Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

НА ТЕМУ:

“Розробка програмного забезпечення для оптимізації бізнес-процесів підприємства”

Студента 2м курсу, 5 групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Чевтаєв Микита Валерійович

Науковий керівник кандидат технічних наук, доцент _____ підпис керівника — Савченко Тетяна Віталіївна

Гарант освітньої програми доктор технічних наук, професор _____ підпис керівника — Криворучко Олена Володимирівна

Київ 2019

Зміст

					КНТЕУ 121-05-23.МР		
					«Розробка програмного забезпечення для оптимізації бізнес-процесів підприємства»	*Стадія*	*Аркуш*	*Аркушів*
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		*Зм*	*2*	*54*
Зав.каф.		*Криворучко О.В.*						
Керівник		*Савченко Т.В.*						
Гарант		*Криворучко О.В.*						
Розробив		*Чевтаєв М.В.*			*Зміст*	Факультет обліку, аудиту та інформаційних систем, 2м курс, 5 група		

ВСТУП

В сучасних умовах мінливого бізнес-середовища головним завданням бізнесу стає швидке реагування на ці зміни і таке ж швидке впровадження адекватних змін в організації і веденні бізнесу. Оптимізація бізнес-процесів є однією з основних, стратегічно важливих завдань підприємства, що визначають усю його подальшу ефективну діяльність.

Сучасне виробництво повинно відповідати підвищеним вимогам, а саме: мати високу гнучкістю, необхідність оновлення продукції значним ускладненням технологій, модернізуватися а, отже – постійно вдосконалюватися.

Актуальність дослідження бізнес-процесів визначається тим, що сучасні підприємства змушені постійно займатися поліпшенням своєї діяльності. Це вимагає розробки нових технологій і прийомів ведення бізнесу, підвищення якості кінцевих результатів діяльності і, звичайно, впровадження нових, більш ефективних методів управління і організації діяльності підприємств.

Мета оптимізації бізнес-процесів як у вигляді послідовних поліпшень, так і радикальних на базі методів реінжинірингу є поліпшення всіх або окремих якісних і кількісних параметрів бізнес-процесу.

Об'єкт дослідження є оптимізацію бізнес-процесів підприємства. Оптимізація бізнес-процесів є одним з аспектів організаційного розвитку, за якого ряд дій приймається власником процесу для виявлення, аналізу та покращення існуючих бізнес-процесів на підприємстві у відповідності з поставленими цілями і завданнями, таких як збільшення потенціалу підприємства, а разом з тим і збільшення прибутку та зниження витрат.

Оптимізація бізнес-процесів спрямована на підвищення ефективності управління бізнес-процесами і формування стандарту діяльності компанії. У свою чергу, стандарт роботи компанії є основою для створення внутрішніх регламентів та посадових інструкцій співробітників, організації підготовки персоналу, а також вирішення завдань з автоматизації бізнес-процесів.

Предмет дослідження являє собою розробку CRM-системи, яка потрібна для аналізу ефективності продажів за допомогою Node.js, Express та Angular.

Задачі дослідження:

- Виконати алгоритм створення складного fullstack-додатку на прикладі CRM
- Реалізувати CRM-систему яка буде аналізувати ефективність продажів

Наукова новизна дослідження полягає в створенні комплексної CRM-системи, яка повинна допомагати автоматизувати весь цикл відносин з клієнтом і давати підстави для нових продажів. Це особливо важливо, якщо врахувати, що більшість компаній купують CRM-технології по окремих частинах.

Метод дослідження являє собою MEAN Stack-розробку - це розробка веб-додатку повного циклу, що включає технології JavaScript як і в Frontend-розробку, так і Backend-розробку.

РОЗДІЛ 1. ОПТИМІЗАЦІЯ БІЗНЕС-ПРОЦЕСІВ

1.1. Поняття бізнес-процесів

Бізнес-процеси - це ланцюжки дій і подій, які відбуваються в компанії і призводять до кінцевого результату. Приклад: прийом на роботу і адаптація співробітника, закупівля і використання програмного продукту, розсилка і т.д.

Одним із завдань оптимізації бізнес-процесів, як у вигляді послідовних поліпшень, так і радикальних на базі методів реінжинірингу, є поліпшення всіх або окремих якісних і кількісних параметрів бізнес-процесу.

Серед основних принципів оптимізації бізнес-процесів слід виділити такі:

- відповідність поліпшення бізнес-процесів стратегічним цілям підприємства;
- орієнтація на внутрішніх та зовнішніх споживачів;
- наявність критеріїв оптимізації бізнес-процесів;
- наявність власників бізнес-процесів, які відповідальні за їх оптимізацію.

Процеси можуть бути однократними (наприклад, це розробка корпоративного сайту компанії) і циклічними, повторюваними (підготовка розсилки, виробництво продукції і т.д.). Також процеси можуть бути укрупненими (проведення рекламної акції) і дробовими (розробка дизайну банерів). Дробові процеси можуть входити до складу разових проектів або різних процесів (наприклад, розробка дизайну банерів може бути для розсилки, для контексту, для оформлення віджета і т.д.).

Якщо в компанії бізнес-процеси не налагоджені, вони призводять до втрати коштів, сил, нервів і в кінцевому підсумку до програної конкурентній боротьбі. Такі процеси можуть бути:

- неефективними - при великих затратах приносити мінімально прийнятний варіант, задіяти занадто багато ресурсів;
- повільними - через неузгодженість учасників процесу і відсутності чітко визначених етапів і термінів, відведених на кожну задачу;
- ненадійними - такі процеси можуть призвести не до очікуваного результату, а до його часткової реалізації або до повної відсутності підсумкового результату;
- дублюючими - не найгірший варіант, однак такі процеси перевантажують працівників і споживають більше ресурсів, ніж це робить один оптимізований процес;
- надлишковими - занадто велика кількість етапів, завдань і зв'язків перевантажують процес, і в підсумку досягнення мети відкладається або обходиться занадто дорого;
- зайвими - в компанії зберігаються старі процеси або процеси, які стали неактуальними в рамках поточної діяльності;

Методи оптимізації бізнес-процесів можна поділити на три групи:

1) Формалізовані універсально-принципові методи, які засновані на застосуванні успішного досвіду і формалізованих принципів для побудови ефективних бізнес-процесів. Дані методи є універсальними і вони підходять для оптимізації будь-яких бізнес-процесів для будь-якого бізнесу і практично не залежать від його специфіки.

2) Методи засновані на вивченні, аналізі і подальшому копіюванні елементів процесів успішних компаній, що займаються схожими видами діяльності. Претендентами на вивчення і копіювання їх успішного досвіду в першу чергу є лідери-конкуренти. Практика показала, що останнім часом

багато підприємств ефективно впровадили у себе технологічні ноу-хау, запозичивши їх у підприємств, що працюють в інших галузях бізнесу.

3) Методи групової роботи. Ця група методів об'єднала різні технології роботи в команді: метод мозкового штурму, метод групового рішення задач тощо. їх використання дозволяє розробити нові ефективні рішення, раніше нікому не відомі, що дозволяє підприємству бути лідером за технологіями, що використовуються.

Використання цих методів на підприємствах залежить від певних факторів, таких як рівень та частота змін, характер організації, специфіка діяльності.

Ідея оптимізації ключових параметрів бізнес-процесів отримала розвиток в науковому напрямку "скорочення часу циклів" (Total Cycle Time Reduction), яке по суті являє реінжиніринг бізнес-процесів, сфокусований в першу чергу на скороченні тривалості бізнес-процесів.

Метод аналізу ланцюжка цінностей. Ланцюжок цінностей – це безліч закінчених стикованих дій (операцій), які в сукупності створюють деяку продукцію, що має споживчу цінність для клієнта.

Мета методу аналізу ланцюжка цінностей полягає в послідовному аналізі кроків процесу з метою виявлення робіт, які не додають цінності продукту. Прикладами таких робіт у виробничому процесі є непотрібні перевірки параметрів, переміщення продукту, складання замовлень і звітів, різноманітні повторної перевірки і переробки, вимушені очікування і простої в процесі роботи.

Автоматизація бізнес-процесів (business process automation, BPA) припускає застосування інформаційних технологій, для того щоб зробити існуючі бізнес-процеси більш ефективними. В ході автоматизації самі бізнес-процеси зазвичай залишаються незмінними, але сам потік робіт, що виконується раніше "вручну", частково або повністю переноситься на комп'ютер.

Скорочення витрат (cost reduction) – метод, в ході якого прийнято аналізувати існуючі процеси з метою виявлення усунених видів витрат, не змінюючи при цьому традиційний спосіб виконання бізнес-процесу.

Безперервне поліпшення процесів (continuous process improvement, CPI) припускає довгострокове і поступове підвищення ефективності бізнес-процесів.

ФВА бізнес-процесів – використання методів функціонально-вартісного аналізу для вдосконалення бізнес-процесів.

Методи оптимізації бізнес-процесів можна поділити на три групи:

1) Формалізовані універсально-принципові методи, які засновані на застосуванні успішного досвіду і формалізованих принципів для побудови ефективних бізнес-процесів. Дані методи є універсальними і вони підходять для оптимізації будь-яких бізнес-процесів для будь-якого бізнесу і практично не залежать від його специфіки.

2) Методи засновані на вивченні, аналізі і подальшому копіюванні елементів процесів успішних компаній, що займаються схожими видами діяльності. Претендентами на вивчення і копіювання їх успішного досвіду в першу чергу є лідери-конкуренти. Практика показала, що останнім часом багато підприємств ефективно впровадили у себе технологічні ноу-хау, запозичивши їх у підприємств, що працюють в інших галузях бізнесу.

3) Методи групової роботи. Ця група методів об'єднала різні технології роботи в команді: метод мозкового штурму, метод групового рішення задач тощо. їх використання дозволяє розробити нові ефективні рішення, раніше нікому не відомі, що дозволяє підприємству бути лідером за технологіями, що використовуються.

Оптимізація бізнес-процесів підприємства має наступні переваги:

- скорочення витрат, тривалості та кількості помилок у кожному з проаналізованих процесів;
- формування у працівників підприємства та керівників чіткого розуміння того як, коли, хто, та що необхідно робити для досягнення поставлених цілей;
- інтегрування зі стратегією підприємства та ключовими показниками її ефективності;
- можливість підготовки до успішного, продуманого та ефективного впровадження інформаційних технологій;
- можливість підготуватись до ефективного та обґрунтованого організаційного редизайну;
- зростання керованості підприємства;
- покращення взаємодії між працівниками та підрозділами підприємства;
- зростання інвестиційної привабливості.

Для даного проекту в якості прикладу було розроблено CRM - систему як інструмент автоматизації бізнес-процесів та аналітики ефективності продажів.

1.2. Бізнес-процеси в CRM

CRM-системи (Customer Relationship Management, або управління відносинами з клієнтами) призначені для оптимізації бізнес-процесів із взаємодії з потенційними та наявними клієнтами.

CRM-системи не призначені для того, щоб замінити працівників у справі управління комунікацією з клієнтами їх основна мета – поліпшити відносини з наявними замовниками, довести потенційних клієнтів до стадії оплати та підвищити ефективність роботи співробітників.

Основною проблемою для впровадження CRM на більшості підприємств була висока вартість коробкових рішень і необхідність їхнього розгортання на

базі власної інфраструктури. Із появою хмарних рішень така проблема, навіть для невеликої компанії, втратила свою актуальність.

Використання таких технологічних рішень у роботі вигідне як для малого, так і для великого бізнесу.

В умовах сучасного висококонкурентного ринку компанії, які управляють відносинами з клієнтами, мають набагато вищі шанси на успіх, ніж ті, які цього не роблять.

Професійно реалізовані CRM-системи забезпечують безліч переваг для відділу продажів, маркетингу, служби підтримки і не тільки.

Основні з них:

Приріст продуктивності.

Багато ручних процесів автоматизуються, що істотно підвищує віддачу від роботи працівників та ефективність роботи компанії в цілому. Важливий момент у контексті – можливість відійти від необхідності використання цілого ряду окремих інструментів, наприклад Google Docs, систем планування завдань, чату та інших окремих сервісів. У багатьох CRM все це інтегровано в єдину систему.

Крім того, взаємодія між відділами стає більш цілісною, а керівник за потреби завжди може оцінити загальну картину роботи. У B2B-бізнесі (Business to business), наприклад, життєвий цикл клієнта, як правило, занадто складний, щоб його міг оцінювати тільки один співробітник.

Автоматизація.

У кожної компанії є повторювані завдання нижчого рівня, наприклад, відправлення звітів за підсумками місяця, на які може витрачатись багато часу. Деякі CRM дозволяють налаштувати їхнє виконання в автоматичному режимі, позбавивши співробітників одноманітних завдань.

Ще одна важлива можливість – налаштування повідомлень, наприклад, нагадувань про дедлайни або необхідність зробити певну дію.

Збереження даних.

Уся інформація конкретного клієнта зберігається в одному місці, і за потреби доступ до неї легко можна надати іншій людині. Завдання в автоматичному або ручному режимі розподіляються між працівниками, і кожен розуміє, чим він повинен займатися зараз.

“Хмарні” платформи забезпечують безпечне та організоване зберігання інформації про клієнтів. Усі дані не тільки зберігаються в одному місці, але і доступні тільки для авторизованих користувачів.

Ефективне планування і відстеження.

Виконання багатьох завдань йде не за планом, коли люди роблять все вручну. Із впровадженням CRM-системи можна планувати та контролювати завдання простіше і прозоріше.

Якщо на якомусь етапі відбувається провал або з’являється проблема, яка заважає подальшій реалізації проекту, легко визначити, що і де пішло не так, хто за це відповідальний і як зробити так, щоб проблема не повторилася.

Користь для маркетингу.

Стає можливим чітке відстеження кожного з етапів воронки продажів і зрозуміло, як та чи інша маркетингова активність впливає на потік замовлень. Причому можливостей застосування саме для інтернет-маркетингу більш ніж достатньо.

Зокрема, CRM-система дозволяє отримати правильне уявлення про найприбутковіші групи клієнтів, а потім на основі цих даних грамотно націлити. Таким чином, можна правильно оптимізувати використання доступного бюджету.

Інтеграція з іншими продуктами.

Деякі розробники CRM-систем пішли іншим шляхом, реалізувавши інтеграцію з іншими сервісами, наприклад, для бухгалтерського обліку, управління проектами, e-mail-розсилок і т. д.

Це дозволяє зробити організацію бізнес-процесів максимально безшовною і виконувати більшу частину роботи на базі єдиної платформи.

Поліпшення відносин із клієнтами.

Зрештою, одна з основних переваг CRM-систем, яку вони забезпечують компанії, що їх використовує, – поліпшення загальної якості обслуговування клієнтів. Зростає ймовірність того, що поточні замовники порекомендують вас своїм знайомим як відповідального і надійного виконавця.

Збір аналітичних даних і побудова звітів.

Правильному налаштуванні CRM-системи, кожна комунікація з клієнтом фіксується системою. Таким чином CPM - це збирач великого пласта цінної аналітичної інформації.

Можна швидко отримати такі звіти:

- Звіт про надходження нових лідах з сайту за обраний період і за якими товарів або послуг вони були.
- Звіт про продажі в розрізі місяця або іншого періоду, в розрізі клієнта, в розрізі конкретного менеджера, в розрізі кожного продукту.
- Звіт про ефективність продажів.
- Воронка продажів - це ключовий звіт, який дозволить визначити на якому етапі взаємодії відбувається найбільша «втрата» клієнтів.

Всі ці звіти дозволять приймати виважені управлінські рішення щодо розвитку системи продажів у бізнесі.

Побудова бізнес-процесів і вбудовування їх в CRM дає можливість оптимізувати безліч завдань, усунути неефективні ітерації в процесах, уникнути додаткових витрат. Таким чином виходить централізована система, націлена на активне управління, а не на пасивне фіксування фактів. Наприклад, впровадження CRM при правильно побудованому бізнес-процесі запустить такі механізми, які не дадуть забути про дзвінок клієнта або про необхідність виставлення рахунку.

Стандартизація операційної діяльності сприяє усуненню людського фактора і допомагає формувати більш прозорі моделі роботи. Така прозорість допоможе не тільки "тут і зараз" в організації конкретного бізнес-процесу, а й вплине на більш масштабні аспекти діяльності організації і на її культуру. Наприклад, позитивний вплив може бути надано на HR-політику компанії - починаючи від оновлених портретів кандидатів закінчуючи кількістю штатних співробітників і новими бізнес-ролями.

Моделювання, оптимізація та управління бізнес-процесами CRM затребувані не тільки гігантам ринку, вони необхідні компаніям будь-якого масштабу і всіх галузей. Процеси є у всіх організаціях - змінюватися може довжина ланцюжка, кількість бізнес-ролей, кількість учасників процесу, обсяг і періодичність виконання того чи іншого завдання. І в невеликій компанії може загубитися маленька невиконана завдання, яке потягне за собою збитки.

Це сприятлива зміна для співробітників: чіткий розподіл обов'язків, можливість реального планування, мінімізація і авралів, в цілому - чітке розуміння бізнес-процесу і, як наслідок, краще розуміння власної роботи, місії компанії.

Отже, моделювання бізнес-процесів необхідно організації, якщо вона бачить необхідність в:

- поліпшенні якості обслуговування;
- мінімізації витрат;
- усунення людського фактора;
- поділі зон відповідальності;
- регламентування відносин між учасниками процесу;
- єдиної IT-структурі;
- ефективному управлінні ресурсами;
- постійному контролі й прозорості діяльності співробітників;
- постійному вдосконаленні діяльності.

При використанні таких систем підвищується точність та спрощується робота за їхньою сегментацією, потреби визначаються і заносяться в базу даних, завдання реалізуються вчасно і точно відслідковуються. Усе це приводить до зменшення термінів укладання договорів, зростання прибутку і високого рівня утримання клієнтів через те, що зростає їхня задоволеність.

На старті, втім, можна обійтись і без таких інструментів – достатньо буде й адмінки інтернет-магазину. Можливість переходу на CRM варто розглядати при зростанні відвідуваності до рівня понад 1000 користувачів на день, великій кількості продажів і повторних замовленнях.

CRM-система для даного проекту повинна реалізовувати огляд ефективності продажів та показувати статистику. Така система ідеально підійде для інтернет-магазинів.

Висновок до розділу 1

Інформаційні технології (ІТ) надають інструменти вдосконалення бізнес-процесів і грають чотири найважливіші ролі в проектах їх поліпшення (ІТ як основа для побудови нових процесів; ІТ як засіб управління проектами; ІТ як засіб електронних комунікації; ІТ як засіб побудови корпоративних інформаційних систем (КІС).

Для реалізації проектного веб-додатку на MEAN stack потрібні такі інструменти, як:

- Середовище розробки Node.js;
- СУБД MongoDB, яка допоможе реалізувати логіку серверної Backend частини проекту
- Фреймворк-платформа Angular, який потрібен для реалізації клієнтської частини проекту
- Postman, для ручного тестування працездатності API
- Текстовий редактор Atom для реалізації коду.

РОЗДІЛ 2. ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ ДОДАТКУ

2.1.Текстовий редактор Atom (Windows, Linux, Mac)

Текстовий редактор має основну роль в будь-якому робочому просторі розробника. Код пишеться, налагоджували і виконується за допомогою текстового редактора.

Вибір ідеального редактора для роботи може бути складним завданням, яке включає в себе: тестування, особисті уподобання і остаточне рішення.

Для реалізації даного проекту був обраний редактор Atom, який розробила команда розробників GitHub, Свій продукт вони позиціонують, як текстовий редактор 21 століття.

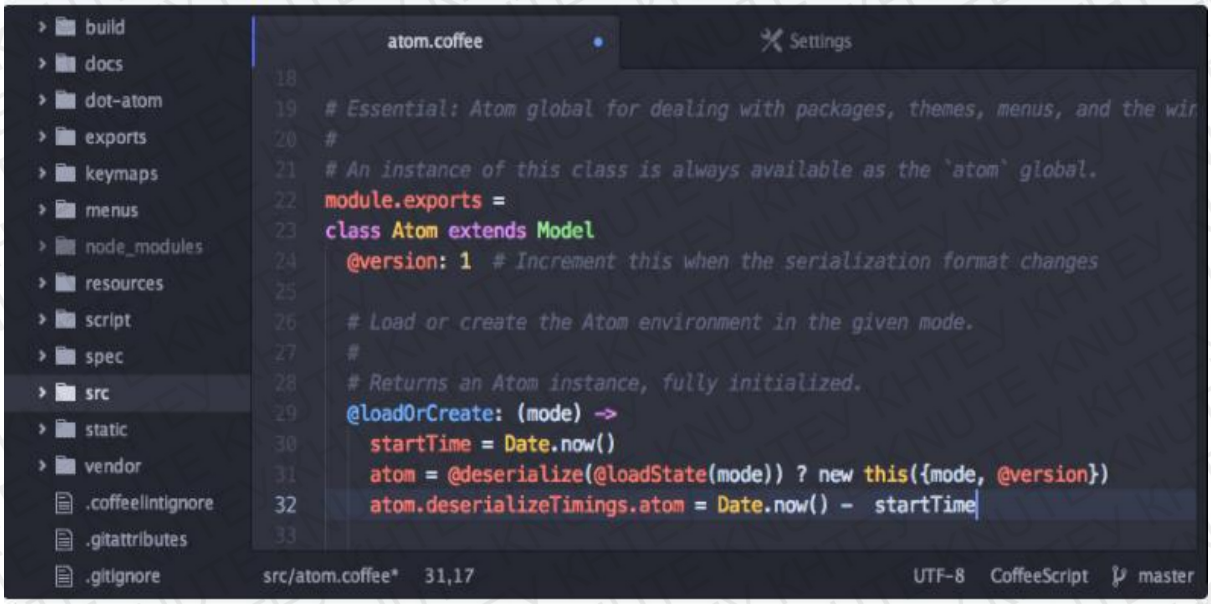

Рис 2.1. Інтерфейс редактору Atom

Atom - продукт безкоштовний, з відкритим вихідним кодом. Він зібраний з 50 модулів і написаний на C ++, JavaScript, CSS і HTML. Існує можливість додавати власні модулі у відкритий репозиторій, щоб ними могли користуватися інші.

У нього є все ті ж базові функції, що і у Sublime Text, начебто швидкого пошуку нечітких збігів в проектах і файлах, наявності міні карти, а також використання фрагментів. Він підтримує згаданий раніше Emmet, Autoprefixer, автоформатирование коду за допомогою atom-beautify, Livereload.

Головна особливість Atom - багаті можливості по налаштуванню. Редактор зручно налаштовувати.

Спочатку в нього вбудовані файл-менеджер, просунуті функції пошуку і заміни, різноманітні курсори, опції згортання коду, ясний інтерфейс, можливість імпорту правил і тем з TextMate.

Десктопний додаток Atom має повний доступ до файлової системи, природні для операційної системи меню та панелі команд. При цьому воно ідеально пристосоване для веб-програмування: можна додавати власні функції для редагування CSS, HTML і JavaScript. Потрібно відзначити також інтеграцію з Node.js, включаючи запуск веб-сервера прямо з редактора. Архітектура програми проста і зрозуміла кожному: можна замінити будь-який пакет своїм власним і закачати його в центральний репозиторій, щоб їм скористався кожен охочий.

Якщо шукаєте безкоштовний, редактор з відкритим вихідним кодом, то Atom ідеально підійде. Він дуже мобільний і доступний для всіх трьох основних ОС.

Переваги:

- Atom є редактором з відкритим вихідним кодом який вільний у використанні;
- Кросплатформеність OS X, Windows і Linux;
- Розумне автодоповнення;
- Браузер файлів;
- Пошук і заміна з багатьох файлів;
- Простий у використанні навіть для новачка.

Недоліки:

- Не може працювати з великими файлами і має тенденцію до збоїв при завантаженні файлів вище 10 МБ;
- Використовує багато пам'яті.

Можна сказати точно, що Atom - гідний редактор для зручної роботи програміста з безліччю приємних доповнень і який ідеально підходить для розробки веб-додатків.

Наступним кроком була обрана СУБД для даного проекту. При введенні в експлуатацію програмних продуктів важливо передбачити правильну роботу бази даних.

2.2. СУБД MongoDB та його сервісна версія Atlas.

MongoDB - документоорієнтована СУБД з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB призначена для додатків, які використовують як структуровані, так і неструктуровані дані. Ядро є дуже гнучким і працює при підключенні бази даних до додатків через драйвери MongoDB. Існує широкий вибір доступних драйверів, тому легко знайти драйвер, який буде працювати з необхідним мовою програмування.

Вона класифікується як NoSQL і використовує BSON (бінарний JSON). Масштабується з коробки, написана на мові C ++ і підтримує синтаксис JavaScript. Підтримка SQL відсутній.

У MongoDB є драйвери для багатьох популярних мов програмування (Ci, C ++, C #, Go, Java, JavaScript, Perl, PHP, Python, Ruby і ін.). Також є неофіційні і підтримувані спільнотою драйвери для інших мов програмування.

Будь-яка реляційна БД має стандартну схему, яка показує кількість таблиць і зв'язку між ними. У MongoDB такої схеми з зв'язку між таблицями немає.

Нижче наведені основні переваги MongoDB:

- відсутність схеми. Дана БД заснована на колекціях різних документів. Кількість полів, зміст і розмір цих документів може відрізнятися. Тобто різні сутності не повинні бути ідентичні за структурою;
- вкрай зрозуміла структура кожного об'єкта;
- легко масштабується;
- для зберігання використовуваних в даний момент даних використовується внутрішня пам'ять, що дозволяє отримувати більш швидкий доступ;
- дані зберігаються у вигляді JSON документів;
- MongoDB підтримує динамічні запити документів (document-based query);
- відсутність складних JOIN запитів;
- швидкість і простота у використанні;
- движок підтримує json і інші традиційні документи NoSQL;
- дані будь-якої структури можуть бути збережені / прочитані швидко і легко.

Можна сказати, що MongoDB є непоганим рішенням, якщо розробник має справу с Big Data.

Для реалізації даного проекту було використано DaaS-версію (database-as-a-service) однойменної платформи СУБД (MongoDB Atlas), яка дозволяє сфокусуватися на своїх додатках.

На етапі створення кластеру в СУБД MongoDB Atlas треба обрати хмарний хостинг.

Хмарний хостинг - це обчислювальні потужності з можливістю гнучкого розширення і розподілу навантаження.

Хмарний хостинг (Amazon Web Services).

Нове рішення, MongoDB Atlas, позиціонується компанією як найбільш ефективний спосіб роботи з MongoDB в хмарі. Хостинг для цього сервісу надає Amazon Web Services, Microsoft Azure і Google Cloud.

При настройці та створенні кластеру в MongoDB Atlas в якості хмарного хостингу було обрано AWS (Amazon Web Services).

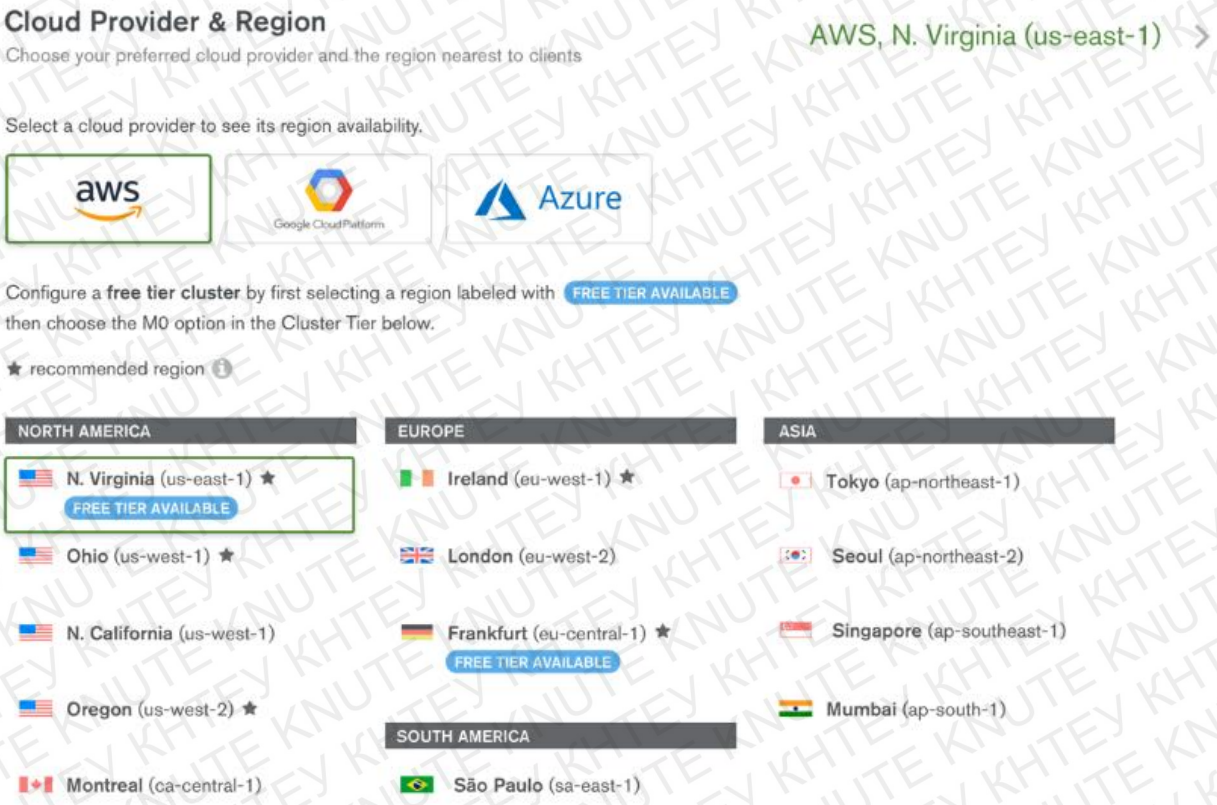

Рис. 2.2. Настройка кластера в MongoDB Atlas

AWS являє собою конструктор, з якого можна зібрати як завгодно складну, розподілену мережеву інфраструктуру. AWS дуже зручний при необхідності розгортати багато однакових інстансів. Безкоштовний пакет AWS Free Usage Tier включає в себе:

- EC2 (інстанси - віртуальні машини ОС)
- 750 годин використання віртуальної машини з Linux або Windows Server (613 Мб ОЗУ, 32 бітна або 64-бітна платформа) - достатня кількість годин для роботи інстанси щомісяця
- 750 годин Elastic Load Balancer плюс 15 Гб обробки трафіку
- 30 Гб Amazon Elastic Block Storage, плюс 2 мільйони операцій введення / виводу і 1 Гб для зберігання снепшот.
- 15 Гб трафіку
- S3 (файлове сховище)
- 5 Гб Amazon S3 стандартного сховища, 20000 Get запитів і 2000 Put запитів
- Relational Database Service (служба реляційних баз даних, RDS)
- 750 годин сервісу для запуску MySQL, Oracle BYOL або SQL Server
- 20 Гб сховище бази даних
- 10 мільйонів операцій введення / виводу
- 20 Гб сховище для бекапів для автоматичного резервного копіювання вашої бази даних і можливістю створити снепшот бази даних.

За заявою Еліота Горовиця (Eliot Horowitz), одного із засновників і технологічного директора MongoDB, Atlas максимально спрощує роботу з MongoDB, будь то одиночна репліка або шаржований кластер з об'ємом хостингу 100 ТБ.

Сервіс в значній мірі заснований на самообслуговуванні, він пропонує вертикальне масштабування з екземплярами різного розміру або горизонтальне - автоматичним - без необхідності зупинки додатків.

Тут немає серверів, які потрібно налаштовувати і конфігурувати, немає необхідності організовувати резервне копіювання, моніторинг операцій або відстеження вразливостей, Якщо сервер обвалиться вночі, система сама

займеться ремонтом, якщо щось не вдасться виправити автоматично, це теж турбота команди цілодобової підтримки.

MongoDB Atlas частково побудований на вже існуючому коді, деякі компоненти запозичені з інших хмарних пропозицій компанії, а модуль самообслуговування, Automation, більше двох років застосовується для створення кластерів, установки оновлень і зміни конфігурації тисяч інстальованих продуктів MongoDB.

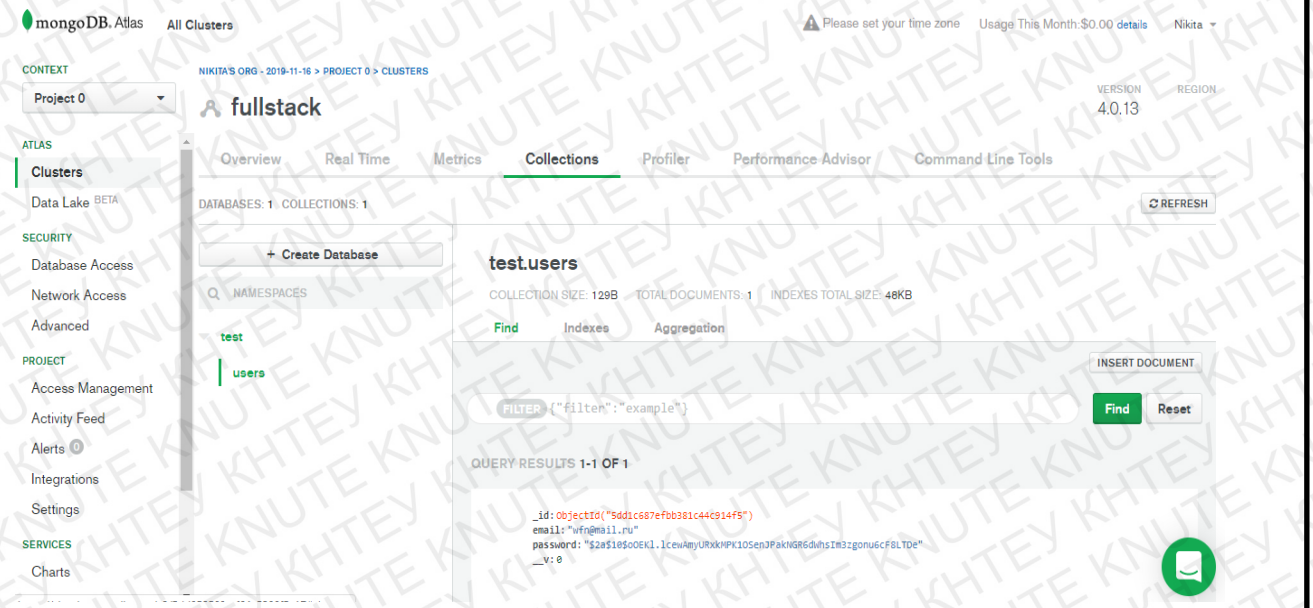

Рис. 2.3. Кластер проекту в MongoDB Atlas

Оскільки розробка додатку почалася с серверної Backend частини, то потрібно було тестувати працездатність та повноцінність API запитів проектного додатку. В цьому допомагає функціонал програми Postman.

2.3 Postman, інструмент тестування API

Postman - це середовище розробки API, яка допомагає людям створювати, тестувати, документувати, контролювати і публікувати документацію для своїх API.

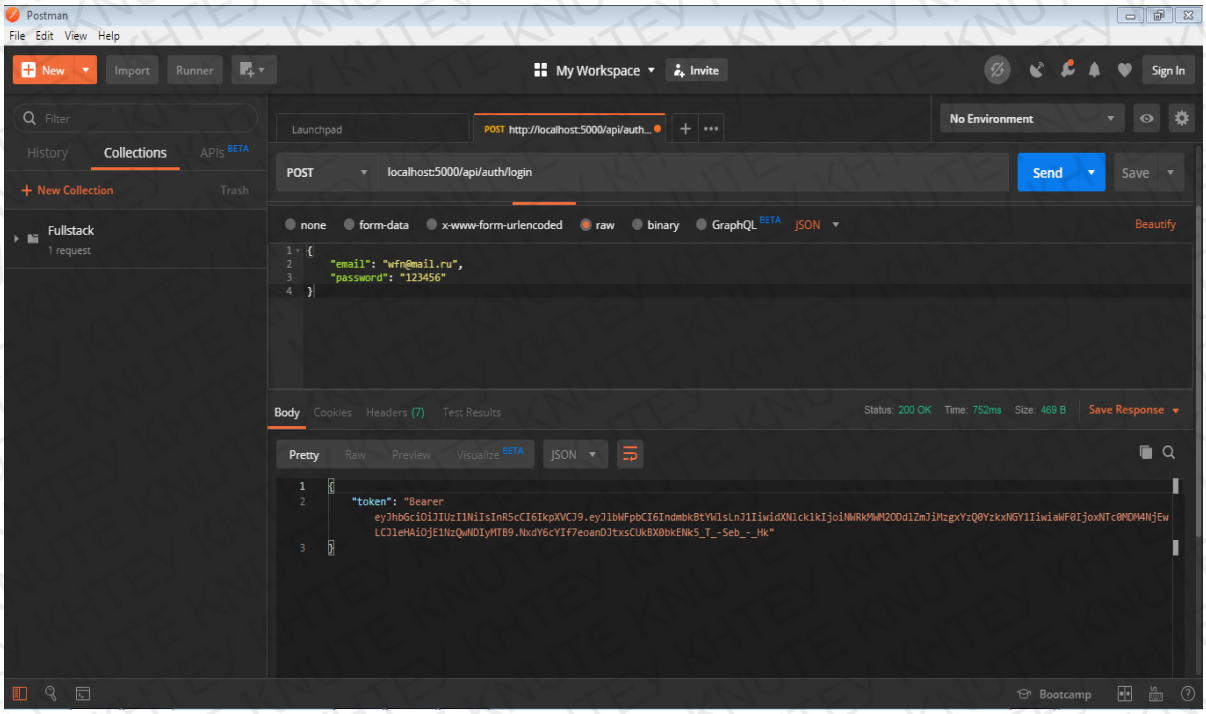

Рис. 2.4. Інтерфейс Postman.

Основні особливості Postman:

- Відправлення запитів (з підтримкою різних схем аутентифікації, Cookies, сертифікатів, заголовків, параметрів запиту, тіла запиту і SOAP з WSDL / без WSDL) та налагодження і збереження відповідей.
- Організація API-інтерфейсів в групи, звані колекціями.
- Спільне використання колекцій з вашою командою.
- Написання тестів. Сценарії тестування можуть запускати попередній запит після отримання відповіді і можуть мати концепції циклу і розгалуження.
- Поняття змінних для установки і отримання. Так само як і змінні середовища глобальних змінних.
- Автоматизація тестування з використанням складальних циклів. Колекції також можуть бути експортовані і запущені в командному

рядку, використовуючи Newman як частина вашого процесу складання.

- Можна планувати тести, які будуть запускатися автоматичним способом, за допомогою моніторів Postman.
- Спосіб автоматичної генерації і налаштування документації API безпосередньо з ваших колекцій. Він може бути приватним, спільно з вашою командою, загальнодоступним і також може бути налаштований у вашому призначеному для користувача домені.
- Імітувати бекенда за допомогою Mock Servers.
- Інтеграція з такими сервісами, як Slack, Github, Bitbucket, Datadog, Keen.io, Microsoft Flow і ряд інших.
- API, який дозволяє вам використовувати дані Postman як частина ваших безперервних процесів інтеграції та доставки.
- Спосіб імпорту існуючих API з Swagger, RAML, cURL і декількох інших інструментів.
- Автоматичне створення фрагментів коду на різних мовах з ваших API.
- Перехоплення запитів.
- Інтерактивна історія всіх ваших запитів.
- Контроль доступу для вашої команди і єдиного входу (SSO).
- Основні компоненти з відкритим вихідним кодом.

Postman спрощує та прискорює процес тестування, тому в ручному тестуванні REST API для Backend частини проекту він був необхідним.

Для Frontend частини проекту було обрано фреймоврк Angular, оскілки функціонал цього фреймворку ідеально підходить для реалізації веб-додатків.

2.4. Angular (Angular 8)

Angular 8 поєднує в собі декларативні шаблони, ін'єкцію залежностей, комплексний end-to-end інструментарій та вже імплементовані best practice для вирішення будь-якої складності завдань. Angular 8 дозволяє створювати не тільки для веб-додатки але так само мобільні і десктопні програми.

Angular 8, повністю переробленому продовження JavaScript фреймворку AngularJS. Перероблений з нуля фахівцями Google, Angular 2+ надає розширені можливості для створення Single-page application, такі як, декларативні шаблони, двосторонній дата Біндінг, підтримка TypeScript, і впровадження залежностей. Замість контролерів, характерних для архітектури MVC, Angular 2+ тепер використовує компоненти. Це оновлення, підходить як для мобільних так і для веб розробників.

Зокрема, була представлена цікава концепція прив'язки даних, по якій уявлення автоматично оновлюється при будь-якій зміні моделі і навпаки. Поверх цього представили ідею директив, які дозволяють створювати власні HTML теги і оживляти їх через JS.

Переваги Angular:

- Angular підтримується на різних платформах (веб, мобільні пристрої, нативний десктоп), він потужний, сучасний, у нього відмінна екосистема
- Angular представляє не тільки інструменти, але і шаблони дизайну для створення обслуговується проекту. При правильному створенні Angular додатки у вас не буде плутанини класів і методів, які складно правити і ще складніше тестувати. Код зручно структурований, можна швидко зрозуміти, що до чого.
- Це JS, але краще. Angular побудований на TypeScript, який, в свою чергу, покладається на ES6. Вам не потрібно вчити повністю нову мову, і ви отримуєте функції типу статичної типізації, інтерфейсів, класів, простору імен, декоратори і т.д.

- Не потрібно винаходити велосипед. У Angular вже є багато інструментів для створення програми. Завдяки директивам, HTML елементи можуть вести себе динамічно. Ви можете підсилити форми за допомогою FormControl і представити різні правила валідації. Можна легко посилати асинхронні HTTP запити різних типів. Можна без праці налаштувати маршрутизацію. У Angular є ще багато функцій!
- Компоненти роз'єднані. Angular намагався прибрати жорстку зв'язок між різними компонентами програми. Ін'єкція проходить подібно NodeJS, що дозволяє легко замінювати компоненти.
- Всі маніпуляції з DOM проходять там, де повинні. У Angular уявлення і логіка програми не пов'язані, що сильно очищає і спрощує розмітку.
- Тестування в центрі уваги. Angular ретельно протестований і підтримує юніт тести і наскрізне тестування за допомогою інструментів типу Jasmine і Protractor.
- Angular підготовлений до мобільних пристроїв і десктопу - один фреймворк під безліч платформ.
- Angular активно обслуговується і має велике співтовариство і екосистему. За фреймворку можна знайти багато матеріалом і корисних сторонніх інструментів.

Web workers.

У Angular 8 реалізовані Web worker'и. Оскільки JavaScript є однопоточним за визначенням. Через це трудомісткі завдання, такі як запит даних, зазвичай виконуються асинхронно і це не допомагає в складних розрахунках. Web worker'и - це скрипти, які запускаються браузером в окремому потоці. Зв'язок з потоком на вкладці браузера здійснюється за допомогою повідомлень.

Хоча web worker'и не мають відношення до Angular як такого, вони повинні бути прийняті до уваги при складанні. Метою є надання одного пакета для кожного web worker.

Диференційне завантаження.

Це завдання виконане новим CLI. До сих пір було прийнято компілювати програми в старий добрий ES 5, так як ця версія працює майже всюди. Це означає, що і IE11 і веб-сканер пошукової системи Google можуть виконувати цей код.

Однак новий ES 2015 і його наступні версії більш ефективні: вони дозволяють створювати більш компактні пакети і браузер може також інтерпретувати їх більш ефективно. Оскільки раніше було прийнято відкочуватися до ES 5 як найменшого спільного знаменника, сучасні браузери, на жаль, не могли використовувати переваги нової версії мови.

Тепер з цим покінчено: починаючи з 8 версії, CLI має функцію, яка називається диференціальної завантаженням. Ідея полягає в тому, щоб надати дві групи пакетів: один заснований на ECMAScript 5 і призначений для старих браузерів, інший заснований на новій версії ECMAScript, наприклад ECMAScript 2015 року, і надає сучасним браузерам раніше згадані переваги.

Можна сказати, що Angular не просто фреймворк, а платформа, яка дозволяє розробникам будувати додатки для інтернету, мобільних пристроїв і робочого столу.

Висновок до розділу 2

В даному розділі були розглянуті основні інструменти які потрібні для реалізації заданого проекту. Для більш комфортної FullStack JavaScript-розробки веб-додатку практично будь-якої складності ці технології (Node.js, Express.js, MongoDB і Angular) найліпші в цій галузі.

Кожний розглянутий інструмент потрібен для оптимізації та спрощення процесу створення проектного веб-додатку.

РОЗДІЛ 3. ПРОЦЕС СТВОРЕННЯ ВЕБ-ДОДАТКУ

3.1. Середовище Node.js та MEAN STACK розробка

База даних MongoDB, фреймворки Express і AngularJS і технологія Node.js разом утворюють стек **MEAN** - потужну платформу, на всіх рівнях якої застосовується всього одна мова: JavaScript. Стек MEAN привабливий для розробників і бізнесу завдяки простоті і економічності, а кінцеві користувачі люблять MEAN-додатки за їх швидкість і чуйність.

JavaScript досяг зрілості. Завдяки йому тепер можна створити веб-додаток від початку до кінця за допомогою всього однієї мови програмування. Стек MEAN включає в себе кращі в своєму класі технології в цій галузі. Як БД ви отримуєте MongoDB, як серверного фреймворка веб-додатків - Express, в якості клієнтського фреймворка - AngularJS, а в якості серверної платформи - Node.

Node.js - це середовище виконання JavaScript на стороні сервера, яка використовується для побудови продуктивних, швидких, масштабованих мережних додатків. Побудована на JavaScript-движку V8, розробленому компанією Google.

Програмна платформа Node.js дозволяє використовувати JavaScript для написання коду як на стороні клієнта (Frontend), так і на стороні сервера (Backend).

Node.js є кроссплатформеним середовищем з відкритим вихідним кодом для розробки серверних і мережевих додатків. Додатки Node.js написані на JavaScript і можуть виконуватися в середовищі виконання Node.js на ОС X, Microsoft Windows і Linux.

Node.js використовує керовану подіями, не блокуючу модель введення-виведення, яка робить її простою і ефективною для додатків з інтенсивним використанням даних в реальному часі, що працюють через розподілені пристрої.

NPM: Менеджер пакетів Node

Обговорюючи Node.js, просто необхідно згадати існуючу в ньому вбудовану підтримку управління пакетами, для якої застосовується інструмент NPM, за замовчуванням присутній в будь-яку установку Node.js. Ідея модулів NPM багато в чому схожа з Ruby Gems: це набір загальнодоступних компонентів для багаторазового використання, які легко встановити через онлайновий репозиторій; для них підтримується управління версіями і залежностями.

Повний список упакованих модулів знаходиться на сайті NPM, а також доступний за допомогою інструменту NPM CLI, який автоматично встановлюється разом з Node.js. Екосистема модулів абсолютно відкрита, будь-хто може опублікувати в ній власний модуль, який з'явиться в списку сховища NPM.

Деякі з найбільш популярних сучасних NPM-модулів:

- Connect - це розширюваний фреймворк, який працює з Node.js як HTTP-сервера, що надає колекцію високопродуктивних «плагінів», відомих під загальною назвою «сполучна ПО» (middleware); служить основою для Express.
- Express - це мінімалістичний і гнучкий веб-фреймворк для додатків Node.js, що надає великий набір функцій для мобільних і веб-додатків.
- Socket.io і sockjs - серверна частина двох найбільш поширених сьогодні веб-сокетних компонентів.
- Jade - один з популярних шаблонизаторів, написаний в дусі HAML, за замовчуванням використовується в Express.js.

- Mongo і mongoos - обгортки MongoDB, що надають API для об'єктних баз даних MongoDB в Node.js.
- Redis - клієнтська бібліотека Redis.
- Coffee-script - компілятор CoffeeScript, що дозволяє розробникам писати програми з Node.js за допомогою Coffee.
- Underscore (lodash, lazy) - найпопулярніша допоміжна бібліотека JavaScript, упакована для використання з Node.js, а також дві аналогічні їй бібліотеки, що забезпечують підвищену продуктивність, оскільки реалізовані трохи інакше.
- Forever - найпоширеніша утиліта, яка забезпечує безперебійне виконання сценарію на заданому вузлі. Підтримує працездатність процесу Node.js при будь-яких несподіваних збоях.

Середовище Node.js в комбінації MEAN stack – це ідеальний вибір для швидкої FullStack-розробки продуктивних, швидких, масштабованих мережних додатків з простою архітектурою.

3.2. Архітектура додатку

Проект буде складатися з 3 сутностей (Сервер, MongoDB, Клієнт).

Сервер – Backend нашого додатку в якому буде реалізована серверна частина, для цього буде використовуватися функціонал таких фреймворків як: Express (написання серверу), Mongoose (взаємодія БД та серверу), Passport (створення авторизації);

MongoDB – БД проектного додатку;

Клієнт – Frontend частина додатку, яка буде написана за допомогою фреймворка Angular.

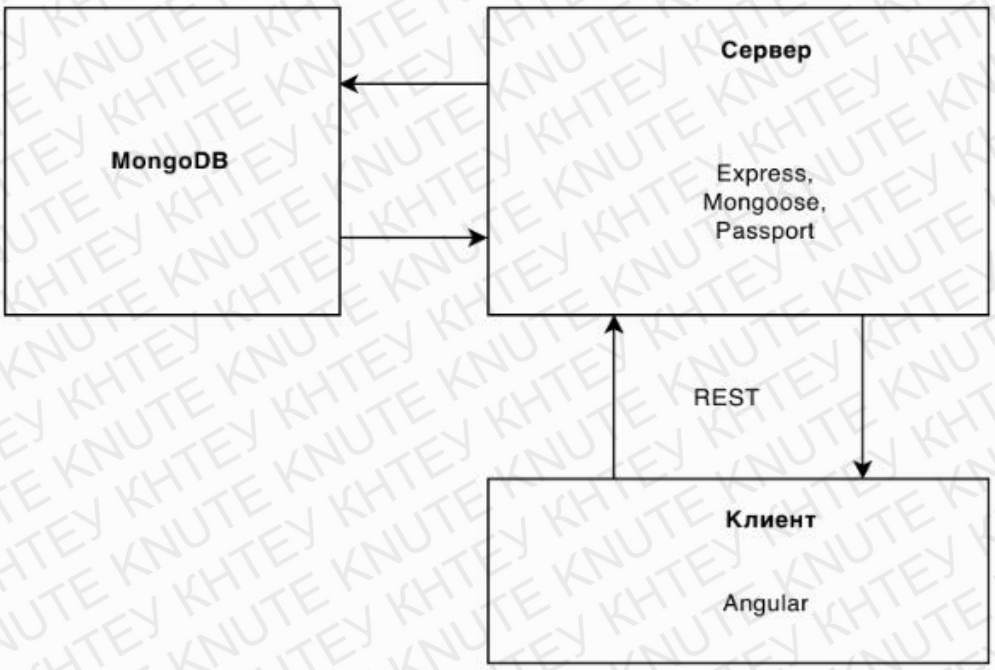

Рис. 3.1. Архітектура додатку

Створивши архітектуру веб-додатку треба визначитися з сутностями, з яких буде складатися веб-додаток.

3.3. Моделі веб-додатку (фреймворк Mongoose)

Веб-додаток являє собою CRM-систему для малого бізнесу, яка буде давати аналітику по замовленням і надаватиме аналіз ефективності продажів та буде складатися з таких сутностей:

Category (name – назва категорії, imageSrc – зображення категорії, user – інформація про користувача який створив дану категорію);

Position (name – назва позиції, cost – ціна позиції, category – id категорії до якої відноситься позиція, user - інформація про користувача який створив дану позицію);

User (email, password);

Order (date – дата створення замовлення, order – номер замовлення, list масив інформації (name – назва позиції, quantity – кількість куплених позицій, cost – ціна позиції), user – користувач, який створив замовлення).

Mongoose - це ODM (* Object Document Mapper - об'єктно-документний відображувач). Це означає, що Mongoose дозволяє вам визначати об'єкти зі строго типизованою схемою, яка відповідає документу MongoDB.

Mongoose надає величезний набір функціональних можливостей для створення і роботи зі схемами. На даний момент Mongoose містить вісім SchemaTypes (* типи даних схеми), які може мати властивість, що зберігається в MongoDB. Ці типи наступні:

- String;
- Number;
- Date;
- Buffer;
- Boolean;
- Mixed;
- ObjectId (* унікальний ідентифікатор об'єкта, первинний ключ, _id);
- Array.

Для кожного типу даних можна:

- задати значення за замовчуванням;
- задати для користувача функцію перевірки даних;
- вказати, що поле є обов`язковим;
- задати get-функцію (геттер), яка дозволяє вам проводити маніпуляції з даними до їх повернення у вигляді об'єкта;
- задати set-функцію (* сетер), яка дозволяє вам проводити маніпуляції з даними до їх збереження в базу даних;
- визначити індекси для більш швидкого отримання даних.

Крім цих загальних можливостей для деяких типів даних також можна налаштувати особливості збереження і отримання даних з бази даних. Наприклад, для типу даних String можна вказати наступні додаткові опції:

- конвертація даних в нижній регістр;
- конвертація даних в верхній регістр;
- обрізка даних перед збереженням;
- визначення регулярного виразу, яке дозволяє в процесі перевірки даних обмежити дозволені для збереження варіанти данни;
- визначення переліку, який дозволяє встановити список допустимих рядків.

Для властивостей типу Number і Date можна задати мінімально і максимально допустиме значення.

Більшість з восьми допустимих типів даних повинні бути вам добре знайомі. Однак, деякі (Buffer, Mixed, ObjectId і Array) можуть викликати труднощі.

Тип даних Buffer дозволяє вам зберігати двійкові дані. Типовим прикладом довічних даних може послужити зображення або закодований файл, наприклад, документ в PDF-форматі (* формат переноситься документа).

Тип даних Mixed використовується для перетворення властивості в "нерозбірливе" поле (поле, в якому допустимі дані будь-якого типу). Подібно до того, як багато розробників використовують MongoDB для різних цілей, в цьому полі можна зберігати дані різного типу, оскільки відсутня певна структура. Цей тип даних треба використовувати з обережністю, оскільки він обмежує можливості, що надаються Mongoose, наприклад, перевірку даних і відстеження змін сутності для автоматичного оновлення властивості при збереженні.

Тип даних ObjectId використовується зазвичай для визначення посилання на інший документ у базі даних. Наприклад, в базі даних є колекція книг і авторів, документ книги міг би містити властивість ObjectId, що посилається на певного автора документа.

Тип даних Array дозволяє зберігати JavaScript-подібні масиви. Завдяки цьому типу даних ви можете виконувати над даними типові JavaScript операції над масивами, наприклад, push, pop, shift, slice і т.д.

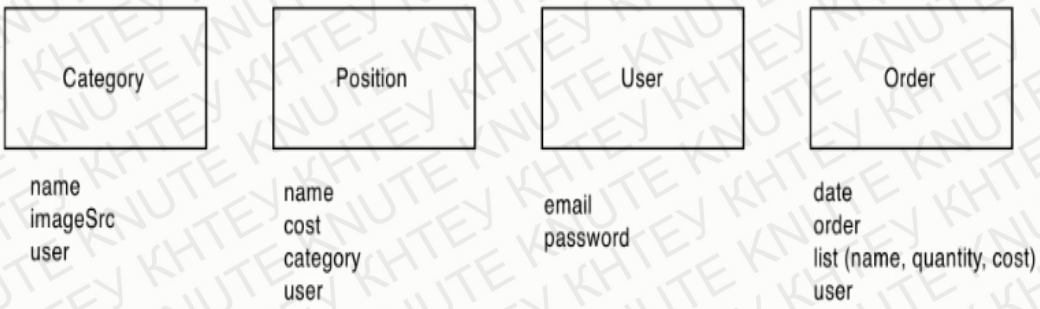

Рис. 3.2. Моделі веб-додатку

Після створення архітектури та визначення сутностей додатку, для більшої повноцінності нашого веб-додатку треба розробити API запити.

3.4. API запити (REST API)

API - application programming interface (інтерфейс програмування додатків), набір правил і механізмів, за допомогою яких один додаток або компонент взаємодіє з іншими

До Web API відносяться XML-RPC і JSON-RPC, SOAP і REST.

RPC (remote procedure call - «віддалений виклик процедур») - поняття, що поєднують давні, середні і сучасні протоколи, які дозволяють викликати метод в іншому додатку. XML-RPC - протокол, що з'явився в 1998 р незабаром після появи XML. Спочатку він підтримувався Microsoft, але незабаром Microsoft повністю переключилася на SOAP, тому в .Net Framework нема класів для підтримки цього протоколу. Незважаючи на це, XML-RPC продовжує жити до сих пір в різних мовах (особливо в PHP).

SOAP з'явився в 1998 р стараннями Microsoft. Він був анонсований як революція в світі ПО. Не можна сказати, що все пішло за планом Microsoft. Протокол продовжував розвиватися і плодитися десятками нових і нових специфікацій, поки в 2003 р W3C не затвердила в якості рекомендації SOAP 1.2, який і зараз - останній. Сімейство у SOAP вийшло значне: WS-Addressing, WS-Enumeration, WS-Eventing, WS-Transfer, WS-Trust, WS-Federation, Web Single Sign-On.

REST (Representational state transfer) - це стиль архітектури програмного забезпечення для розподілених систем, таких як World Wide Web, який, як правило, використовується для побудови веб-служб. Термін REST був введений у 2000 році Роєм Філдінгом, одним з авторів HTTP-протоколу. Системи, що підтримують REST, називаються RESTful-системами.

У загальному випадку REST є дуже простим інтерфейсом управління інформацією без використання якихось додаткових внутрішніх прошарків. Кожна одиниця інформації однозначно визначається глобальним ідентифікатором, таким як URL. Кожна URL в свою чергу має строго заданий формат.

Принципи REST.

- Клієнт-серверна архітектура - без цього REST немислимий.
- Будь-які дані - ресурс.
- Будь-ресурс має ID, за яким можна отримати дані.
- Ресурси можуть бути пов'язані між собою - для цього в складі відповіді передається або ID, або, як частіше рекомендується, посилання.
- Використовуються стандартні методи HTTP (GET, POST, PUT, DELETE) – так як вони вже закладені в складі протоколу, можна їх використовувати для того, щоб побудувати каркас взаємодії з нашим сервером.

- Сервер не зберігає стан - це значить, сервер не відокремлює один виклик від іншого, не зберігає всі сесії в пам'яті. Це значною мірою знижує масштабування - при додаванні ще одного вузла все прекрасно працює.

Властивості HTTP-методів.

HTTP Method	Idempotent	Safe
OPTIONS	Yes	Yes
GET	Yes	Yes
HEAD	Yes	Yes
PUT	Yes	No
POST	No	No
DELETE	Yes	No
PATCH	Yes	No

Рис. 3.3. HTTP-методів

Як відбувається управління інформацією сервісу - це цілком і повністю ґрунтується на протоколі передачі даних. Найбільш поширений протокол звичайно ж HTTP. Для HTTP дію над даними задається за допомогою методів: GET (отримати), PUT (додати, замінити), POST (додати, змінити, видалити), DELETE (видалити). Таким чином, дії CRUD (Create-Read-Updtae-Delete) можуть виконуватися як з усіма 4-ма методами, так і тільки за допомогою GET і POST.

Як видно архітектура REST дуже проста в плані використання. По виду прийнятого запиту відразу можна визначити, що він робить, не розбираючись в форматах (на відміну від SOAP, XML-RPC). Дані передаються без застосування додаткових шарів, тому REST вважається менш ресурсномістких, оскільки не треба парсити запит, щоб зрозуміти що він повинен зробити і не треба переводити дані з одного формату в інший.

Практичне застосування.

Найголовніше гідність сервісів в тому, що з ними працювати може яка завгодно система, будь то сайт, flash, програма та ін. Так як методи парсинга XML і виконання запитів HTTP присутні майже всюди.

Архітектура REST дозволяє серйозно спростити цю задачу. Звичайно в реальності, того що описано мало, адже не можна кому завгодно давати можливість змінювати інформацію, тобто потрібна ще авторизація та аутентифікація. Але це досить просто дозволяється за допомогою різного типу сесій або просто HTTP Authentication.

Для того, щоб проектний додаток був повноцінним, потрібно реалізувати URL адреси для нашої серверної частини в REST API для того щоб обробляти запити користувача.

3.4. Результат розробки

На виході було отримано комплексну FullStack JavaScript-розробку у вигляді CRM-системи для малого бізнесу з потрібною фіксацією продажів та аналітикою яка оптимізує процес ведення бізнесу.

Даний веб-додаток ще потребує модернізації та розширення функціональності.

Основні етапи розробки

Створення всіх роутів.

Робота над структурою проекту потребувала реалізувати кілька роутів для оброки різних запитів користувача. Для цього потрібні можливості фреймворку Express.

Для того, щоб створити роут потрібно реалізувати конструктор Router за допомогою якого стає можливим створення нових роутів та який дає початок реалізації логіки цих роутів.

Для реалізації логіки та функціоналу роутів були створенні контролери, які будуть заповнюватися в випадку визначених роутів (всі контролери зберігаються директорії «controllers»).

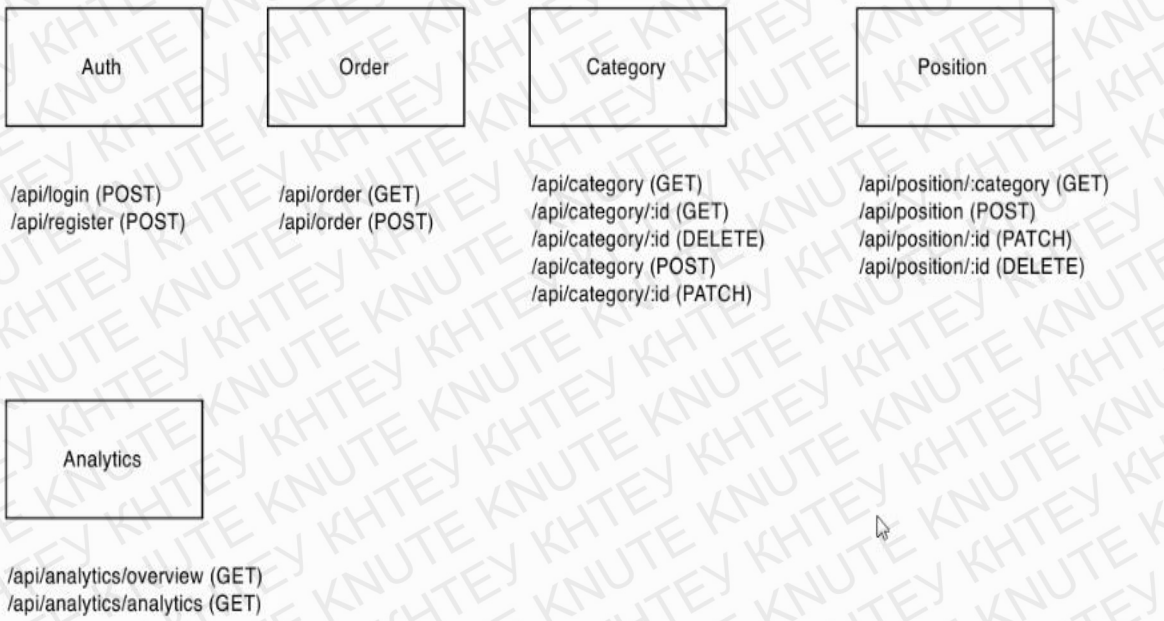

Рис. 3.4. Роути веб-додатку

Роут Auth призначений для авторизації користувача, який має два роути(URL адрес):

Api/login (POST) – логін до системи;

Api /register (POST) – дає змогу регіструватися в систему.

Роут Order призначений для заказів в якому буде два роути:

Api /order (GET) – дає змогу отримати список всіх замовлень які були зроблені в системі;

Api/order (POST) – дає можливість створювати нові замовлення.

Роут Category складється з 5 сутностей:

Api/category (GET) – дозволяє отримати всі категорії які у нас є;

Api/category/:id (GET) – дозволяє отримати конкретну категорію;

Api/category/:id (DELETE) – дає можливість видалити категорію;

Api/category (POST) – дозволяє створювати нові категорії;

Api/category/:id (PATCH) – дозволяє редагувати вже існуючу категорію.

Роут Position являється підсутністю блока Category і складається з 4 сутностей:

Api/position/:category (GET) – дозволяє отримати список всіх позицій які відносяться до потрібної категорії;

Api/position (POST) – дозволяє створити потрібну позицію;

Api/position/:id (PATCH) – редагування потрібної позиції;

Api/position/:id (DELETE) – видалення потрібної позиції.

Роут Analytics який відповідає за деяку аналітику бізнесу:

Api/analytics/overview (GET) – перегляд аналітики;

Api/analytics/analytics (GET) – вивід даних.

Парсинг даних користувача.

Коли всі потрібні роути для додатку створені їх треба протестувати, оскільки в браузері можна протестувати тільки запити з методом GET (тому що, ми щось отримаємо у відповідь).

Для цього потрібен інструмент Postman який дозволить тестувати URL адреси та запити.

Підключення утиліт.

Для більш комфортної розробки проектного додатку було потрібно встановити пакет cors та morgan.

Пакет cors потрібен для того щоб сервер міг обробляти cors запити.

Cross-Origin Resource Sharing (CORS) є технікою для ослаблення правила одного джерела, дозволяючи JavaScript на web сторінці обробляти відповідь від REST API від іншого джерела.

Правило одного джерела є важливим принципом забезпечення безпеки, реалізоване web браузерами для запобігання виконання JavaScript кодом запитів до інших джерел (тобто. До інших доменів), а не до того, з якого він переданий. Незважаючи на ефективність такого правила, воно також обмежує в законних взаємодіях між сервером і клієнтами з надійних джерел.

Пакет morgan потрібен для більш зручного процесу логування, оскільки на цьому етапі у додатку ще немає GUI для відображення свого стану та стану сервера.

Логування - незамінний інструмент в налагодження JS коду.

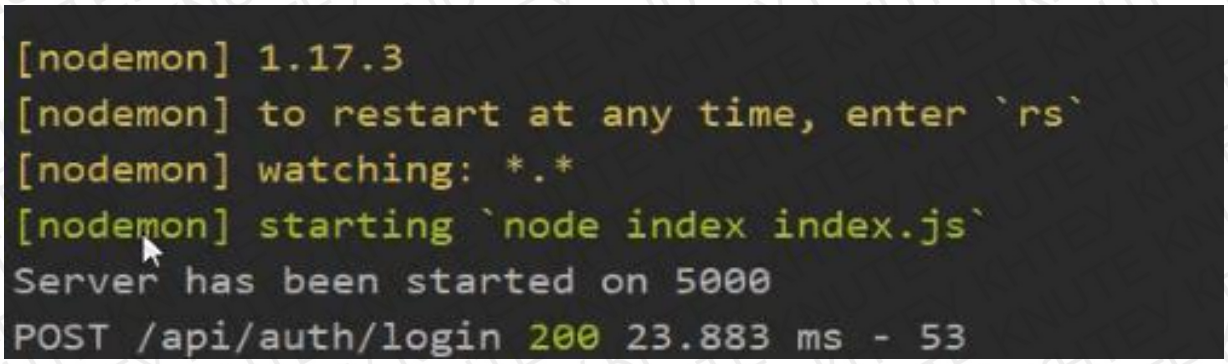

Рис. 3.5. Інформація про роботу додатку на сервері

Створення моделей.

Після створення роутів та контролерів потрібно було створити моделі, які будуть взаємодіяти з СУБД. Для цього потрібно було додати до проекту фреймворк Mongoose.

Авторизація (Захист роутів з Passport.js.).

Треба захистити роути від доступу неавторизованих користувачів, доступ можуть мати тільки ті користувачі, які пройшли авторизацію та отримали свій валідний токен. В цьому допоможе функціонал фреймворку Passport.js. та стратегія passport-jwt.

Завдання авторизації виникає практично в кожному Node.js проект, однак, щоб її правильно налаштувати, необхідно підключити велику кількість модулів і зібрати купу інформації з різних джерел.

JSON Web Token (JWT) - це JSON об'єкт, який визначений у відкритому стандарті RFC 7519. Він вважається одним з безпечних способів передачі інформації між двома учасниками. Для його створення необхідно визначити заголовок (header) із загальною інформацією по токені, корисні дані (payload), такі як id користувача, його роль і т.д. і підписи (signature).

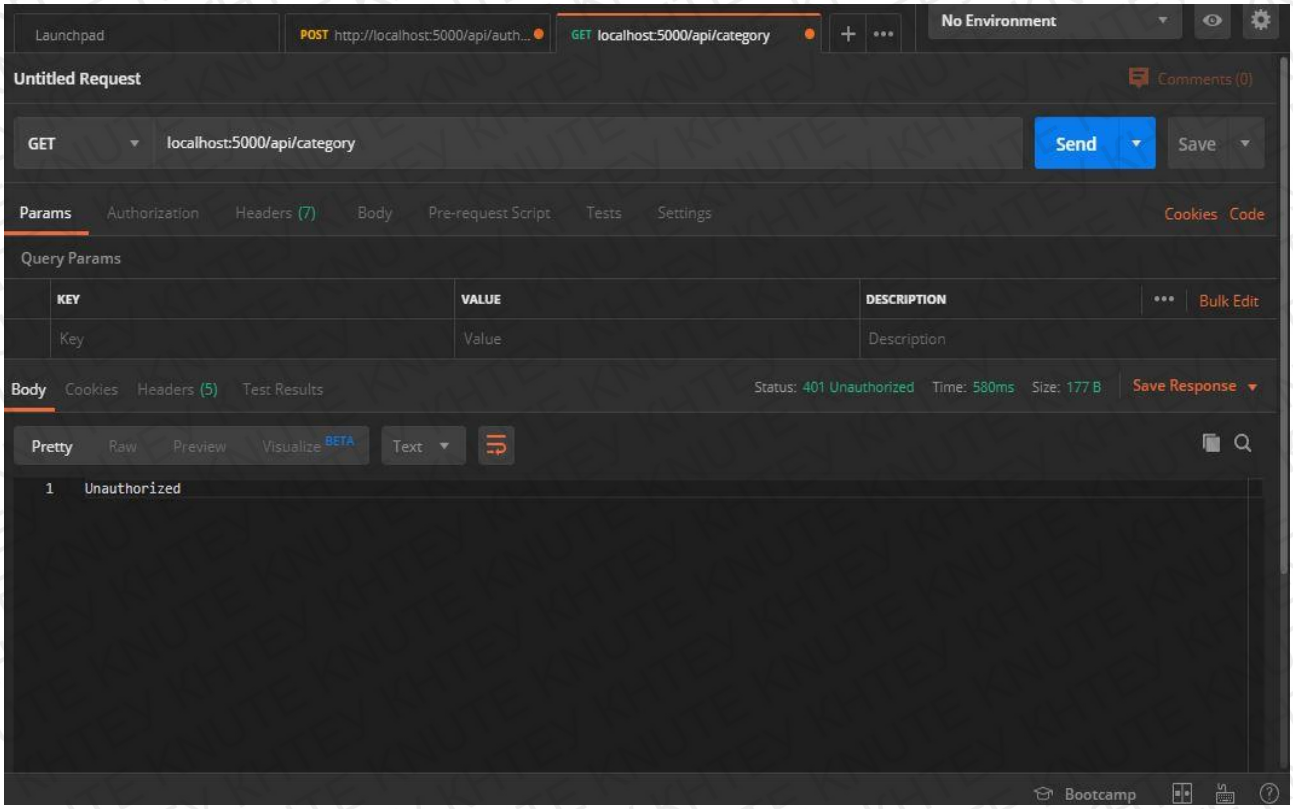

Рис. 3.6. Passport відреагувала на запит неавторизованого користувача без валідного точену і видав помилку

Авторизація (універсальний обробник помилок).

Для спрощення розробки було написано універсальний обробник помилок (catch (e)).

Для цього в корні проекту була створена папка «utils» в якій знаходиться файл обробки помилок errorHandler.js.

Дана утиліта представляє з себе просту функцію яка приймає в себе 2 параметри: response який відповідає користувачу та error.

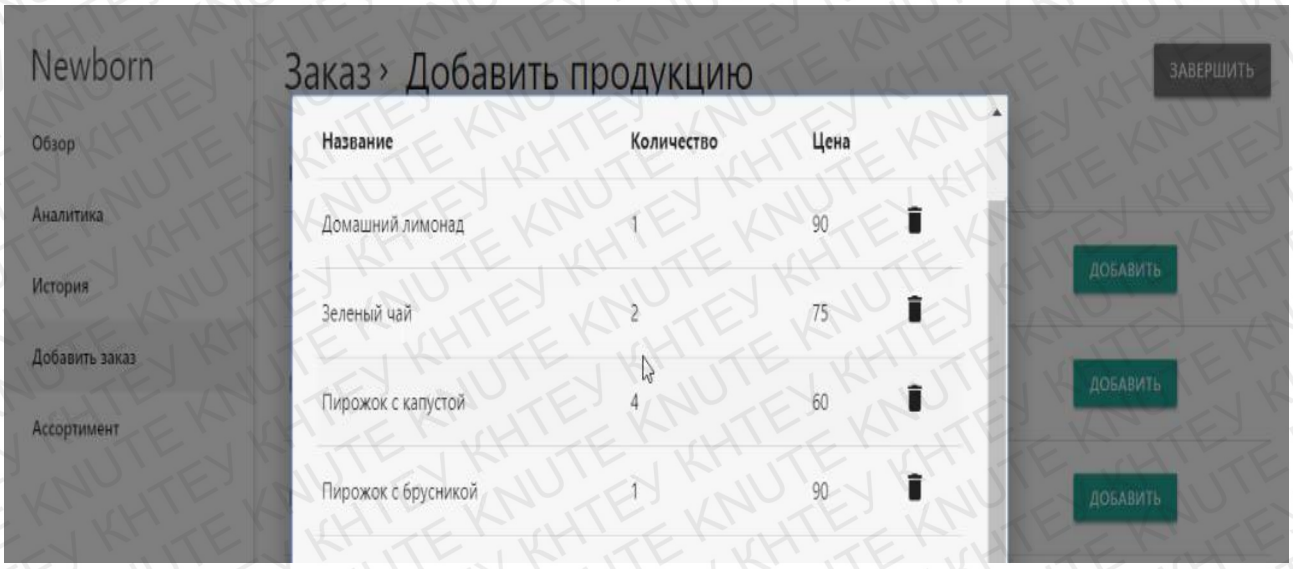

Рис. 3.7. Інтерфейс додатку (перегляд замовлення)

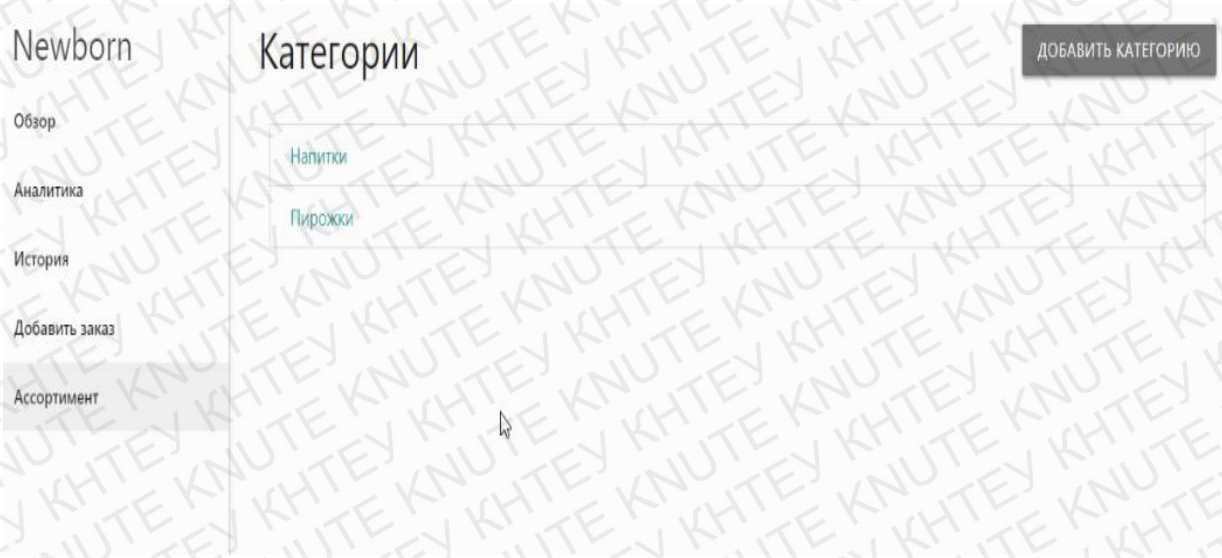

Рис. 3.8. Інтерфейс додатку (категорії)

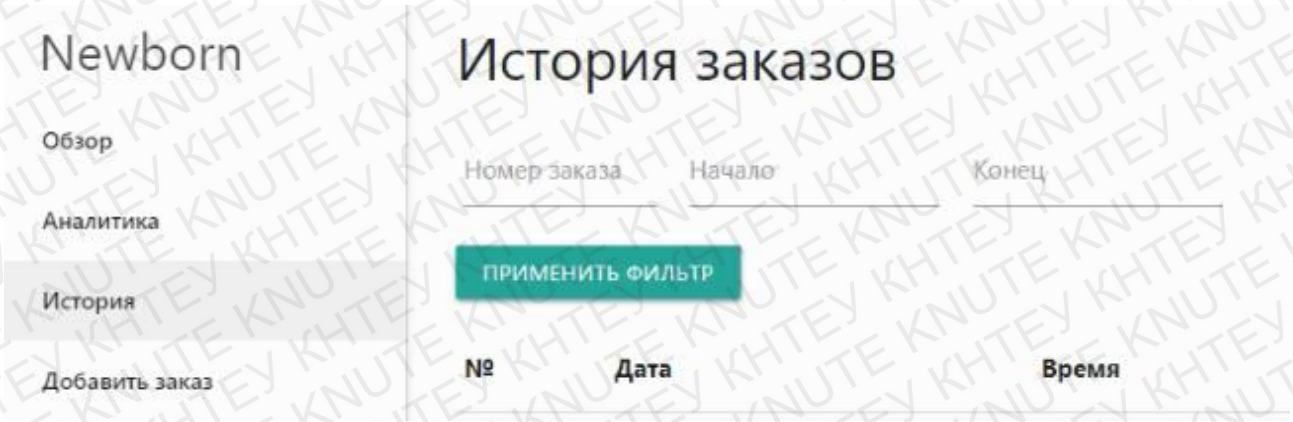

Рис. 3.9. Інтерфейс додатку (історія замовлень та фільтрація)

Висновок до розділу 3

В даному розділі були розкриті ключові аспекти розробки на MEAN STACK, також даний проект дає повне розуміння алгоритму розробки складного Full Stack-додатку на прикладі створення CRM-системи з нуля.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

У цьому дипломному проекті був реалізований комплексний Fullstack Javascript веб-додаток у вигляді CRM-системи, яка повинна аналізувати ефективність продажів для малого бізнесу.

В першому розділі були розглянуті основи про бізнес-процеси та CRM-системи.

У другому розділі було розглянуто основні інструменти, які потрібні для комплексної реалізації проектної роботи на MEAN stack.

В третьому розділі були розкриті основні аспекти реалізації проектного веб-додатку.

На виході було отримано комплексний веб-додаток, який повність розкриває алгоритм реалізації повного Fullstack додатку на MEAN stack.

Можна сказати, що в результаті роботи вдалося досягти поставлених цілей і вирішити заявлені завдання.

Список використаних джерел

1. Monteiro F., Learning Single-page Web Application Developmen / Monteiro F. Packt Publishing, 2014 – 214 с.
2. Козловский П., Разработка веб-приложений с использованием AngularJS / Козловский П., Дарвин П. - ДМК Пресс, 2014 – 394 с.
3. Knol A., Dependency Injection with AngularJS / Knol A. - Packt Publishing, 2015– 78 с.
4. Seshardi S., AngularJS: Up and Running. / Seshadri S., Green B. - O'Reilly Media,2014 – 322 с.
5. Williamson K., Learning AngularJS. / Williamson K. - O'Reilly Media, 2015 – 212 с.
6. Gechev M., Switching to Angular 2 / Gechev M. - Packt Publishing, 2016 – 254 с.
7. Herrington J., Learning AngularJS. / Herrington J. - Packt Publishing, 2015 – 235 с.
8. Тягненко, В. В. Оптимизация бизнес-процессов предприятия как способ эффективного ведения бизнеса в условиях мирового финансового кризиса / В. В. Тягненко / Актуальные вопросы экономических наук. – 2013.
9. Скрипюк, И. И. Краткий курс оптимизации бизнес-процессов на примере процесса продаж и управления персоналом / И. И. Скрипюк. – Санкт-Петербург : ООО «Издательство Форум Медиа», 2014.
10. Чевтаєв М.В. Розробка програмного забезпечення для оптимізації бізнес-процесів підприємства// Software engineering комп'ютерних систем. – К.: Київ. нац. торг.-екон. ун-т, 2019 – 125 с.

Додатки

Додаток А

Лістинг контролерів проекту

Контролер auth

```
const bcrypt = require('bcryptjs')
const jwt = require('jsonwebtoken')
const User = require('../models/User')
const keys = require('../config/keys')
const errorHandler = require('../utils/errorHandler')

module.exports.login = async function(req, res) {
  const candidate = await User.findOne({email: req.body.email})

  if (candidate){
    //Проверка пароля, пользователь существует
    const passwordResult = bcrypt.compareSync(req.body.password,
candidate.password)
    if (passwordResult) {
      //Генерация токена, пароли совпали
      const token = jwt.sign({
        email: candidate.email,
        userId: candidate._id
      }, keys.jwt, {expiresIn: 60 * 60})

      res.status(200).json({
        token: `Bearer ${token}`
      })
    } else {
```

```
      // Пароли не совпали
      res.status(401).json({
        message: 'Пароли не совпадают. Попробуйте снова.'
      })
    }
  } else {
    //Пользователя нет, ошибка
    res.status(404).json({
      message: 'Пользователь с таким email не найден.'
    })
  }
}


module.exports.register = async function(req, res) {
  //email password

  const candidate = await User.findOne({email: req.body.email})
  if (candidate) {
    //если пользователь существует, то отдаем ощибку
    res.status(409).json({
      message: 'Такой email уже занят. Попробуйте другой.'
    })
  } else {
    //нужно создать пользователя
    const salt = bcrypt.genSaltSync(10)
    const password = req.body.password
    const user = new User({
      email: req.body.email,
```

```
    password: bcrypt.hashSync(password, salt)
   })
   try{
    await user.save()
    res.status(201).json(user)
   } catch(e) {
    //обработать ошибку
    errorHandler(res, e)
   }
  }
}
```

Контролер category

```
const Category = require('../models/Category')
const Position = require('../models/Position')
const errorHandler = require('../utils/errorHandler')

module.exports.getAll = async function(req, res){
  try {
    const categories = await Category.find({user: req.user.id})
    res.status(200).json(categories)
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.getById = async function(req, res){
  try {
    const categories = await Category.findById(req.params.id)
```

```
    res.status(200).json(category)
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.remove = async function(req, res){
  try {
    await Category.remove({_id: req.params.id})
    await Position.remove({category: req.params.id})
    res.status(200).json({
      message: 'Категорія видалена.'
    })
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.create = async function(req, res){
  console.log(req.user)
  const category = new Category({
    name: req.body.name,
    user: req.user.id,
    imageSrc: req.file ? req.file.path : ''
  })

  try {
    await category.save()
    res.status(201).json(category)
```

```
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.update = async function(req, res){
  const updated = {
    name: req.body.name
  }
  if (req.file) {
    updated.imageSrc = req.file.path
  }

  try {
    const category = await Category.findOneAndUpdate(
      {_id: req.params.id},
      {$set: update},
      {new: true}
    )
    res.status(200).json(category)
  } catch (e) {
    errorHandler(res, e)
  }
}
```

Контролер order

```
const Order = require('../models/Order')
const errorHandler = require('../utils/errorHandler')
```

```
module.exports.getAll = async function(req, res){
  const query = {
    user: req.user.id
  }
//date of start
if (req.query.start){
  query.date = {
    //больше или равно
    $gte: req.query.start
  }
}

if(req.query.end){
    if (!query.date) {
      query.date = {}
    }

    query.date['$lte'] = req.query.end
}

if(req.query.order) {
  query.order = +req.query.order
}

  try {
    const order = await Order
      .find(query)
      .sort({date: -1})
      .skip(+req.query.offset)
```

```
      .limit(+req.query.limit)

    res.status(200).json(orders)
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.create = async function(req, res){
  try {
    const lastOrder = await Order
      .findOne({usser: req.user.id})
      .sort({date: -1})

    const maxOrder = lastOrder ? lastOrder.order : 0

    const order = await new Order({
      list: req.body.list,
      user: req.user.id,
      order: maxOrder + 1
    }).save()

    res.status(201).json(order)
  } catch (e) {
    errorHandler(res, e)
  }
}
```

Контролер position

```
const Position = require('../models/Position')
const errorHandler = require('../utils/errorHandler')

module.exports.getByCategoryId = async function(req, res){
 try{
   const position = await Position.find({
    category: req.params.categoryId,
    user: req.user.id
   })
   res.status(200).json(position)
 } catch (e) {
   errorHandler(res, e)
 }
}

module.exports.create = async function(req, res){
 try{
   const position = await new Position({
    name: req.body.name,
    cost: req.body.cost,
    category: req.body.category,
    user: req.user.id
   }).save()
   res.status(201).json(position)
 } catch (e) {
   errorHandler(res, e)
 }
}
```

```
module.exports.remove = async function(req, res){
 try{
  await Position.remove({_id: req.param.id})
  res.status(200).json({
   message: 'Позиція була видалена'
  })
 } catch (e) {
  errorHandler(res, e)
 }
}

module.exports.update = async function(req, res){
 try{
  const position = await Position.findOneAndUpdate(
   {_id: req.params.id},
   {$set: req.body},
   {new: true}
  )
  res.status(200).json(position)
 } catch (e) {
  errorHandler(res, e)
 }
}
```

Додаток Б

Методика тестування

В даному проекті проводилося ручне тестування REST API за допомогою функціонала програми Postman, яка спрощує та прискорює процес тестування.

Першим кроком створюється колекція – вона об'єднує в собі запити по роботі з однією сутністю. В колекції треба додати запит, для кожного запита ми обираємо метод та редагуємо.

Керівництво користувача

Блок авторизації:

- За допомогою інпутів «email» та «пароль» реєструємося в систему або створюємо новий аккаунт;
- В інпутах маємо динамічну валідацію, тому коли користувач не зможе зайти в системи від буде знати в чому причина.

Блок Асортимент

Маємо можливість додавати нові категорії за допомогою кнопці «Додати категорію», в кожній категорії є свої позиції, категорію можна редагувати (додавати/видаляти позиції).

Блок Замовлення

Цей блок фіксує замовлення клієнта (ціну та кількість елементів), можна редагувати ці замовлення і в **блоці «Історія замовлень»** переглядати їх.

Також присутня фільтрація цих даних по даті замовлення та його номеру.

Блок Аналітики

Відображає два графіка: замовлення (тенденція замовлень), та прибуток (тенденція прибутку)

Отримана інформація стрaктується та відображається в головному блоці «Огляд».