що її використовую. Це дозволяє розробникам оптимізувати її характеристики різних баз даних в залежності від їх використання. Базова перевірка даних може бути реалізована. Це дозволяє легше та безпечніше перевірити, що дані знаходяться у правильному полі типу бази даних, мають правильний фор-

мат та не являються шкідливими. Наприклад такий веб фреймворк як Django представляє ORM та зсилається на об’єкт використовує мий для виділення структуру запису у якості моделі. Модель задае типи полей, котрі повинні бути збережені, що може забезпечити перевірку на рівні поля того, яка інформація може бути збережена. В деяких випадках полей також можна указувати їх максимальний розмір, значення по умовчуванням, параметри списка вибору, текст справки для документації,текст помітки для форм. Модель не містить у собі ніякої інформації про бази даних, так як це параметр конфігурації, котрий може бути збережений та змінений не сумісно з кодом.

Перший параметр фрагменту коду показує модель Django для об’єкта TEAM. Це дозволяє зберігати назву команди та рівень команди такі як символьні поля та виділяти максимальну кількість символів для кожної записи. team_level це поле вибору того це зв’язує варіанти значень на вибір з збереженими даними, це значення по умовчуванню. Модель Django продставляє з себе простий API запитів для виконування пошуку у базах даних.

Багата кількість фреймворків існує практично для кожного мови програмування. При такій великій кількості веб-фреймворків виділити та знайти кращий є дуже важким завданням

2.4 Визначення для фронтенд-розробників кращих JavaScript-фреймворків

Якщо брати до уваги популярність то розповсюдженість по світу з інструментів веб-фреймворків, то ці п’ять є замих замітних з усіх JavaScript та бібліотек. Навколо кожного з цих інструментів склалось значна кількість спільноти розробників . Та якщо треба знайти основу для веб-проекту, то великий шанс що саме ці 5 фреймовків будуть використовуватись.

1. React – в світі JavaScript-у React являється безпосередньо лідером. Для того щоб використати React та гнучко володіти ним потрібно вивчити багату

кількість додаткових інструментів. Наприклад такі інструменти можна використовувати сумісно з React: Redux, MobX, Fluxy, Fluxible, RefluxJS. В React-разработке, кроме того, можно использовать jQuery AJAX, Superagent, Axios и Fetch API. Технологія React.Suspense дозволяє покращити роботу обробки асинхроннох загрузки даних у React додатки. Якщо описувати цю технологіюю дуже коротко , то можна сказати, що вона дозволяє організувати компоненти виконування для деяких умов и при цьому не збивати роботу усього додатку. Також з'явилось нове нововедення – це хукі. Хукі у Reactдозволяють розробнику користуватися найважливішими властивостями React та при цьому обходитись без використання компонентів що основані на класах.

2. Angular –. Планується зробити компілятор Angular Ivy стандартним, що буде доступним на усіх додатках. Основні переваги цієї технології заключається у тому, що її використання дозволяє прискорити процес її працездатності, зменшити розмір додатку, та виросте надійність. Сучасний Angular – це продвинутий модульний фреймворк для фроенд розробки. Якщо раніше для підключеня Angular до сторінки будо достатньо того що просто додати її до HTML коду, то зараз вже розробник мрє імпортувати у свій проект необідні йому модулі Angular. Angular популярний завдяки своєї гнучності. Саме тому ще все актуальні версії Angular1.x. Однак багато розробників використовують Angular2+ тому зо він має іншу архітектуру фреймворка, котра значно змінилась у сторону компонентів. Angular - це повноцінний фреймворк, що дає можливість сучасному веб розробнику мати все необхідне для продуктивної праці. На Angular слід приділяти увагу тім, хто хотів би отримати у своє розпорядження великий набор стандартних інструментів та звести до мінімуму використання сторонніх бібліотек.

3. Backbone.js – це JavaScript фреймворк що заснований на архітектури яка схожа на MVC. Якщо сказати що MVC називається контролером в Backbone називається представленням. Представлення Backbone можуть бути використаними у різних системах шаблонізації Наприклад - Mustache, Handlebars, jQuery-tmpl. У Backbone додатках можна використати строні бібліо-

теки, але слід відмітити той факт, що однією сильною залежністю, є бібліотека Underscore.js. Backbone – дуже легкий в використанні фреймворк котрий дозволяє швидко розробляти одно сторінкові додатки. Якщо потрібен додаток у якому будуть працювати користувачі, що прилежать до розділення можелей, можна використатись масивами Backbone. В моделях, масивах, маршрутах та прдеставленях Backbone можна використовувати випадки. Backbone не можна назвити конкурентом для тіх фреймворків про які було згадано вище, однак цей фреймворк користується своєю популярністю через розробників. Тому Backbone являється досить непоганим фреймворком.

4. Ember - у цьому році вийшов Ember 3.14. У цій версії фреймворку зв'явилось багато нового. Ember схожий на Backbone та Angular. Але крім цього один з найстаріших фреймворків JavaScript. Не дивлячись на це він в плані свїх властивостей, не відстає від своїх прямих конкурентів, які біль сучасні. Наприклад він підтримує технологію відслідних змін властивостей, яка спрощує наблюдання за змінами станом додатку та спрощує візуалізацію цих змін.

5. Vue – ідеї котрі лежать у основі Vue, були взяті у Angular та React але Vue зачасту являється кращим за цих двух у фроендн розробці. У 2021 році Vue встановили собі більше ніж 40 мілйонів разів. Він у своїй основі був націлений на обслідування безпеки React та Angular. Але в ньому знайшлося місцу для вивчення других проектів. Про Vue можна дізнитись з цього звіту те что існує лише 4 його небезпеки, що були вже успішно прибрані. Про Vue є декілька ключових фактів, пр. праці з Vue логіка компоненту, його макет та стилі знаходяться у одному файлі. За виключеням стилів матеріали проекту так як і в Vue знаходяться в React. Взаємодія компонентів у Vue забезпечується за допомогую об'єктів, що зберігають свойства та свій стан компонентів. Цей підхід також ще до того був реалізований в React. Vue схожий на Angular тим що може змішувати HTML розмітку та код JavaScript-у..Одна з причин, по якої Vue слід роздивлятись лише у тому випадку як гарна зміна React-у, залежить у мому що як організовано управління станом додатку. В React додатках при використанні зв'язку React+Redux по мірі розта розмрів додатків прислонюються і процеду-

ри, котрі необхідні ддля управління його станом. Це може привести до того що розробнику замість роботи над додатком прийдеться татити час на работу з механізмом Redux. У Vue для управління станом є створена одна спеціальна бібліотека Vuex. Вона схоже на Flux та створена для Vue спеціально.

3 РОЗРОБКА КОДА ПРОГРАМИ CRM СИСТЕМИ НА ФРЕЙМВОРЦІ JAVASCRIPT, VUE.JS

3.1 Етапи розробки програмного забезпечення Web-додатків

. В процесі створення програми можна виділити такі етапи розробки:

1. *Постановка завдання*

2. *Аналіз задачі та моделювання*.

3. *Розробка або вибір алгоритму вирішення задачі*

4. *Проектування загальної структури програми*)

5. *Кодування*.

6. *Тестування та отладка програми.*

7. *Аналіз результатів*.

8. *Передача замовнику на експлуатацію та публікація отриманих результатів*.

9. *Супровід програми*

Проектувати програму та йти по плану розробки програмного забезпечення необхідне в наслідок наступних факторів.

В залежності від розміру програмних проектів етапи розробки будуть змінюватись, у деяких випадках це будуть дуже деталізовані та бюрократичні етапи, в деяких - просто сформовані у любому виді що будуть комфортними для розробників. Наприклад при будуванні чогось для себе на майданчику и не будете детально планувати кожний крок будівництва, інспектувати, виміряти, але в випадку з будівництвом електростанції або великого заводу, все будете максимально детально сплановано, спроектовано. Кожний час робочих буде розписаний на кожну хвилину, так як ціна помилки буде великою на будь-якому етапі будівництва, ніж при будівництві будки для собаки. Однаково і відбувається при розробці програмного забезпечення, якщо проект великий та

важливий, котрий буде впливати на життя на людей або зв’язаний з великими фінансовими ризиками, то всі етапи розробки програмного забезпечення будуть виконуватись, також детально пророблеими та навіть будуть додаватись і нові етапи та мікроетапи. Все це робиться для того щоб не допустити помилок та реалізувати цей продукт, котрий дуже важливий. Ніж раніше будуть виявленні помилки або дізнаються про неправильний підхід в реалізації той чи іншої дії, тим меншою буде ціна цієї помилки. Іншими словами, в залежності від етапу ціна помилки може збільшуватись від 0 до 10. Наприклад якщо на першому етапі рівень помилки буде рівнятись 1, то на етапі тестування вона може досягати до 9. Тому етапи розробки програмного забезпечення дуже важливі, та кожен розробник повинен дотримуватись їх та спробувати донести це бачення до менеджерів, котрим важливий завжди лише результат. Так як вони або відводять для цього не багато часу або і взагалі не вважають цу необхідним. Наприклад, навіщо при програмуванні виробляти якість вимоги або навіщо здалося це проектування.

3.2 Опис алгоритмів програмного забезпечення Web-додатків

Vue.js - це JavaScript фреймворк з відкритим вихідним кодом для створення призначених для користувача інтерфейсів. Він дуже легко інтегрується в проекти з використанням інших JavaScript бібліотек. Функціонує як веб-фреймворк для розробки односторінкових додатків у реактивному стилі. Vue.js на мою думку є простішим ніж Angular JS так як API збудований набагато простіше в освоєнні. Vue.js має адаптивний дизайн розробники просто прив'язують представлення до існуючої моделі, а Vue.js автоматично спостерігає за змінами в моделі та промальовує представлення, ця функція робить управління станом Vue.js достатньо простим та інтуїтивно зрозумілим. У Vue.js є офіційна документація на багатьох мовах,яка є прикладом в поясненні проектування і розробки у браузері. Любий додаток Vue.js має один центральний екземпляр, для кожного HTML-файла можлива люба кількість екземплярів.

CRM – прикладне програмне забезпечення для організацій, автоматизації стратегій взаємодії з замовниками, підвищення рівня продажів, оптимізація маркетингу, встановлення і покращення бізнес-процесів та послідуючих аналізів результатів. Вона в себе включає: фронтальну частину яка забезпечує обслуговування клієнтів на точках продажів з автономною або централізованою обробкою інформації, операційну частину яка забезпечує авторизацію операцій та оперативну звітність, базу даних, аналітичну підсистему, розподілену систему підтримки продаж. Моя програма є аналітичною CRM, це є звітністю та аналізом інформації.

Перейдемо до представлення моєї CRM системи яка написана на фреймворці JavaScript, Vue.js. Моя CRM система являється домашній помічник у розрахунку власних фінансів. Інтерфейс являється дуже зручним та простим у користуванні. Рахувати власні кошти дуже важливо, так як це може уберегти від незапланованих розтрат. Перейдемо до самої програми та її демонстрації.

Представлено алгоритм роботи CPM системи, блок схеми роботи програми, графічно представлено систему авторизації користувачів у WEB-додатку.

Алгоритм – це кінцева сукупність точно заданих правил рішення деякого класу завдань або набору інструментів, описуючих порядок дій виконавця для рішення відведеної задачі.

Поняття алгоритму відноситься до первинних, основним, базовим поняттям математики. Обчислювальні процеси алгоритмічного характеру відомі людству с давніх часів.

Блок-схема – це графічне представлення алгоритму яке дуже часто використовується у програмуванні. Блок-схема – це розповсюджений ти схем, описуючих алгоритми або процеси, в яких окремі кроки відображаються у вигляді блоків різної форми, з’єднаних між собою лініями які указують напрямок послідовності (Рис.2.1)

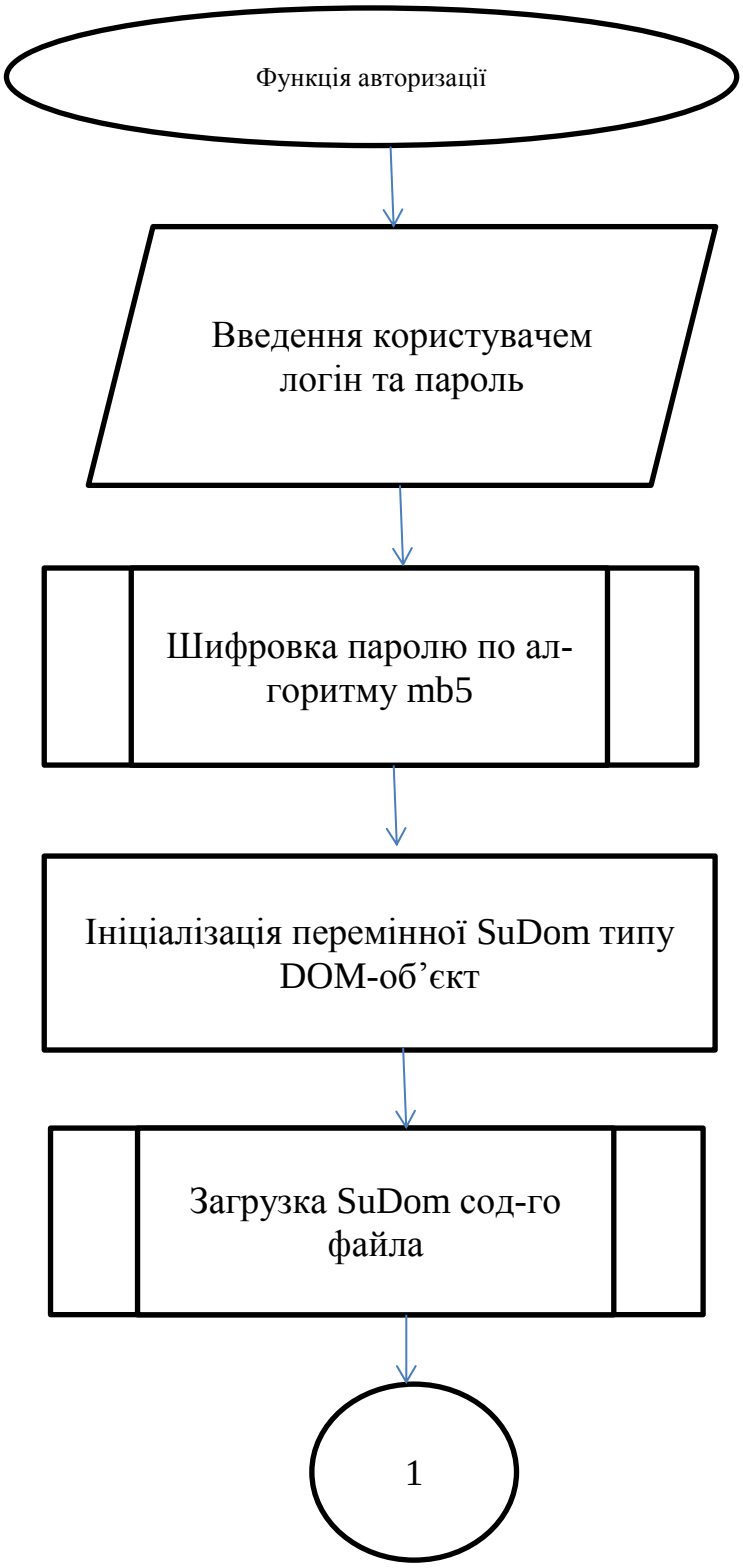

Рисунок 2.1 – Алгоритм авторизації

На першій блок-схемі відображено алгоритм авторизації користувача до додатку.

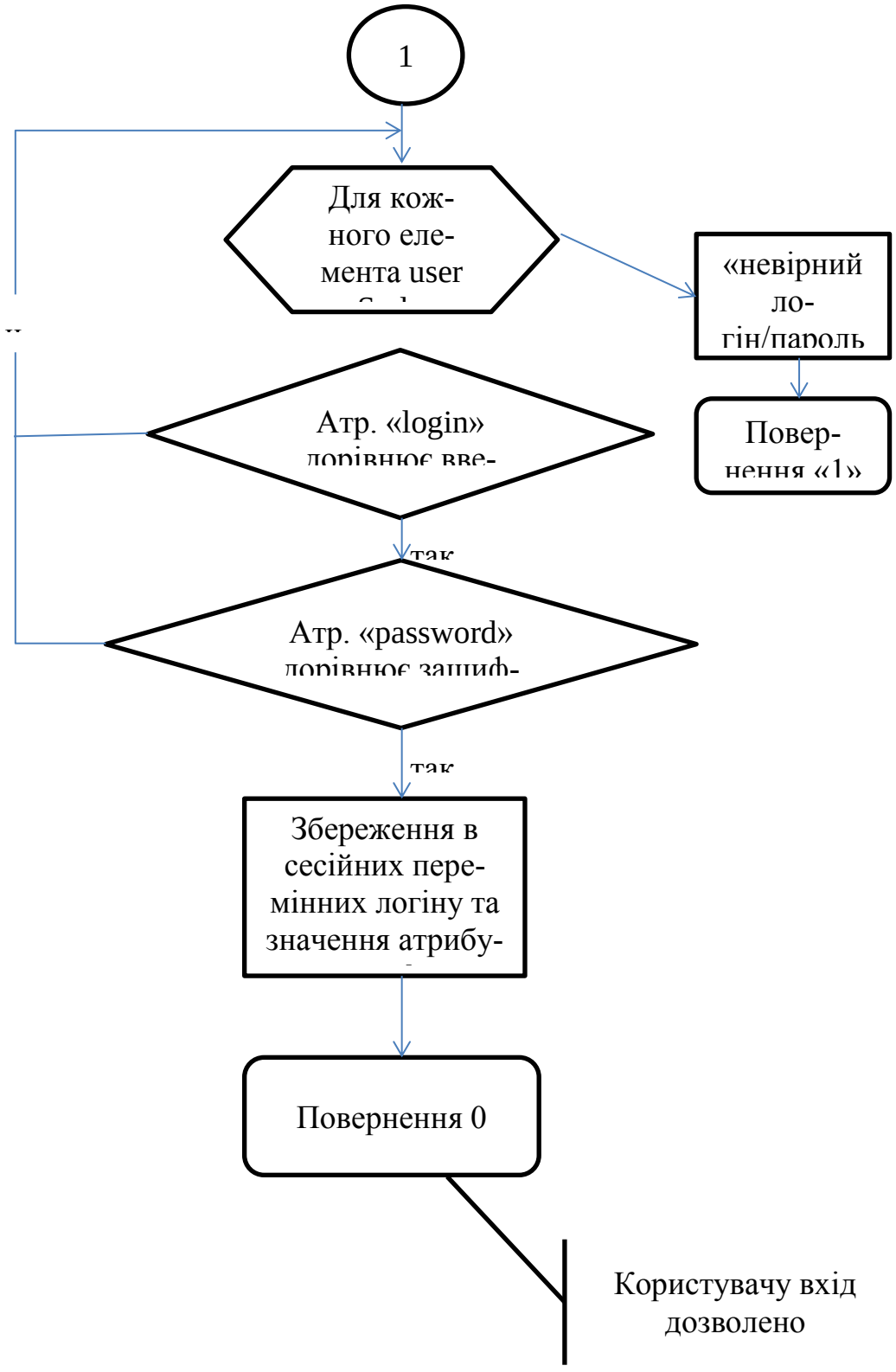

Рисунок 2.2 – Алгоритм авторизації

Також для того щоб користувач ідентифікувався у базі, його потрібно зареєструвати, код та вигляд сторінки буде надано у наступній главі. Ця блок-схема показує простий алгоритм реєстрації

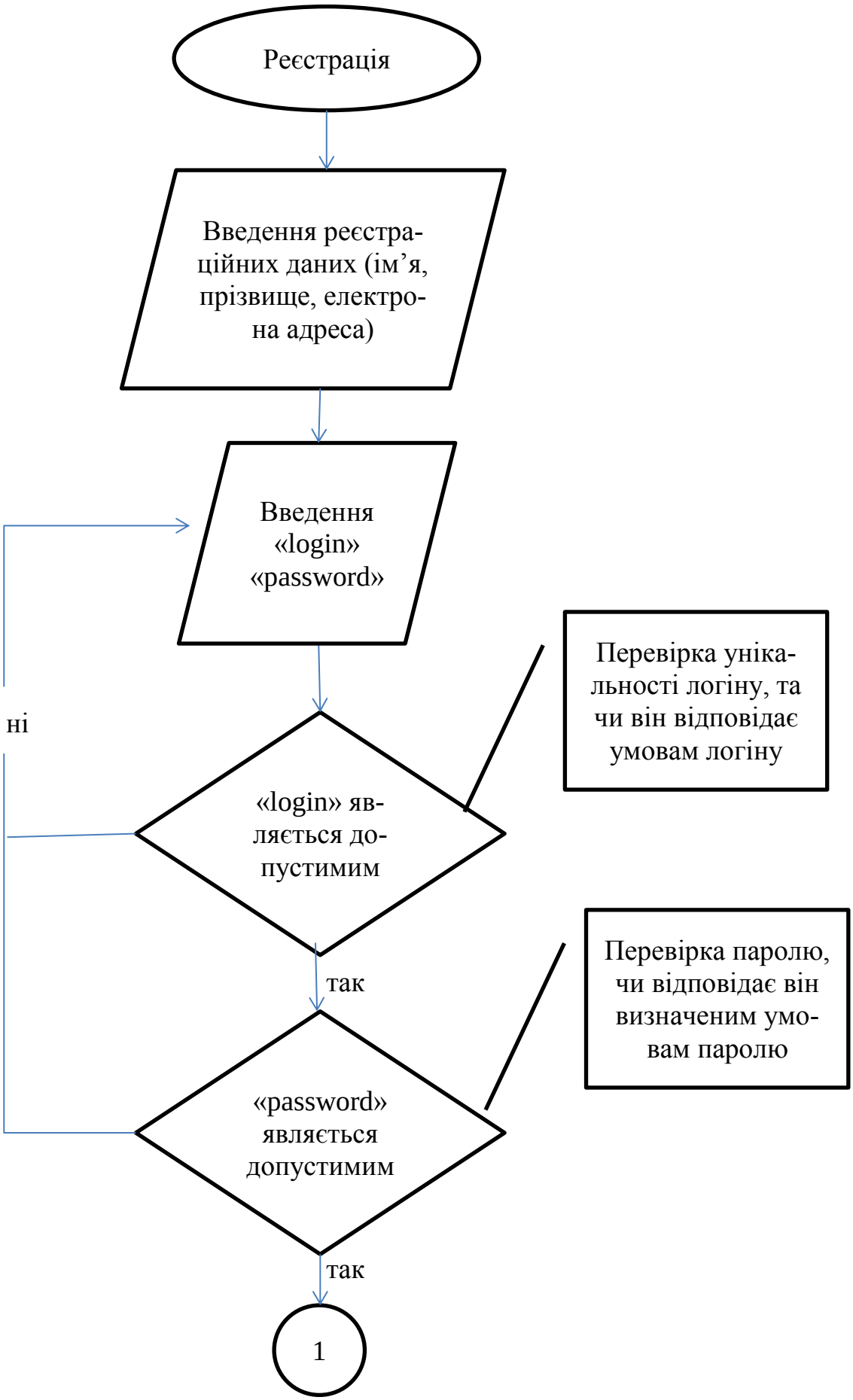

Рисунок 2.3 – Алгоритм процесу реєстрації

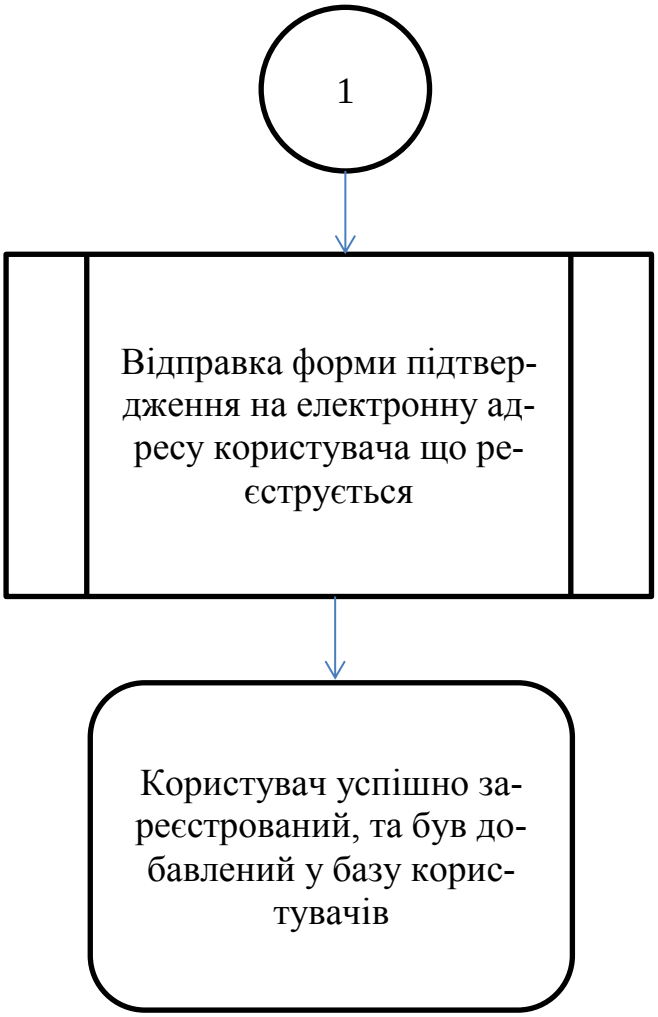

Рисунок 2.4 – Алгоритм процесу реєстрації

На цій блок-схемі показано кроки користувача при реєстрації у системі, та якім умовам потрібно відповідати. У наступній, та останній блок схемі наведено основний алгоритм роботи програми.

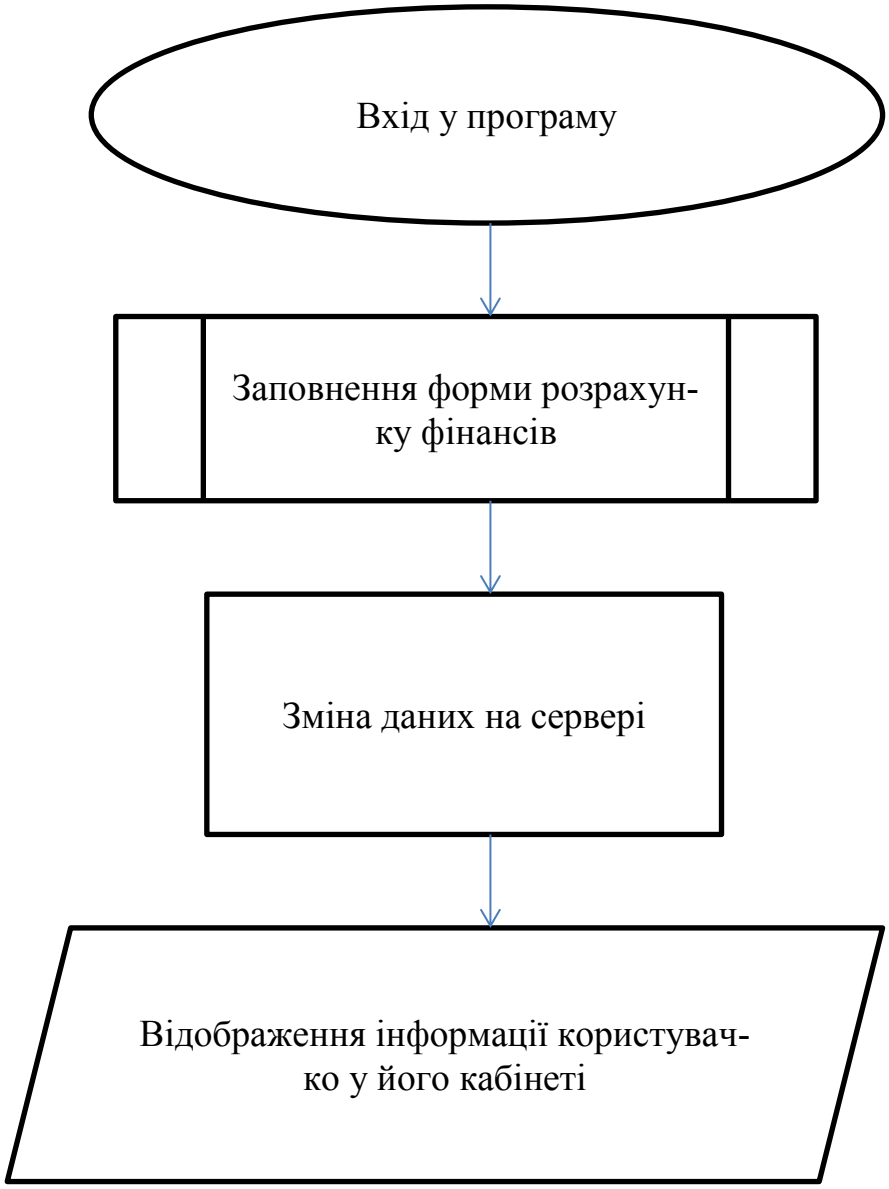

Рисунок 2.5 – Основний алгоритм роботи додатку

3.3 Рекомендації щодо створення програмного забезпечення Web-додатків

На початку зустрічає сторінка з входом для переходу до свого власного кабінету

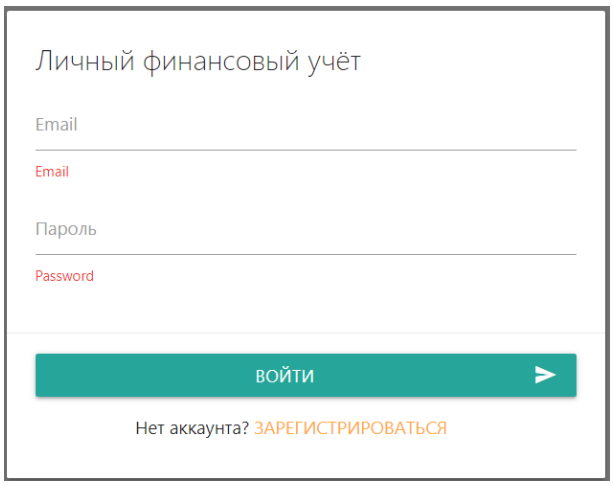

Рисунок 2.6 – Вікно авторизації до кабінету пористувача

Код сторінки:

```
<template>
  <form class="card auth-card" @submit.prevent="submitHandler">
    <div class="card-content">
      <span class="card-title">Учет личных финансов</span>
      <div class="input-field">
        <input id="email" type="text" v-model.trim="email" :class="{invalid: ($v.email.$dirty && !$v.email.required) || ($v.email.$dirty && !$v.email.email)}">
        <label for="email">Email</label>
        <small class="helper-text invalid" v-if="$v.email.$dirty && !$v.email.required">
          Поле Email не должно быть пустым</small>
```

```
            <small class="helper-text invalid"                    v-else-
if="$v.email.$dirty && !$v.email.email">
            Введите корретный Email</small>
          </div>
          <div class="input-field">
           <input id="password" type="password" v-model.trim="password"
              :class="{invalid: ($v.password.$dirty && !$v.password.required) || ($v.
password.$dirty && !$v.password.minLength)}" >
           <label for="password">Пароль</label>
           <small class="helper-text invalid"                    v-
if="$v.password.$dirty && !$v.password.required">
            Введите пароль
           </small>
           <small class="helper-text invalid" v-else-
if="$v.password.$dirty && !$v.password.minLength">
            Па-
роль должен быть {{$v.password.$params.minLength.min}} символов. Сейчас он
{{password.length}}
           </small>
          </div>
         </div>
         <div class="card-action">
          <div>
           <button                  class="btn waves-effect waves-light auth-submit"
type="submit">
            Войти
            <i class="material-icons right">send</i>
           </button>
          </div>
        <p class="center">
```

```
            Нет аккаунта?
            <router-link to="/register">Зарегистрироваться</router-link>
          </p>
        </div>
      </form>
    </template>
    <script>
    import {email, required, minLength} from 'vuelidate/lib/validators'
    import messages from '@/utils/messages'
    export default {
      name: 'login',
      data: () => ({
        email: '',
        password: ''
      }),
      validations: {email: {email, required},
password: {required, minLength: minLength(6)} },
      mounted() { if (messages[this.$route.query.message]) {
          this.$message(messages[this.$route.query.message]) }
      },
      methods: {
        async submitHandler() {
          if (this.$v.$invalid) {
            this.$v.$touch()
            return}
          const formData = {email: this.email, password: this.password }
            try { await this.$store.dispatch('login', formData)
            this.$router.push('/')
          } catch (e) {}
        }}}
```

</script>

На цій сторінці відбувається авторизація користувача до системи, якщо користувач ще не зареєструвався у системі по кнопці «Зарегистрироваться» користувача переводять до наступної сторінки с формою реєстрації.

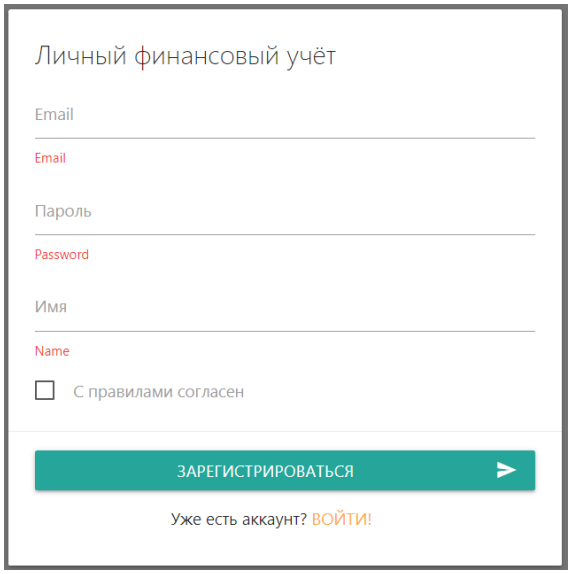

Рисунок 2.7 – Вікно реєстрації користувача до системи

Код цієї сторінки:

```
<template>
  <form class="card auth-card" @submit.prevent="submitHandler">
    <div class="card-content">
      <span class="card-title">Домашняя бухгалтерия</span>
      <div class="input-field">
        <input id="email" type="text" v-model.trim="email"
          :class="{invalid: ($v.email.$dirty && !$v.email.required) || ($v.email.$
dirty && !$v.email.email)}" >
        <label for="email">Email</label>
        <small class="helper-text invalid"                                    v-
if="$v.email.$dirty && !$v.email.required">
```

```
Поле Email не должно быть пустым</small>
<small class="helper-text invalid" v-else-if="$v.email.$dirty && !$v.email.email">
Введите корретный Email</small>
</div>
<div class="input-field">
<input id="password" type="password" v-model.trim="password"
:class="{invalid: ($v.password.$dirty && !$v.password.required) || ($v.password.$dirty && !$v.password.minLength)}" >
<label for="password">Пароль</label>
<small class="helper-text invalid" v-if="$v.password.$dirty && !$v.password.required">
Введите пароль
</small>
<small class="helper-text invalid"
v-else-if="$v.password.$dirty && !$v.password.minLength">
Па-
роль должен быть {{$v.password.$params.minLength.min}} символов. Сейчас он {{password.length}}
</small>
</div>
<div class="input-field">
<input id="name" type="text" v-model.trim="name"
:class="{invalid: $v.name.$dirty && !$v.name.required}">
<label for="name">Имя</label>
<small class="helper-text invalid"
v-if="$v.name.$dirty && !$v.name.required" >
Введите ваше имя
</small>
</div>
```

```
      <p>
        <label>
          <input type="checkbox" v-model="agree" />
          <span>С правилами согласен</span>
        </label>
      </p>
    </div>
    <div class="card-action">
      <div>
        <button class="btn waves-effect waves-light auth-submit"
           type="submit">
          Зарегистрироваться
          <i class="material-icons right">send</i>
        </button>
      </div>
<p class="center">
        Уже есть аккаунт?
        <router-link to="/login">Войти!</router-link>
      </p>
    </div>
  </form>
</template> <script>
import {email, required, minLength} from 'vuelidate/lib/validators'
export default { name: 'register',  data: () => ({ email: '', password: '', name: '',
    agree: false
  }),
  validations: {
    email: {email, required},
    password: {required, minLength: minLength(6)},
    name: {required},
```

```
    agree: {checked: v => v}
  },
  methods: {
    async submitHandler() {
      if (this.$v.$invalid) {
        this.$v.$touch()
        return}
const formData = {
        email: this.email,
        password: this.password,
        name: this.name}
      try {
        await this.$store.dispatch('register', formData)
        this.$router.push('/')
       } catch (e) {}}}}
</script>
```

Тепер можна перейти до самої системи, та як вона виглядає, та дослідити її функціонал.

Перша сторінка «Счет» на ній відображається заданий вами курс валют на сегодні для відстеження його рівня

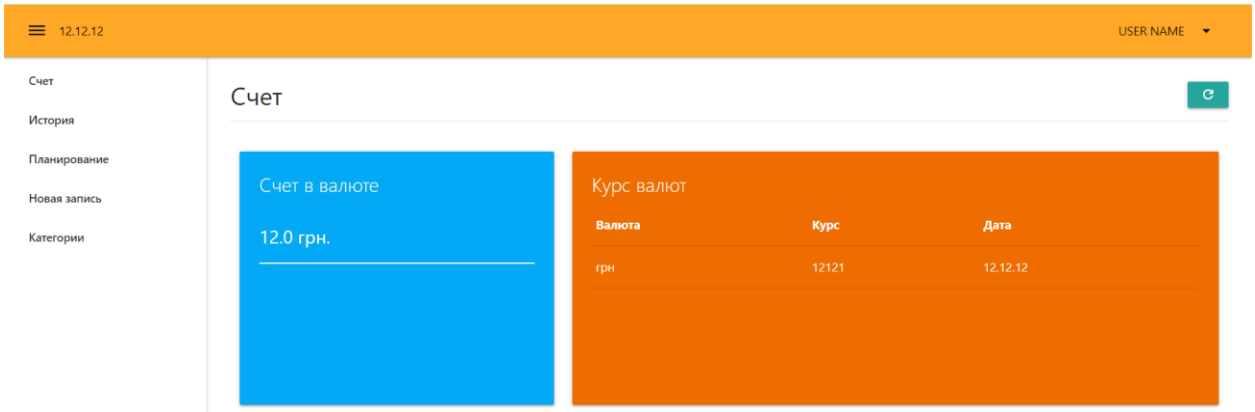

Рисунок 2.8 – Вікно сторінки «Рахунок» у власному кабінеті користувача

Код цієї сторінки:

```
<template>
  <div>
    <div class="page-title">
      <h3>Счет</h3>
 <button class="btn waves-effect waves-light btn-small" @click="refresh">
        <i class="material-icons">refresh</i>
      </button>
    </div>
<Loader v-if="loading" />
<div v-else class="row">
<HomeBillrates="currency.rates"/>
<HomeCurrency :rates="currency.rates" :date="currency.date"/>
 </div>
  </div>
</template>
<script>
import HomeBill from '@/components/HomeBill'
import HomeCurrency from '@/components/HomeCurrency'
export default { name: 'home',
  data: () => ({ loading: true,  currency: null }),
  async mounted() {
    this.currency = await this.$store.dispatch('fetchCurrency')
    this.loading = false },
  methods: { async refresh() { this.loading = true
      this.currency = await this.$store.dispatch('fetchCurrency')
      this.loading = false }},
  components: {
```

HomeBill, HomeCurrency }} </script>

Наступна сторінка функціоналу «История» в ній можна подивитись історію своїх платежів то розтрат.

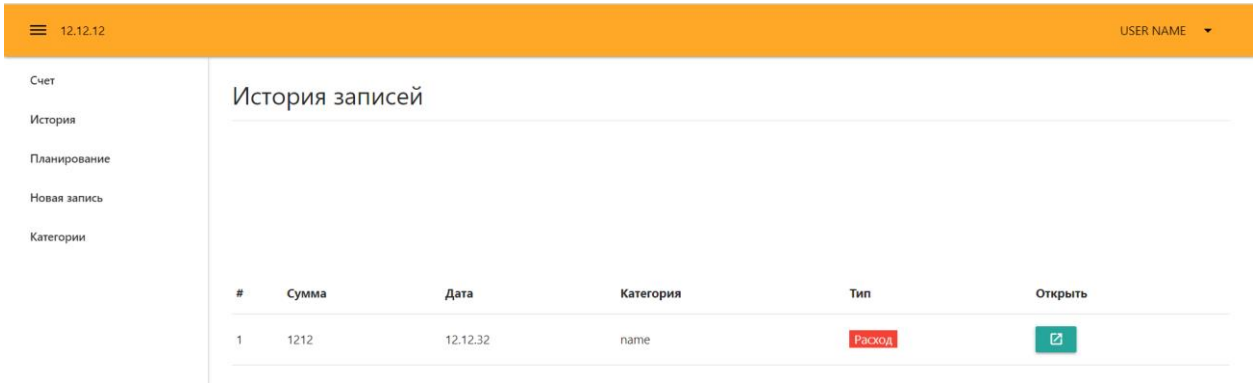

Рисунок 2.9 – Вікно «Історія» у власному кабінеті користувача

Код цієї сторінки:

```
<template>
  <div>
    <div class="page-title">
      <h3>История записей</h3>
    </div>

    <div class="history-chart">
      <canvas ref="canvas"></canvas>
    </div>
<Loader v-if="loading"/>
<p class="center" v-else-if="!records.length">
      Записей пока нет.
      <router-link to="/record">Добавьте первую</router-link>
    </p>
<section v-else>
```

```
      <HistoryTable :records="items"/>
<Paginate
       v-model="page"
       :page-count="pageCount"
       :click-handler="pageChangeHandler"
       :prev-text="'Назад'"
       :next-text="'Вперед'"
       :container-class="'pagination'"
       :page-class="'waves-effect'"
      />
    </section>
  </div>
</template>
<script>
import paginationMixin from '@/mixins/pagination.mixin'
import HistoryTable from '@/components/HistoryTable'
import { Pie } from 'vue-chartjs'
export default {
  name: 'history',
  extends: Pie,
  mixins: [paginationMixin],
  data: () => ({
    loading: true,
    records: []
  }),
  async mounted() {
    this.records = await this.$store.dispatch('fetchRecords')
    const categoires = await this.$store.dispatch('fetchCategories')
this.setup(categoires)
this.loading = false
```

```
},
methods: {
  setup(categoires) {
    this.setupPagination(
      this.records.map(record => {
        return {
          ...record,
          categoryName: categoires.find(c => c.id === record.categoryId)
            .title,
          typeClass: record.type === 'income' ? 'green' : 'red',
          typeText: record.type === 'income' ? 'Доход' : 'Расход'}}))
    this.renderChart({
      labels: categoires.map(c => c.title),
      datasets: [
        {
          label: 'Расходы по категориям',
          data: categoires.map(c => {
            return this.records.reduce((total, r) => {
              if (r.categoryId === c.id && r.type === 'outcome') {
                total += +r.amount
              }
              return total
            }, 0)
          }),
          backgroundColor: [
            'rgba(255, 99, 132, 0.2)',
            'rgba(54, 162, 235, 0.2)',
            'rgba(255, 206, 86, 0.2)',
            'rgba(75, 192, 192, 0.2)',
            'rgba(153, 102, 255, 0.2)',
```

```
        'rgba(255, 159, 64, 0.2)'
      ],
      borderColor: [
        'rgba(255, 99, 132, 1)',
        'rgba(54, 162, 235, 1)',
        'rgba(255, 206, 86, 1)',
        'rgba(75, 192, 192, 1)',
        'rgba(153, 102, 255, 1)',
        'rgba(255, 159, 64, 1)'],
      borderWidth: 1}]})}},
  components: {HistoryTable }} </script>
```

Наступна сторінка «Планирование» в ній відображаються категорії та сума яку ви задали для цієї категорії, ці категорії можна змінювати за бажанням. На ній є шкала відображення проценту потрачених коштів на ту чи іншу категорію.

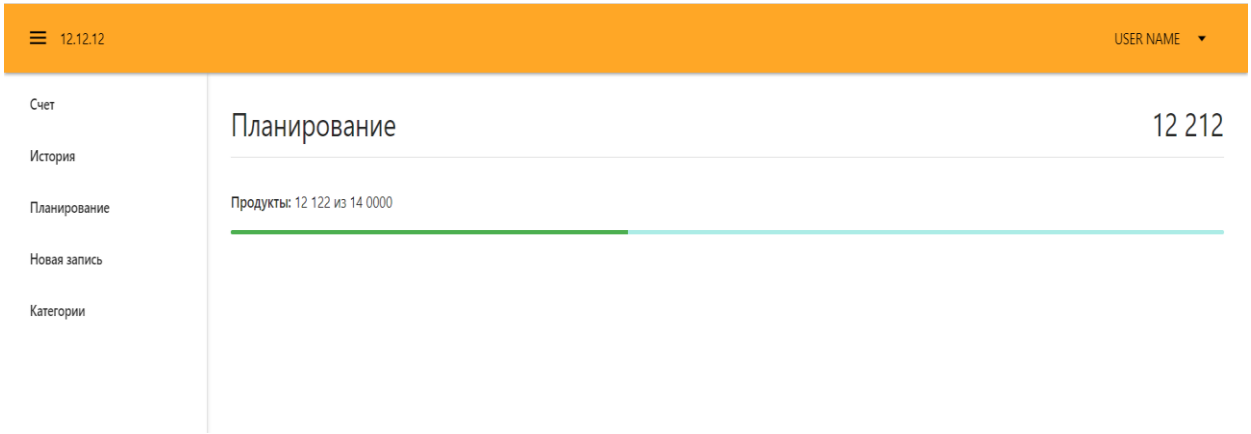

Рисунок 2.10 – Вікно «Планування» у власному кабінеті користувача

```
<template>
  <div>
    <div class="page-title">
      <h3>Планирование</h3>
      <h4>{{info.bill | currency('RUB')}}</h4>
    </div>
```

```
<Loader v-if="loading" />
<p class="center" v-else-if="!categories.length">Категорий пока нет. <router-link to="/categories">Добавить новую категорию</router-link></p>
<section v-else>
  <div v-for="cat of categories" :key="cat.id">
    <p>
      <strong>{{cat.title}}:</strong>
      {{cat.spend | currency}} из {{cat.limit | currency}}
    </p>
    <div class="progress" v-tooltip="cat.tooltip">
      <div
        class="determinate"
        :class="[cat.progressColor]"
        :style="{width: cat.progressPercent + '%'}"
      ></div>
    </div>
  </div>
</section>
</template>
<script>
import {mapGetters} from 'vuex'
import currencyFilter from '@/filters/currency.filter'
export default {
  name: 'planning',
  data: () => ({
    loading: true,
    categories: []
  }),
  computed: {
```

```
    ...mapGetters(['info'])
  },
  async mounted() {
    const records = await this.$store.dispatch('fetchRecords')
    const categoires = await this.$store.dispatch('fetchCategories')
this.categories = categoires.map(cat => {
    const spend = records
      .filter(r => r.categoryId === cat.id)
      .filter(r => r.type === 'outcome')
      .reduce((total, record) => {
        return total += +record.amount
      }, 0)
const percent = 100 * spend / cat.limit
    const progressPercent = percent > 100 ? 100 : percent
    const progressColor = percent < 60
      ? 'green' : percent < 100 ? 'yellow'
        : 'red'
    const tooltipValue = cat.limit - spend
    const tooltip = `${tooltipValue < 0 ? 'Превышение на' : 'Осталось'} ${curr
encyFilter(Math.abs(tooltipValue))}`
return {
    ...cat,
    progressPercent,
    progressColor,
    spend,
    tooltip
  }}) this.loading = false}}
</script>
```

Також є функція додатку нових категорій, та можливість задати це буде дохід чи розхід.

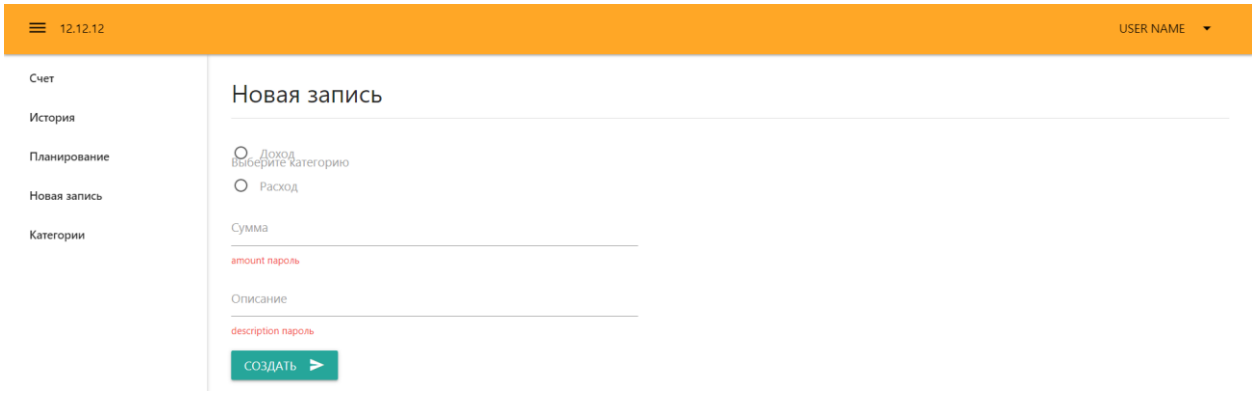

Рисунок 2.11 – Вікно «Створення нового запису »у власному кабінеті користувача

Код цієї сторінки:

```
<template>
  <div>
    <div class="page-title">
      <h3>Новая запись</h3>
    </div>
<Loader v-if="loading" />
<p class="center" v-else-if="!categories.length">Категорий пока нет. <router-link to="/categories">Добавить новую категорию</router-link></p>
<form class="form" v-else @submit.prevent="handleSubmit">
    <div class="input-field" >
      <select ref="select" v-model="category">
        <option v-for="c in categories" :key="c.id" :value="c.id" >{{c.title}}</option>
      </select>
```

```
        <label>Выберите категорию</label>
      </div>
<p>
        <label>
          <input class="with-gap" name="type" type="radio" value="income"
            v-model="type"/>
          <span>Доход</span>
        </label>
      </p>
<p>
        <label>
          <input class="with-gap" name="type" type="radio" value="outcome"
            v-model="type"/>
          <span>Расход</span>
        </label>
      </p>
<div class="input-field">
        <input id="amount" type="number" v-model.number="amount"
          :class="{invalid: $v.amount.$dirty && !$v.amount.minValue}">
        <label for="amount">Сумма</label>
        <span v-if="$v.amount.$dirty && !$v.amount.minValue"
          class="helper-text invalid">
          Минимальная значение {{$v.amount.$params.minValue.min}}
        </span>
      </div>
<div class="input-field">
        <input id="description" type="text" v-model="description"
          :class="{invalid: $v.description.$dirty && !$v.description.required}">
        <label for="description">Описание</label>
        <span v-if="$v.description.$dirty && !$v.description.required"
```

```
            class="helper-text invalid">
            Введите описание
          </span>
      </div>
<button class="btn waves-effect waves-light" type="submit">
        Создать
        <i class="material-icons right">send</i>
      </button> </form> </div> </template>
<script>
import {required, minValue} from 'vuelidate/lib/validators'
import {mapGetters} from 'vuex'
export default {
  name: 'record',
  data: () => ({ loading: true,  select: null,  categories: [], category: null,
    type: 'outcome',  amount: 1, description: ''}),
  validations: { amount: {minValue: minValue(1)}, description: {required}},
  async mounted() {
    this.categories = await this.$store.dispatch('fetchCategories')
    this.loading = false
if (this.categories.length) {
      this.category = this.categories[0].id}
setTimeout(() => {
      this.select = M.FormSelect.init(this.$refs.select)
      M.updateTextFields()}, 0) },
  computed: {
    ...mapGetters(['info']),
    canCreateRecord() {
      if (this.type === 'income') {
        return true}
return this.info.bill >= this.amount}},
```

```
      methods: {
        async handleSubmit() {
          if (this.$v.$invalid) {
            this.$v.$touch()
            return}
      if (this.canCreateRecord) {
            try {
              await this.$store.dispatch('createRecord', {
                categoryId: this.category,
                amount: this.amount,
                description: this.description,
                type: this.type,
                date: new Date().toJSON()})
              const bill = this.type === 'income'
                ? this.info.bill + this.amount
                : this.info.bill - this.amount
               await this.$store.dispatch('updateInfo', {bill})
              this.$message('Запись успешно создана')
              this.$v.$reset()
              this.amount = 1
              this.description = ''} catch (e) {}} else {
             this.$message(`Недостаточно средств на счете (${this.amount -
this.info.bill})`)
            }}},
        destroyed() {
          if (this.select && this.select.destroy) {
            this.select.destroy()
          }}}</script>
```

Тепер можна перейти до біль технічних файлів, за допомогою яких CRM система функціонує, має дизайн.

Первшим файлом є index.css в цьому файлі задані всі стилі які використовувались в CRM системі. Для більшої оптимізації з файлу були видалені всі табуляції.

Наступний файл «main.js». В цьому файлі підключаються усі модулі які потрібні для роботи. Такі як бібліотеки стилів(materialize.css), загружчик, плагін повідомлень, плагін бази данних. Задаємо URL, домен автора, айді проекту, ключ API. Та в кінці «маунтимо» в нашу програму.

Файл «router.js» являється каркасом всього додатку. В ньому ми патчимо усі сторінки додатку, та задаємо зв'язок між ними. Код цього файлу:

```
import Vue from 'vue'
import Router from 'vue-router'
import firebase from 'firebase/app'
Vue.use(Router)
const router = new Router({
  mode: 'history',
  base: process.env.BASE_URL,
  routes: [
    {path: '/login',                                                    name: 'login',
meta: {layout: 'empty'},component: () => import('./views/Login.vue') },
router.beforeEach((to, from, next) => {
  const currentUser = firebase.auth().currentUser
  const requireAuth = to.matched.some(record => record.meta.auth)
if (requireAuth && !currentUser) {
   next('/login?message=login')
  } else {next()}})
export default router
```

Файл «MainLayout.vue» в ньому задається рас положення для основних блоків відносно один одного у рамках сторінки або другого іншого інтерфейсу. Цей «Layout» використовується в сторінках додатку(самого функціоналу) не враховуючи сторінки авторизації та реєстрації. Код цього файлу:

```
<template>
  <div>
    <Loader v-if="loading"/>
    <div class="app-main-layout" v-else>
      <Navbar @click="isOpen = !isOpen"/>
<Sidebar v-model="isOpen" :key="locale"/>
<main class="app-content" :class="{full: !isOpen}">
        <div class="app-page">
          <router-view/></div></main>
<div class="fixed-action-btn">
        <router-link class="btn-floating btn-large blue" to="/record"
          v-tooltip="'Создать новую запись'">
          <i class="large material-icons">add</i>
        </router-link> </div></div></div></template>
<script>
import Navbar from '@/components/app/Navbar'
import Sidebar from '@/components/app/Sidebar'
import messages from '@/utils/messages'
export default {
  name: 'main-layout',
  data: () => ({
    isOpen: true,
    loading: true }),
  async mounted() {if (!Object.keys(this.$store.getters.info).length) {
      await this.$store.dispatch('fetchInfo') }
```

```
this.loading = false},
  components: { Navbar, Sidebar},
  computed: {error() {return this.$store.getters.error},
    locale() {return this.$store.getters.info.locale}},
  watch: {error(fbError) {
      this.$error(messages[fbError.code] || 'Что-то пошло не так') }}}
</script>
```

Також є другий Layout назва цього файлу «EmptyLayout.vue», цей Layout використовується на сторінках авторизації та реєстрації і не на яких більше. Код цього файлу:

```
<template>
  <div class="grey darken-1 empty-layout">
    <router-view /></div></template>
<script>
import messages from '@/utils/messages'
export default {computed: {error() {return this.$store.getters.error}},
  watch: {error(fbError) {
      this.$error(messages[fbError.code] || 'Что-то пошло не так') }}}
</script>
```

Та останній файл, але не останній по важливості «App,vue». В цей файл імпортовано попередні два Layout, та в подальшому буде використовуватись саме App.vue для більшої оптимізації додатку, також в цей файл імпортуються бібліотеки css (index.css, materialize.css). Код цього файлу:

```
<template>
  <div id="app">
    <component :is="layout">
      <router-view/>
    </component>
```

```
  </div>
</template>
<script>
import EmptyLayout from '@/layouts/EmptyLayout'
import MainLayout from '@/layouts/MainLayout'
export default {
  computed: {
    layout() {
      return (this.$route.meta.layout || 'empty') + '-layout'}},
  components: {EmptyLayout, MainLayout}}
</script><style lang="scss">
@import '~materialize-css/dist/css/materialize.min.css';
@import 'assets/index.css';
</style>
```