МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут неперервної освіти
Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО
«___»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИЦІ ОСВІТНЬОГО СТУПЕНЯ **«МАГІСТР»**
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ»

Тема: «Додаток керування подіями та фінансами до банківського рахунку**»**

Виконавець: студентка групи УС-201Мз Чекан Катерина Ігорівна

Керівник: к.т.н., доцент Холявкіна Тетяна Володимирівна

Нормоконтролер:_____ Ігор РАЙЧЕВ

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Навчально-науковий інститут неперервної освіти*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 «Інформаційні технології», 122 «Комп'ютерні науки», «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____Аліна САВЧЕНКО
«_____»_____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студентки

Чекан Катерині Ігорівні
(прізвище, ім'я, по батькові)

1. Тема роботи: «Додаток керування подіями та фінансами до банківського рахунку» затверджена наказом ректора від «10» жовтня 2023 р. за №2074/ст.

2. Термін виконання роботи: 02.10.2023 – 31.12.2023р.

3. Вихідні дані до роботи: додаток керування фінансами до банківського рахунку.

4. Зміст пояснювальної записки: вступ, дослідження і аналіз предметної області, огляд використаних технологій для розробки додатку, порівняльна характеристика існуючих сервісів, опис процесу створення серверної та клієнтської частини додатку, дослідження безпеки та конфіденційності сервісу, тестування та аналіз продукту, висновки.

5. Перелік обов'язкового ілюстративного матеріалу: таблиця переваги та недоліки додатка Plum, таблиця переваги та недоліки додатка Monese, таблиця Переваги та недоліки додатка Snoop Finance, ілюстрація принципу роботи Open Banking API, REST API, JWT токена, MongoDB та Docker, зображення кроків реалізації додатку.

6. Календарний план-графік

№ п/п	*Завдання*	*Термін виконання*	*Підпис керівника*
1.	Проаналізувати літературу та джерела за темою дипломної роботи	02.10.23 – 08.10.23р.	
2.	Розроблення та затвердження плану дипломної роботи	09.10.23 – 11.10.23р.	
3.	Привести консультації з науковим керівником щодо створення першого розділу	12.10.23 – 16.10.23р.	
4.	Розробка розділу 1	17.10.23 – 28.10.23р.	
5.	Розробка розділу 2	29.10.23 – 10.11.23р.	
6.	Розробка розділу 3	13.11.23 – 24.11.23р.	
8.	Висновки та оформлення пояснювальної записки дипломної роботи	27.11.23 – 01.12.23р.	
9.	Підписання необхідних документів у встановленому порядку	04.12.22 – 08.12.23р.	
10.	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломної роботи	11.12.23 – 18.12.23р.	

7. Дата видачі завдання: «02» жовтня 2023 р.

Керівник дипломної роботи _____ Тетяна ХОЛЯВКІНА
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Катерина Чекан
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Додаток керування фінансами до банківського рахунку» містить 90 сторінок, 25 рисунків, 3 таблиці, 30 літературних джерел.

Об'єктом дослідження **є**: розробка додатку для керування особистими подіями та фінансами, інтегрованого з банківським рахунком.

Предметом дослідження є: методи, технології та інструменти управління особистими фінансами та подіями для створення додатку.

Мета роботи: створення та реалізація додатку для керування особистими подіями та фінансами до банківського рахунку.

Для досягнення поставленої мети необхідно виконати **наступні завдання**:

- дослідити предметну область (важливість вміння керувати часом та фінансами, методи для покращення цих навичків);
- проаналізувати існуючі додатки схожої тематики;
- вивчити інформацію про технології для розробки сервіса;
- розробити і протестувати додаток для керування подіями і фінансами;
- проаналізувати ефективність додатку.

Методи дослідження:

- аналіз літератури;
- програмування і тестування;
- користувацьке опитування;
- аналіз результатів;
- практичне використання.

Ключові слова: ДОДАТОК КЕРУВАННЯ ФІНАНСАМИ, REST API, MONGODB, JAVASCRIPT, БІБЛІОТЕКА REACT, JAVA, БІБЛІОТЕКА SPRING, СЕРВЕР, ІНТЕРФЕЙС КОРИСТУВАЧА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

БВ – база даних

СУБД – система управління базами даних

JS – мова програмування JavaScript

XML – розширювана мова розмітки

UI – інтерфейс користувача

REST – представницький стан передачі

API – прикладний програмний інтерфейс

POJO – простий об’єкт Java

JSON – нотація об'єктів JavaScript

JWT – токен JavaScript Web

DOM – об’єктна модель документу

ВСТУП

На сьогоднішній день дуже важливо вміти керувати своїми фінансами і часом для розвитку своєї продуктивності та доглядом за своїми ресурсами. Маючи повний контроль над подіями свого життя, людина почуває себе впевнено, легше досягає поставлені перед собою цілі і мінімізує стрес та помилки. Дані навички важливі не лише в особистому житті, але й в бізнесі. За допомогою тайм менеджменту і моніторингу фінансами, збільшується конкурентоспроможність і гнучкість реакції на конкретні зміни.

Саме для цього зручно використовувати різні додатки або сервіси, до яких можна мати доступ з будь-якого місця та тоді, коли це буде потрібно. Існує багато додатків, які виконують функцію допомоги керування часом. Наприклад, Any.do, Trello, Todoist та Google Календар. Також багато сервісів створені для того, щоб користувач моніторив свої особисті кошти. Прикладом можуть слугувати Money Manager, Bills Monitor, Goodbudget, Splittable та Expensify.

Метою дипломної роботи є створення та реалізація додатку для керування особистими подіями та фінансами до банківського рахунку. **Науковою новизною є** поєднання інструментів для управління часом і фінансами в одному додатку, а також можливість взаємодії з банківським рахунком для автоматичного моніторингу витрат. Людина може бачити відображення запланованих корпоративних чи особистих подій і одночасно контролювати витрати на кожну з годину часу свого життя. Даний додаток може отримувати доступ до банківської карти та пропонувати деякі заощадження опираючись на історію витрат.

Об'єктом дослідження є розробка додатку для керування особистими подіями та фінансами, інтегрованого з банківським рахунком. **Предметом дослідження є** методи, технології та інструменти управління особистими фінансами та подіями для створення додатку.

Для розробки додатку використовуються сучасні технології для швидкої реалізації продукту, високої надійності, безпеки і ефективної підтримки. Серверна частина реалізована за допомогою мови програмування Java, всі дані зберігаються у нереляційній базі даних (NoSQL) під назвою MongoDB. Бібліотека React, в основі якої лежить мова програмування JavaScript, була використана для створення інтерфейсу користувача. Для зручного та швидкого програмування використані такі інтегровані середовища розробки, як IntelliJ IDEA та Visual Studio Code. Модуль Spring Security забезпечує повну конфіденційність та безпеку даних користувача.

Актуальність теми полягає в тому, що багато людей не слідкує за кількістю коштів, які вони витрачають або отримують впродовж дня. Тому важливо мати сервіс, який буде контролювати це та допомагати візуально бачити свої витрати та події, які заплановані на певний період часу. Даний сервіс може використовуватися як для особистих цілей, так і для робочих. Особливо зручно користуватися таким додатком фріланс-працівникам. Людина чітко бачить кількість коштів, які вона отримає чи витратить за кожну годину свого життя.

Особистий внесок випускника полягає в розробці архітектури додатку, написанні серверну частину, створенні інтерфейс користувача та реалізації інтеграцію з банківським API. Для реалізації проекту були використані методи аналізу літератури, програмування, тестування та користувацького опитування.

Розроблений додаток був протестований і використаний декількома користувачами, які працюють в різних професійних сферах для оцінки його ефективності та зручності використання. Також результати дослідження можуть бути представлені в наукових журналах або на конференціях з інформаційних технологій.

Отже, даний додаток є дуже корисним в будь-який час та для багатьох спеціалістів. Розроблений сервіс має **практичне застосування** для

індивідуальних користувачів, фрілансерів і бізнес-осіб, які шукають зручний спосіб керувати своїми фінансами та часом Тільки з умінням планувати та керувати своїм часом і фінансами, людство може стати успішним та прогресивним.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ І АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Значення керування часом в житті людини

1.1.1. Загальна інформація про Time Management

Малі підприємства та фрілансери повинні оптимізувати свій робочий час для ефективного розвитку свого бізнесу. Вміння управляти часом є ключовим елементом для зайнятих фахівців, оскільки воно допомагає встановити пріоритети для всіх завдань і прискорити досягнення цілей. Збалансоване розпорядження часом дозволить вам використовувати нові можливості та розвивати свій бізнес стійким та ефективним способом.

Важливість навичок управління часом не може бути недооціненою, особливо для власників малого бізнесу. Ось деякі ключові аспекти цього важливого набору навичок:

✓ Вміння тайм-менеджменту можна вдосконалювати і послідовно застосовувати в повсякденному житті, і цей процес не є надто складним.

✓ Ефективне управління часом принесе безліч переваг, таких як покращений баланс між роботою і особистим життям, а також більш ефективна професійна діяльність.

✓ Навички тайм-менеджменту допоможуть вам привернути нових клієнтів і зберегти існуючих.

Початок роботи з тайм-менеджментом не обов'язково є складним – ви можете розпочати вже сьогодні.

Управління часом – це стратегія організації вашого вільного часу та контролю за часом, який ви витрачаєте на конкретні завдання, з метою досягнення більшої ефективності. Покращене управління часом сприяє збалансованій роботі та особистому життю. Збалансованість між професійною діяльністю та особистим життям приносить численні позитивні користі, такі як зниження рівня стресу і більше часу для родини та близьких.

Методи управління часом можуть відрізнятися в залежності від ролі особи в організації та поставлених перед нею завдань. Наприклад, для власника бізнесу тайм-менеджмент може допомогти зменшити витрати часу на рутинні завдання, щоб більше уваги приділяти стратегічному плануванню. Для фрілансера, незалежного від організації, метою управління часом може бути забезпечення того, щоб кожен клієнт отримував відповідну увагу, не відводячи при цьому надто багато часу.

1.1.2. Переваги ефективного управління часом

Деяким людям легше дається ефективне управління часом, ніж іншим, але кожен може розвивати навички, щоб покращити своє управління часом. Без ефективного управління часом якість роботи може страждати, терміни можуть бути пропущені, рівень стресу зросте, а рівновага між роботою та особистим життям може порушитися, що може вразити вашу професійну репутацію.

Переваги ефективного управління часом:

Підвищення продуктивності. Планування часу для виконання важливих завдань допомагає вам краще розуміти ваші обов'язки та час, що потрібно на кожне завдання. Це сприяє ефективності, уникненню відволікань і зосередженню на головних завданнях.

Покращення якості роботи. Без постійного гонитися за термінами ви маєте більше часу для уважного планування і виконання робіт. Це

призводить до підвищення якості результатів, оскільки ви можете робити свою роботу більш ретельно і з уважністю.

Вчасне виконання завдань. Грамотне розподілення часу на кожне завдання допомагає вам завжди встигати до термінів. Також, це надає можливість запасати час на непередбачувані ситуації, що зменшує стрес та ризик пропуску дедлайнів.

Зниження рівня стресу. Ефективне управління часом допомагає структурувати ваш список завдань та надає можливість розставити пріоритети. Це допомагає зменшити стрес, адже ви чітко розумієте, що потрібно робити та скільки часу вам на це потрібно.

Зростання кар'єрних можливостей. Вміння керувати часом робить вас надійним та дисциплінованим співробітником, завжди готовим до виконання завдань вчасно і якісно. Це сприяє підвищенню цінності на робочому місці і покращенню професійної репутації.

Підвищення впевненості. Спроможність досягати цілей вчасно сприяє впевненості у власних силах. Постійне виконання завдань зі списку сприяє мотивації та розвитку навичок управління часом.

Підвищення ефективності. Розуміння того, як ефективно керувати своїм часом, дозволяє фокусуватися на завданнях у більш концентрований спосіб, що дозволяє досягати більших результатів в коротший час. Це означає, що ви можете більше досягти, витрачаючи менше часу. Наприклад, замість намаганняться вмістити великий проект у короткий час, ви можете розподілити його на невеликі завдання та виконувати їх, коли є короткі вільні моменти, що дозволяє використовувати час більш ефективно.

Ефективне управління часом має численні переваги як для вашої професійної діяльності, так і для забезпечення балансу між роботою і особистим життям. Тайм-менеджмент грає важливу роль в збереженні контролю над вашим робочим графіком і дозволяє розвивати бізнес, не втрачаючи особистого життя.

Навички ефективного управління часом вимагають часу для розвитку і впровадження на практиці. Проте, коли ви опануєте ці навички, ви помітите, що для досягнення тих самих результатів потрібно менше зусиль. Це допоможе вам економити час на виконанні завдань, підтримувати високий рівень енергії і продуктивності та відчувати менше стресу під час роботи зі списком справ. [1]

Основи управління часом: важливі аспекти

Для більшості людей легко розуміти, чому вивчення основ управління часом є настільки корисним – значення навичок управління часом неможливо переоцінити. Проте впровадження їх у щоденне життя для досягнення більшої продуктивності та одночасного збереження балансу між роботою та особистим життям може виявитися викликом.

1.1.3. Поради для покращення вміння керувати своїм часом

Вдосконалення навичок управління часом може зробити вас більш продуктивними та менше стресовими. Ось 5 простих кроків для покращення управління часом:

Плануйте заздалегідь. Планування вашого часу заздалегідь є найважливішим аспектом ефективного управління часом. Розумійте, коли ви найбільш продуктивні: можливо, рано вранці або пізно ввечері. Резервуйте цей час для важливих і складних завдань. Залиште менш важливі або менш термінові завдання на потім, коли у вас буде більше часу для них. Перед початком кожного дня зробіть план того, скільки часу ви плануєте витратити на кожне завдання зі свого списку.

Розставте пріоритети. Оцініть кожен проект, над яким вам потрібно працювати, щоб визначити, які завдання є найбільш терміновими та важливими, і зробіть їх головними пріоритетами на день. Залиште менш важливі завдання або проекти, які ще не стали терміновими, на потім, коли у вас буде більше часу для них.

Позбавтеся від відволікань. Відволікання – один із найбільших ворогів продуктивності. Соціальні медіа, смартфони та власні колеги можуть легко відвернути вашу увагу від пріоритетів і порушити ваш робочий ритм. Дослідження Think Money показали, що третина працівників відволікається на три години робочого дня. Якщо ви вважаєте, що деякі відволікання особливо сильні, розгляньте можливість залишити смартфон у столі або використовувати програми, які блокують веб-сайти, що вас найбільше відволікають.

Уникайте багатозадачності. Багатозадачність може здаватися корисним методом для виконання більшої кількості роботи, але фактично вона знижує продуктивність. Замість того, щоб розпочинати кілька завдань і не завершувати жодного з них, краще зосередьтеся на одній справі і приділяйте їй всю увагу, щоб уникнути помилок.

Проводьте винагороди за хорошу роботу. Винагороди можуть бути відмінним стимулом для ефективного управління часом. Дозвольте собі маленькі винагороди за кожне завдання, яке ви виконуєте протягом дня. Наприклад, ви можете відсвяткувати завершення важливого проекту короткою прогулянкою на свіжому повітрі. Винагороди підтримують вашу мотивацію і можуть допомогти зберегти баланс між роботою і особистим життям.

1.2. Ведення обліку витрат

Як власник бізнесу або приватна особа, легко залишитися поглинутими щоденною справжньою метушею і забути про слідкування за витратами. Проте відстеження витрат може принести численні переваги, від допомоги в кращому управлінні фінансами до зменшення стресу під час податкового періоду. У цьому блоз-пості ми розглянемо поняття відстеження витрат, пояснимо, чому воно важливе як для підприємств, так і

для індивідуальних осіб, покажемо, як це робити ефективно і результативно, а також розкажемо про переваги, які може принести ретельне управління своїми фінансами.

Ведення обліку витрат – це процес реєстрації та контролю всіх фінансових транзакцій, пов'язаних з вашим бізнесом або особистими витратами. Це включає в себе відстеження різних видів витрат, від невеликих покупок, таких як офісні принади та продукти, до більших, таких як придбання обладнання або платежі за оренду.

Існують різні методи ведення обліку витрат, від простих електронних таблиць до складних програм, які автоматизують цей процес. Один із поширених підходів - ведення докладного журналу всіх витрат за допомогою електронних таблиць Excel або спеціалізованого бухгалтерського програмного забезпечення.

Облік витрат може бути корисним у різних контекстах, включаючи управління особистими фінансами, малі підприємства, великі корпорації, державні установи та неприбуткові організації. Незалежно від того, де ви застосовуєте його, основна мета залишається однаковою – надавати окремим особам та організаціям точну інформацію щодо джерел доходів і витрат, щоб допомогти приймати обґрунтовані фінансові рішення.

Регулярне відстеження витрат, найкраще щодня, надасть вам корисну інформацію про те, скільки грошей ви витрачаєте щомісяця і куди вони йдуть. Ця інформація допоможе вам приймати кращі рішення щодо скорочення витрат (якщо це необхідно) та ефективніше вкладати ресурси в можливості зростання. [2]

Відстеження особистих витрат є дієвим інструментом, що сприяє контролю над фінансами і прийняттю обґрунтованих рішень щодо використання грошей. Крім того, це перший крок у формуванні особистого бюджету. Відси постають деякі переваги належного ведення обліку особистих витрат:

1. Розуміння особистих витратних звичок: Однією з ключових переваг відстеження особистих витрат є можливість осмислено розглядати власні звички витрачати кошти. Реєструючи витрати, ви маєте можливість аналізувати напрямки своїх фінансових витрат щоденно, щотижня або щомісяця та визначати області, де можна заощадити. Наприклад, ви можете виявити, що витрачаєте занадто багато грошей на їжу поза домом або на непотрібний одяг. Інколи ви можете побачити, що покупки в оптовому обсязі, такі як продукти чи одяг, можуть допомогти вам економити гроші та час.

2. Встановлення фінансових цілей: Відстеження особистих витрат може також допомогти вам ставити перед собою фінансові цілі. Розуміючи власні звички витрачати гроші, ви можете надавати реалістичний характер своїм фінансовим цілям. Наприклад, ви можете вирішити вкладати кошти в інвестиції, навчання або майбутні відпустки. Моніторинг особистих витрат може допомогти розробити план досягнення цих цілей і визначити приблизний часовий рамок для їх досягнення.

3. Виявлення можливих фінансових проблем: Відстеження особистих витрат також може допомогти виявити можливі проблеми, пов'язані з фінансами. Наприклад, якщо ви витрачаєте більше, ніж заробляєте, це може свідчити про нестачу доходу або необхідність розглянути варіанти скорочення витрат. Крім того, якщо ви регулярно перевищуєте бюджет, це може бути результатом необдуманих покупок або імпульсивного споживання. Ви також можете помітити, що деякі витрати відбуваються періодично, і ви можете заздалегідь виділити кошти на такі події.

4. Підготовка до податкового обліку: Відстеження особистих витрат може також виявитися корисним при підготовці податкових декларацій. Це дозволяє зберігати записи про бізнес-витрати, благодійні внески та інші відрахування, що може значно спростити процес подачі податкових декларацій і зменшити податкові обов'язки.

5. Контроль над особистим станом: Ведення обліку особистих витрат дозволяє вам контролювати ваш власний капітал, який представляє собою різницю між активами та пасивами. Ця інформація може допомогти вам краще розуміти своє фінансове положення і відстежувати його динаміку з часом.

6. Створення особистого бюджету: Коли ви маєте відомість про свої основні джерела доходів і витрат, ви можете легше створити особистий бюджет, який відповідає вашим потребам. Ви можете переглядати свої витрати наприкінці тижня, 10-денний або місяць, щоб переглянути їхню динаміку на майбутні дні.

1.2.1. Методи відстеження витрат

Оволодіння мистецтвом ведення обліку витрат – завдання, яке може виглядати різноманітно для кожної особи. У кожного свій власний спосіб структурування та обробки інформації. Але, якщо ви зможете адаптувати свій процес бюджетування, щоб зробити його зручним і невимушеним, ймовірно, ви будете продовжувати це робити і покращувати свої фінансові звички.

Існує кілька методів, які можуть вам допомогти відстежувати свої витрати, включаючи створення електронних таблиць, використання програмного забезпечення або навіть ведення записів вручну. Незалежно від того, яку техніку ви обираєте, вона повинна включати наступні етапи:

1. Запишіть всі ваші щомісячні доходи та витрати, включаючи постійні витрати, такі як оренда чи іпотека, і змінні витрати, такі як продукти.

2. Класифікуйте свої витрати за типом: їжа, житло, транспорт (включаючи витрати на пальне та громадський транспорт), розваги і т. д.

3. Організуйте дати оплати ключових витрат, таких як оренда і комунальні послуги, щоб завжди знати, коли їхній термін закінчується.

4. Постійно вдосконалюйте свій метод, визначаючи місця, де можлина більш раціональна витрата коштів, та виділяючи ситуації, коли доходи обмежені.

Зберігаючи всю цю інформацію в електронних таблицях чи спеціальних програмах, ви зможете відстежувати ваші витрати місяць за місяцем. Це дозволить вам краще розуміти тенденції, коригувати невиправдані звички та заздалегідь планувати довгострокові витрати. [3]

1.3. Класифікація додатків для Time Management

В умовах сучасного життя, коли час – це найцінніший ресурс, додатки для керування часом стають незамінними помічниками. Вони допомагають організувати робочий та особистий життєвий ритм, зберігаючи завдання, плани та пріоритети в одному місці. Розглянемо деякі з найкращих додатків для керування часом, які спрощують планування, поліпшують продуктивність та допомагають краще управляти своїм часом.

Існують п'ять різновидів інструментів для ефективного управління часом, які ви повинні використовувати та розумно поєднувати. Помимо цих інструментів, ви також можете використовувати різні методи управління часом, такі як "Getting Things Done", Матриця Ейзенхауера, техніка "Помідор" та інші.

Ось перелік цих п'яти інструментів:

- ✓ Система календаря.
- ✓ Управління завданнями або проектами.
- ✓ Моніторинг витрати часу.
- ✓ Застосунок для записів (з наявністю шаблонів).
- ✓ Слідкування за звичками, інструменти для уникнення відволікання та інші.

1.3.1. Календар

Початок ефективного управління та організації часу завжди пов'язаний з користуванням календарем та правильним плануванням. Використання (онлайн) календаря стає необхідною складовою робочого процесу.

Це стає очевидним фактором. Проте, ще важливіше зробити максимальний використання календаря. Багато людей обмежують його використання записами про заплановані зустрічі.

Але календар – це більше, ніж просто список зустрічей. Він може стати вашим інструментом контролю над часом, якщо ви використовуєте його на повну потужність. Вам слід вносити в календар практично всі ваші плани та обов'язки на майбутнє. Ось декілька додаткових порад, які допоможуть вам максимально використовувати календар:

- Розплануйте наступний робочий тиждень не пізніше, ніж у вечір неділі.
- Почніть з планування важливих речей: фізичні тренування, читання, час для сім'ї і відпочинку.
- Створіть блоки на 2-3 години для робочих завдань і переконайтеся, що в цей час ви присвячуєте себе найважливішим завданням, уникаючи відволікань.
- Групуйте зустрічі та телефонні дзвінки в одній часовій зоні.
- Відведіть окремий час для перевірки електронної пошти: короткі 15-хвилинні інтервали, під час яких ви відповідаєте на якомога більше листів.
- Заплануйте один день без зустрічей та інших відволікань, щоб спокійно працювати над важливими завданнями. [4]

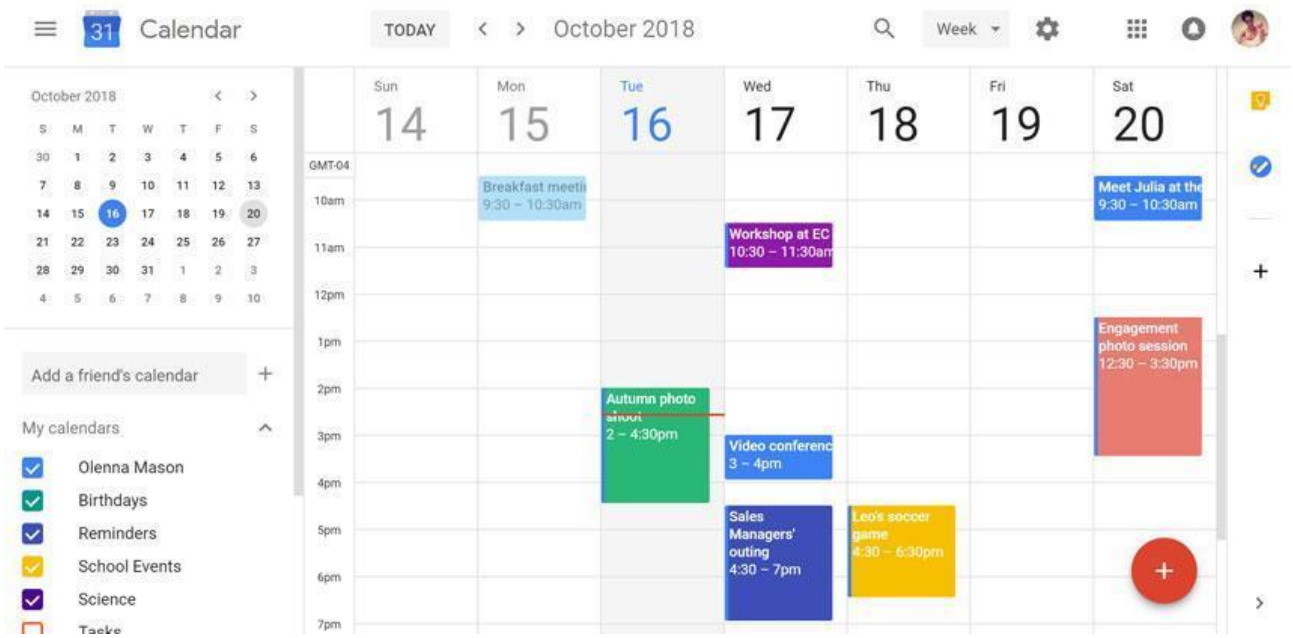

Рис. 1.1. Календар

1.3.2. Керівник завдань або керівник проекту

У застосунку календаря необхідно виділити часові блоки для виконання обраних (важливих) завдань, планування зустрічей та фіксування всього іншого, що вам необхідно зробити.

У застосунку для управління завданнями (або управління проектами) вам слід стежити за всіма завданнями, які потрібно виконати, встановлювати пріоритети та веде відслідковування вашого прогресу.

Якщо ви є фрілансером, простого рішення для керування завданнями буде більш ніж достатньо. Але якщо ви керівник більшої команди, вам варто розглянути більш потужний інструмент для управління проектами. Незалежно від обраної опції, існують два загальних типи керівництва завданнями/проектами:

1. На основі списків – Ви перелічуєте всі завдання в окремих списках.

2. На основі системи Канбан – Ви створюєте різні віртуальні дошки зі стовпцями «до виконання», «у процесі» та «виконано», а також віртуальні

картки (аналогічно запискам) для кожного завдання. Потім переміщуєте картки через стовпці, працюючи над завданнями.

Вибір між системою на основі списків чи системою Канбан повністю залежить від ваших особистих вподобань і потреб. Деякі програми навіть підтримують обидва типи відображення.

Що є ще важливішим, це можливість легко встановлювати пріоритети завдань через обрану систему. На практиці це означає, що ви можете легко змінювати порядок завдань чи віртуальних карток.

При встановленні пріоритетів завдань завжди варто розрізняти між терміновими та важливими завданнями. Термінові завдання – це ті, які потребують вашої негайної уваги.

З іншого боку, важливі завдання – це ті, які сприяють досягненню вашої довгострокової місії, цінностей та цілей. Завжди варто надавати пріоритет завданням, які є одночасно терміновими і важливими (переконайтеся, що вони видно вгорі вашого додатка для управління завданнями).

Ось чотири різних типи завдань, які потрібно належним чином управляти:

1. Терміново + Важливо (Виконати)

2. Терміново + Не важливо (Делегувати)

3. Не терміново + Важливо (Запланувати їх у своєму календарі заздалегідь)

4. Не важливо + Не терміново (Видалити)

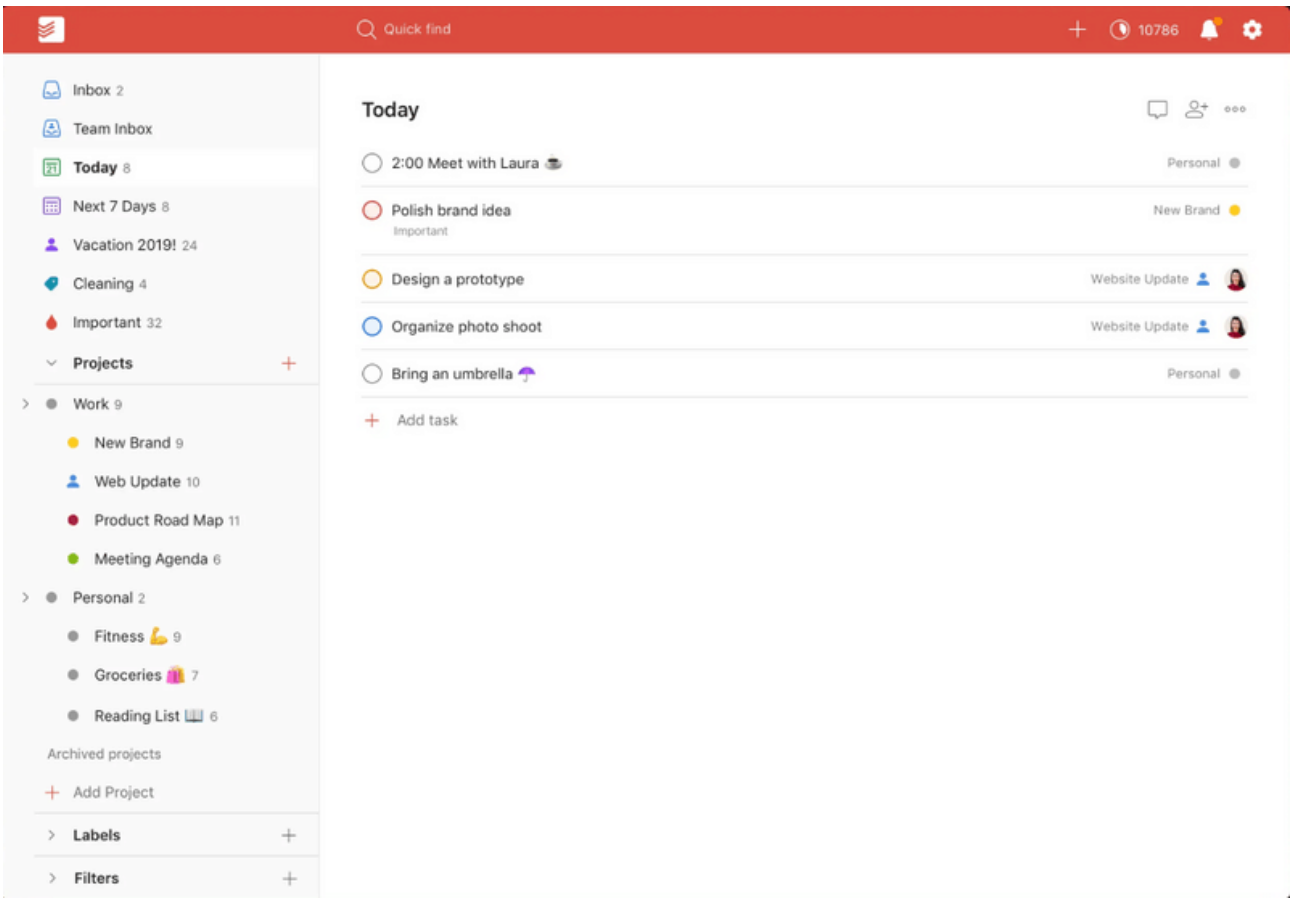

Рис. 1.2. Керівник завдань

1.3.3. Відстеження часу

Наступним у переліку інструментів для управління часом є корисний тайм-трекер. Використання відстеження часу має численні переваги, незалежно від того, чи працюєте ви самі, чи в команді.

Ефективний тайм-трекер допомагає підвищити вашу продуктивність (оскільки ви стежите за тим, як ви витрачаєте час), дозволяє вам аналізувати ваш розпорядок дня і, крім того, надає можливість автоматизувати завдання, такі як створення звітів та виставлення рахунків.

Використання тайм-трекера, без сумніву, є однією з найкращих порад щодо управління часом. Як кажуть, те, що ви вимірюєте, ви зможете краще керувати.

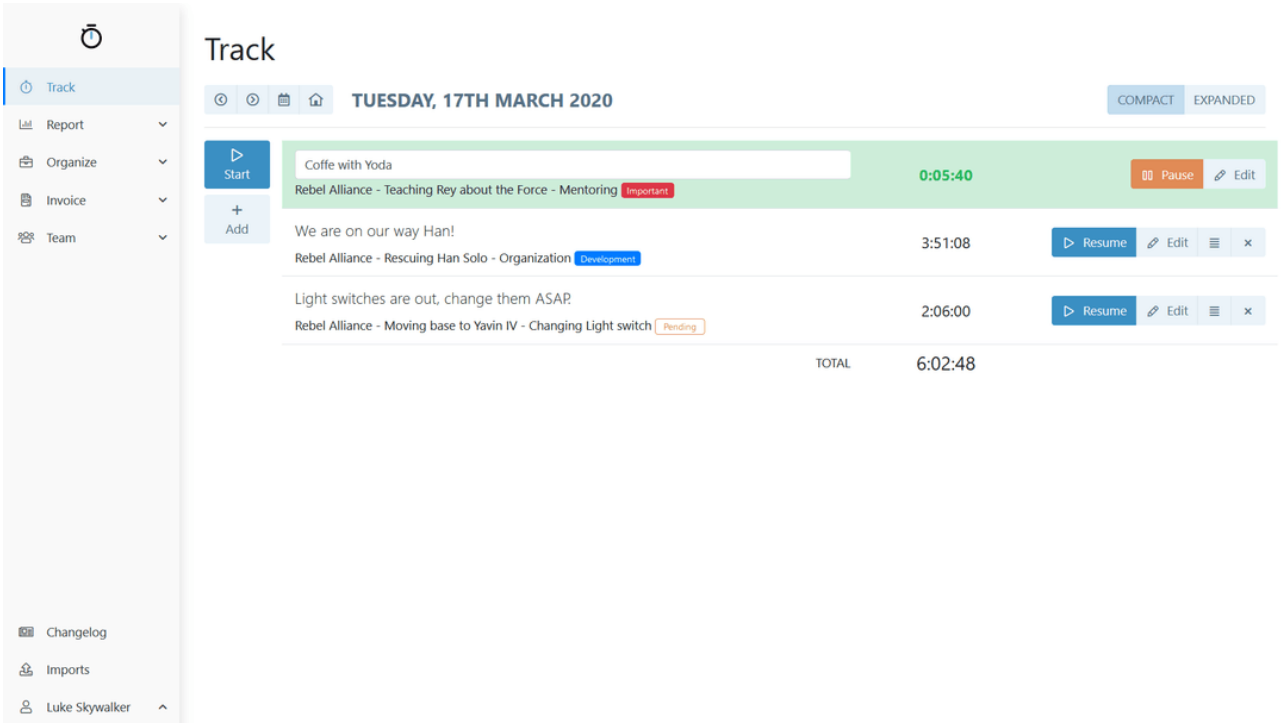

Рис. 1.3. Відстеження часу

1.3.4. Програма для нотаток (зі зразками)

Ще одним важливим інструментом, який може значно покращити ваше управління часом, є додаток для створення нотаток. Ви повинні користуватися програмою для створення записів, щоб:

1. Фіксувати ваші довгострокові (на рік) і короткострокові (на квартал, місяць) цілі.

2. Вести записи про засідання.

3. Розмішувати над вашим робочим процесом та шукати способи для покращення його.

4. Зберігати всі посилання і статті в одному місці.

5. Зберігати цифрову версію різних документів.

Для додатка для записів надзвичайно корисно мати різноманітні готові шаблони, а також можливість створювати власні.

Можливості використання таких шаблонів включають брайнштормінг, написання інструкцій та списків завдань, аналіз бізнес-питань, створення дорожніх карт, складання редакційних календарів та багато іншого. Хороший додаток для записів також повинен підтримувати різні розширення та плагіни, щоб ви могли імпортувати дані з різних джерел.

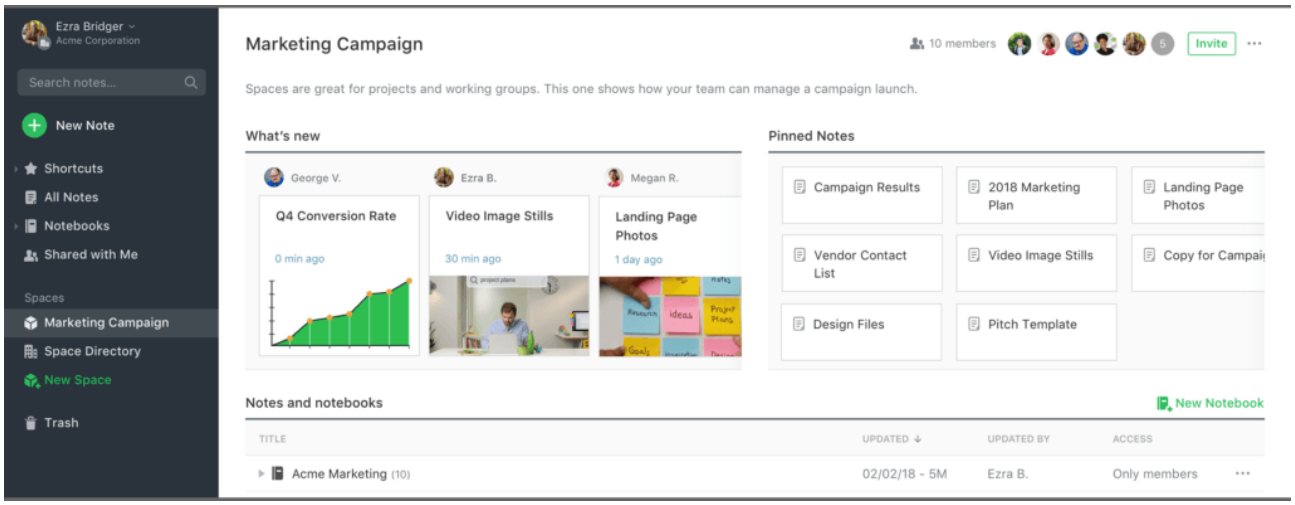

Рис. 1.4. Програма для нотаток

1.3.5. Відстеження звичок, засоби запобігання відволіканню та інше

І на останок, але не менш важливо, існують різні спеціалізовані інструменти для управління часом. Наприклад, ви можете використовувати різні засоби для відстеження звичок, програми для блокування відволікань та рішення для автоматизації та інші.

Також існують рішення, які підтримують різні методи управління часом. Просто пам'ятайте, що краще використовувати менше інструментів і використовувати їх повністю, ніж встановлювати їх забагато і використовувати лише напівповністю.

Рис. 1.5. Програма для відстеження звичок

1.4. Найкращі додатки для керуванням фінансами та їх порівняння

Відстеження витрат часто становить перший крок у створенні порядку у фінансах. Зрозумівши, куди йдуть ваші гроші та скільки ви витрачаєте, ви отримуєте точний огляд того, як розподіляються ваші фінанси і області, де можна зекономити.

Це можна легко зробити, включивши в це свою щоденну рутину завдяки додаткам для відстеження витрат, які допомагають керувати грошима, коли ви в дорозі. Ці програми, безперечно, схожі на програми для створення бюджету, але, у відмінність від останніх, додатки для відстеження витрат зосереджуються на вашому витратному способі. Зазвичай ці програми автоматично класифікують ваші витрати і допомагають зрозуміти вашу покупкову поведінку. [5]

Незалежно від того, чи ви шукаєте програму для автоматичного відстеження всіх транзакцій, чи програму для автоматизованої звітності про витрати на робочому місці, або навіть програму, яка вимагає вручну вводити кожну транзакцію, є додаток, що підходить саме для вас.

1.4.1. Plum

Plum – це фінансова програма, яка поєднує фінансові рахунки і кредитні картки з різних постачальників для того, щоб ви могли краще розуміти свої витрати.

Функція автоматичних депозитів дозволяє вам округлити ваші витрати до найближчого фунта і автоматично зберігати різницю на спеціальному рахунку, який може приносити до 3,82% річних відсотків, в залежності від вашого рівня підписки на Plum.

Також є можливість інвестувати в акції і фонди, створювати інвестиційний портфель для пенсійного забезпечення та консолідувати наявні пенсійні вклади у Plum Self Invested Personal Pension (SIPP).

Таблиця 1.1

Переваги та недоліки додатка Plum

Переваги	Недоліки
З'єдання до вашого облікового запису Надає можливість відкрити ISA та рахунки для збережень Автоматичні внески на ваш рахунок збережень Сповіщення про майбутні оплати і рахунки Дебетова карта Plum з можливістю отримання кешбеку (у пакеті Pro + Ultra)	Декого може відпугувати розширена функціональність. Найкращі можливості доступні лише у платних варіантах.

1.4.2. Monese

Monese може сприяти поліпшенню вашого кредитного рейтингу і управлінню бюджетом.

Це надає можливість технічно використовувати 0% позику без перевірки кредитної історії. Відстежуючи виплати, ви зможете підвищити свій кредитний рейтинг за допомогою Experian, Equifax і TransUnion, кредитних агентств.

Співпраця з Monese дозволяє синхронізувати ваші фінансові облікові записи, що дозволяє вам відстежувати та групувати транзакції та зберігати гроші для різних цілей заощаджень. Ви також можете налаштувати і відстежувати прямі дебети, встановлювати бюджетні цілі та контролювати свій фінансовий прогрес.

Monese співпрацює з фінансовою платформою Raisin для надання можливості зберігання коштів через свій додаток.

Таблиця 1.2

Переваги та недоліки додатка Monese

Переваги	Недоліки
Посилання до ваших облікових записів Покращуйте свій кредитний рейтинг завдяки Monese кредиту Отримайте дебетову картку з безкоштовною доставкою в усіх пакетах	Без можливості вкладень в інвестиції Найкращі можливості доступні лише у платній версії.

1.4.3. Snoop Finance

Підказка в назві Snoop Finance. Окрім того, щоб відстежувати ваші фінанси на рахунках, ця програма може допомогти вам знайти способи

заощадити гроші, починаючи від пошуку знижок на послуги доставки їжі до пошуку більш вигідних тарифів на інтернет.

Ви будете отримувати щоденні рекомендації щодо економії грошей, які будуть відповідати вашим витратам, а також спеціальні пропозиції від партнерів Snoop, а також інформацію про ваші фінанси та поради. Кожен місяць і постійно ви зможете аналізувати свої витрати за категоріями, від Amazon до комунальних платежів.

Snoop буде нагадувати вам про важливі події, такі як щоденні баланси та надходження рахунків. Він також автоматично сортує ваші витрати.

Snoop вивчає вашу фінансову історію, щоб виявити комісії (наприклад, комісії за іноземні транзакції, коли ви платите за кордоном) і пропонує більш вигідні способи витрат. Щонеділі ви також отримуватимете сповіщення з оглядом платежів на наступний тиждень.

Таблиця 1.3

Переваги та недоліки додатка Snoop Finance

Переваги	Недоліки
Посилання на ваші профілі Виявляє нерозумні витрати Оповіщення щодо надходження майбутніх рахунків	Найкращі можливості доступні лише у платних варіантах.

ВИСНОВОК ДО РОЗДІЛУ 1

В даному розділі проведено аналіз предметної області та дослідження необхідності вміти керувати особистим часом та фінансами для кожної людини.

Також була наведена загальна інформація про Time Management, керування коштами, проаналізовано методи і поради для покращення навичок керування своїми ресурсами. Більш до того, було проаналізовано, як може змінитися ваше особисте життя або розвиток бізнесу, якщо ви будете притримувати вищеописаних стратегій.

У цьому розділі перелічені додатки, які допоможуть справлятися з керуванням своїми ресурсами, приведено їх опис, класифікацію та порівняльну характеристику.

РОЗДІЛ 2

ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ДОДАТКА

2.1. Аналіз технологій для розробки серверної частини додатка

2.1.1. Мова програмування Java

Для написання серверної частини додатка було обрано мову програмування Java. Java – це популярна об'єктно-орієнтована мова програмування та платформа, що працює на численних пристроях, включаючи ноутбуки, мобільні пристрої, ігрові консолі, медичне обладнання та інші. Інструкції та синтаксис Java мають коріння в мовах C та C++.

Однією із ключових переваг розробки програм на Java є її переносимість. Код, написаний на Java для ноутбука, можна легко використовувати на мобільних пристроях. При створенні цієї мови в 1991 році Джеймсом Гослінгом з компанії Sun Microsystems (пізніше придбаною Oracle), основною метою була можливість "пишіть один раз, запускайте де завгодно".

Також важливо розуміти, що Java значно відрізняється від JavaScript. JavaScript не потребує компіляції, в той час як код Java потребує компіляції. Крім того, JavaScript працює лише у веб-браузерах, тоді як Java можна запускати в різних середовищах.

На ринку постійно з'являються нові та вдосконалені засоби розробки програмного забезпечення, які витісняють існуючі продукти. Однак Java, яка була створена понад два десятиліття тому, все ще залишається однією з найпопулярніших мов для розробки додаткового програмного забезпечення. Розробники продовжують віддавати перевагу Java над іншими мовами, такими як Python, Ruby, PHP, Swift, C++ та інші. В результаті знання Java залишається важливим активом для конкуренції на ринку праці.

Java – це інноваційна технологія, яка включає в себе мову програмування та програмну платформу. Для створення програм на Java, спочатку потрібно завантажити Java Development Kit (JDK), який доступний для операційних систем Windows, macOS та Linux. Після написання програми на мові програмування Java, компілятор перетворює її в байт-код Java - набір інструкцій для віртуальної машини Java (JVM), що входить до складу середовища виконання Java (JRE). Байт-код Java може запускатися без змін на будь-якій системі, яка підтримує JVM, дозволяючи вам виконувати ваш код Java де завгодно.

Платформа Java включає в себе JVM, Java API та повноцінне середовище розробки. JVM аналізує та виконує (інтерпретує) байт-код Java. Java API містить розширену бібліотеку функцій, включаючи основні об'єкти, мережеві можливості та функції забезпечення безпеки, а також можливості роботи з розміткою XML та веб-сервісами. Загалом мова програмування Java та програмна платформа Java утворюють потужну та перевірену технологію для розробки корпоративного програмного забезпечення.

Головним аргументом на користь Java для нових корпоративних додатків є її основна ідея – забезпечення сумісності між різними пристроями. Об'єктно-орієнтована структура Java дозволяє створювати модульні програми та використовувати код повторно, що скорочує час розробки та продовжує життєвий цикл корпоративних додатків.

Масштабованість – це ще один ключовий аспект платформи Java. Java дозволяє використовувати одну систему для різних сценаріїв використання.

Існуючі десктопні додатки можна легко адаптувати для роботи на менших пристроях з обмеженими ресурсами. Ви також можете мігрувати додатки з мобільних пристроїв на настільні комп'ютери, створюючи бізнес-додатки для платформи Android і інтегруючи їх в поточне програмне забезпечення настільних комп'ютерів, уникнувши при цьому довгих і витратних циклів розробки. [8]

Java також користується популярністю серед стратегічних планувальників завдяки своїй здатності адаптуватися до нових сценаріїв використання. Наприклад, Java ідеально підходить для розробки додатків для Інтернету речей (IoT). Зазвичай, додатки IoT об'єднують велику кількість різних пристроїв - завдання, яке спрощується завдяки тому, що мільярди пристроїв використовують Java. Крім того, розширена спільнота розробників Java постійно розширює та надає нові бібліотеки, спеціально призначені для розробки додатків IoT.

2.1.2. Бібліотека Spring

Spring Framework, відомий як Spring, є відкритим вихідним фреймворком для розробки програмного забезпечення, який забезпечує інфраструктурну підтримку для створення переважно Java-додатків. Цей фреймворк є одним з найпопулярніших серед фреймворків для Java Enterprise Edition (Java EE) і допомагає розробникам створювати високопродуктивні додатки, використовуючи звичайні об'єкти Java (POJO). Інші поширені фреймворки для розробки на Java включають Java Server Faces (JSF), Maven, Hibernate і Struts.

Spring Framework був випущений у червні 2003 року Родом Джонсоном під ліцензією Apache 2.0 та розміщений на платформі SourceForge.

Програми, розроблені на Java, є складними та містять численні компоненти, зовнішній вигляд та властивості яких залежать від операційної

системи, на якій вони працюють. Spring вважається надійним, вартісним та гнучким фреймворком, який сприяє підвищенню продуктивності у процесі розробки і скороченню загального часу створення додатків завдяки ефективному використанню системних ресурсів.

Spring автоматизує важливий процес налаштування, щоб розробники мали можливість зосередитися на написанні бізнес-логіки. Цей фреймворк керує інфраструктурою, дозволяючи розробникам фокусуватися на реалізації функціональності додатку.

Зазвичай веб-додаток з багаторівневою архітектурою складається з трьох основних рівнів:

Рівень презентації (UI), який відповідає за відображення контенту та взаємодію з користувачем.

Рівень бізнес-логіки, який включає в себе основні функціональні характеристики програми.

Рівень доступу до даних, що керує процедурами отримання даних.

Для правильного функціонування додатка кожен з цих рівнів взаємодіє з іншими. Наприклад, рівень презентації може залежати від рівня бізнес-логіки, який, у свою чергу, взаємодіє з рівнем доступу до даних. Ці взаємодії призводять до створення зв'язків і зазвичай називаються залежностями між різними компонентами системи.

Однією з ключових особливостей Spring є його здатність виконувати ін'єкцію залежностей, що є шаблоном програмування, що дозволяє розробникам створювати більш роз'єднані архітектури. Spring розуміє різні анотації Java, які розробник додає до класів, і може допомогти переконатися, що всі створені екземпляри мають належним чином заповнені залежності.

Для того, щоб Spring Framework міг створювати об'єкти та задовольняти їхні залежності, розробник просто вказує Spring, які класи потрібно керувати та які залежності існують для кожного класу, використовуючи анотації. Наприклад, анотація @Component дозволяє Spring визначити, які класи слід керувати, відзначає об'єкти як управляємі

компоненти та ідентифікує класи, для яких потрібна ін'єкція залежностей. Інший приклад – анотація @Autowired, яка вказує Spring, як обробляти екземпляри класу та починати пошук відповідних залежностей серед компонентів і класів, щоб забезпечити їх встановлення.

Існують декілька ключових термінів, пов'язаних з Spring, які розробники повинні зрозуміти перед тим, як розпочати роботу з цим фреймворком:

Автопідключення: Це процес, за допомогою якого Spring визнає, співставляє та заповнює залежності об'єктів.

Ін'єкція залежностей: Це патерн програмування, який сприяє слабкому зв'язку в коді, що означає, що будь-які зміни в одній частині програми мають меншу ймовірність вплинути на інші частини.

Інверсія управління (IoC): Це принцип проектування, який передає виконання управління від даного класу до окремого фреймворку управління.

IoC-контейнер: Це складова Spring Framework, де об'єкти створюються, з'єднуються, конфігуруються та управляються протягом їх життєвого циклу.

Бін (Bean): Це об'єкт, який створюється та управляється контейнером IoC. Біни становлять основу додатків на основі Spring. [9]

Spring Framework має значну кількість функціональних можливостей, які ретельно структуровані в близько двадцять модулів. Ці модулі можна класифікувати за їхніми основними функціями у такі групи: ядро (Core Container), доступ до даних та інтеграція (Data Access/Integration), веб, аспектно-орієнтоване програмування (AOP), інструментарій (Instrumentation) та тестування (Test). Структура цих груп відображена на наведеній нижче схемі.

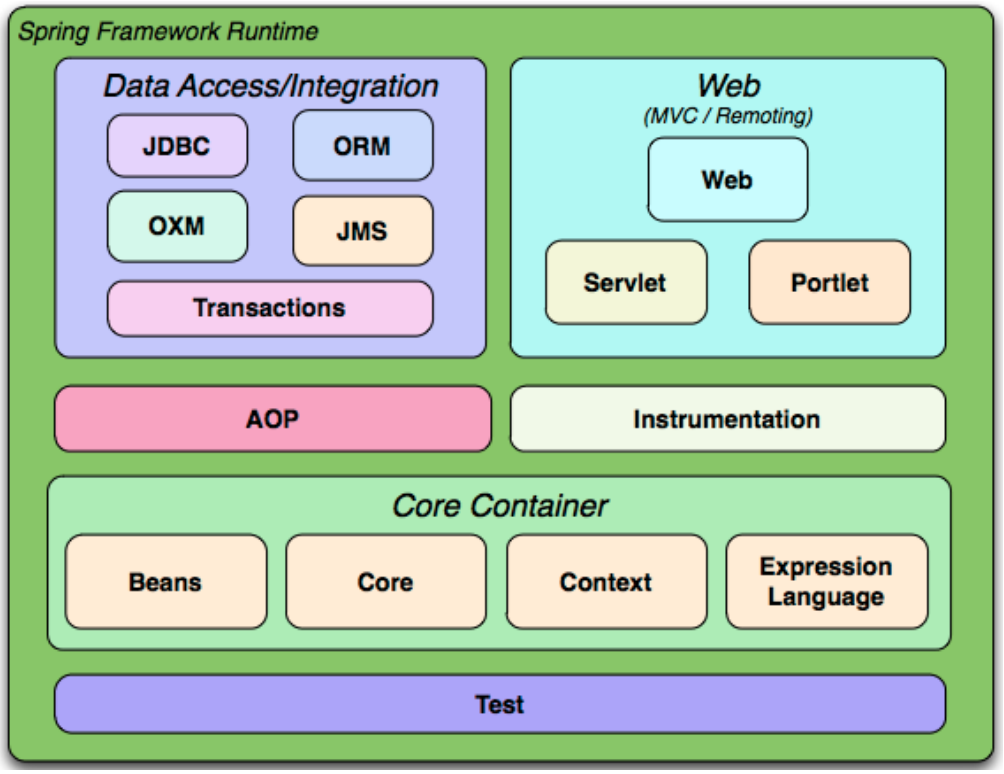

Рис. 2.1. Структура Spring Framework Runtime

Контейнер Core Container складається з модулів Core, Beans, Context та Expression.

Модулі Core та Beans є найбільш фундаментальними складовими фреймворку і відповідають за реалізацію понять IoC та Dependency Injection. Основною ідеєю є використання BeanFactory, яка забезпечує ефективну реалізацію патерну фабрики об'єктів. Це дозволяє уникнути прямої прив'язки до синглетонів у програмному коді і дозволяє відокремити конфігурацію і опис залежностей від фактичної бізнес-логіки додатку.

Модуль Context будується на основі фундаменту, закладеного модулями Core та Beans. Він надає зручний спосіб доступу до об'єктів у стилі фреймворку, який трохи нагадує JNDI-реєстр. Модуль Context успадковує можливості модуля Beans і розширює їх, додаючи підтримку інтернаціоналізації (I18N) (з використанням ресурсних пакетів, наприклад), подій, завантаження ресурсів і прозоре створення контекстів, включаючи

можливість створення контейнера сервлетів. Модуль Context також включає підтримку деяких функцій Java EE, таких як EJB, JMX, і базову можливість віддаленого доступу. Центральним інтерфейсом модуля Context є ApplicationContext, який надає доступ до цих розширених можливостей.

Модуль Expression Language надає потужну мову виразів для формування запитів та маніпулювання об'єктним графом під час виконання програми. Цю мову можна розглядати як розширення уніфікованої мови виразів (Unified EL), яка була визначена в специфікації JSP 2.1. Вона надає можливість встановлювати та отримувати значення властивостей, присвоювати властивості, викликати методи, отримувати доступ до контексту масивів, колекцій та індексаторів, використовувати логічні та арифметичні оператори, працювати з іменованими змінними та отримувати об'єкти за їхніми іменами з IoC-контейнера Spring. Мова також підтримує проекцію та вибірку списків, а також загальні агрегатори для операцій над списками.

Рівень доступу до даних та інтеграції включає модулі JDBC, ORM, OXM, JMS та транзакцій.

Модуль JDBC надає абстракцію рівня JDBC, яка дозволяє уникнути великої кількості рутиної роботи з кодування JDBC та обробки власних помилок, які специфічні для різних постачальників баз даних.

Модуль ORM надає інтеграцію з популярними API об'єктно-реляційного відображення, включаючи JPA, JDO, Hibernate та iBatis. Використовуючи модуль ORM, ви можете поєднувати ці O/R-відображення з іншими можливостями Spring, такими як декларативне керування транзакціями, яке було згадано раніше.

Модуль OXM надає абстракцію для роботи з різними реалізаціями відображення об'єктів в XML. Підтримувані технології включають JAXB, Castor, XMLBeans, JiBX та XStream.

Модуль JMS забезпечує підтримку Spring для служби обміну повідомленнями Java і включає можливості як для створення, так і для споживання повідомлень.

Модуль Transaction забезпечує програмне та декларативне керування транзакціями не лише для класів, які реалізують спеціальні інтерфейси, але також для звичайних об'єктів Java (POJO).

Веб-рівень включає в себе модулі Web, Web-Servlet і Web-Portlet.

Модуль Web Spring надає основні можливості для інтеграції веб-технологій, такі як завантаження багатокомпонентних файлів, ініціалізація контейнера IoC через слухачі сервлетів та контекст веб-додатка. Крім того, він містить компоненти, які підтримують веб-орієнтовану віддалену підтримку Spring.

Модуль Web-Servlet надає реалізацію Spring Model-View-Controller (MVC) для веб-додатків. MVC фреймворк Spring - це не просто стандартна реалізація; він забезпечує чітке розділення між кодом доменної моделі та веб-формами, а також дозволяє використовувати всі інші можливості Spring Framework.

Модуль Web-Portlet надає реалізацію MVC для використання в середовищі портлетів, і він віддзеркалює те, що надається в модулі Web-Servlet.

Модуль AOP Spring пропонує аспектно-орієнтоване програмування (AOP) у відповідності з AOP Alliance. Цей модуль дозволяє вам визначити перехоплювачі методів та точкові вказівники для чіткого відділення коду, який реалізує функціональність, яка повинна бути окремою. Використовуючи метадані на рівні коду, ви також можете включити всі види інформації про поведінку у ваш код, аналогічно до атрибутів в .NET.

Додатково, існує окремий модуль Aspects, який дозволяє інтегрувати засоби AspectJ.

Модуль Instrumentation надає підтримку інструментування класів та реалізує завантажувачі класів для використання у певних серверах додатків.

Модуль Test включає в себе Test Framework, призначений для перевірки Spring-компонентів за допомогою інструментів, таких як JUnit або TestNG. Цей фреймворк забезпечує послідовне створення Spring ApplicationContexts та кешування їх для ефективного тестування. Крім того, в ньому також присутні різноманітні об'єкти-моки, які можуть бути корисні для ізольованого тестування вашого коду в різних тестових сценаріях. [10]

2.1.3. Open Banking API

Відкритий банкінг включає в себе використання API для обміну фінансовою інформацією та послугами з третіми сторонами. Зазвичай, ці треті сторони надають клієнтам банку технології, послуги або програми, які використовують загальну фінансову інформацію та послуги. Ця загальна фінансова інформація включає в себе, наприклад, виписки та записи операцій, які належать клієнтам банку. Важливо зауважити, що ці дані не надаються загальнодоступними і доступними для всіх; вони надаються лише відповідно до конкретних запитів клієнтів. Відкритий банкінг надає технологічну інфраструктуру та правовий каркас для здійснення такого обміну інформацією на підставі згоди всіх зацікавлених сторін.

Як відкритий банкінг функціонує з погляду звичайного користувача? Він забезпечує різноманітні сценарії використання, і ось два з них як приклади.

Ситуація 1 відкритого банкінгу: Кеті та бухгалтерська програма

Кеті – власниця невеликого бізнесу, яка веде облік фінансів за допомогою веб-програми бухгалтерського обліку. До цих пір вона витрачала значну кількість часу на ручне копіювання інформації про фінансові транзакції зі свого онлайн-банкінгу в свою бухгалтерську програму.

Але тепер бухгалтерське програмне забезпечення Кеті підтримує відкритий банкінг. Кеті налаштовує його для синхронізації зі своїм

банківським рахунком під номером 1234. Після того, як вона дає вказівку бухгалтерській програмі звернутися до її банку, щоб отримати доступ до рахунку 1234, банк на всякий випадок запитує у Кеті, чи вона впевнена, що хоче поділитися своїми фінансовими транзакціями з бухгалтерською програмою. Банк також просять Кеті підтвердити свою особу, увійшовши до свого банківського облікового запису цифрово. Після цього одноразового налаштування дані про транзакції регулярно надходять від банку до бухгалтерської програми Кеті, і її бухгалтерське програмне забезпечення завжди автоматично оновлюється.

Сценарій використання 2 відкритого банкінгу: Сінді та іпотека

Сінді шукає ідеальний будинок, і коли вона нарешті знаходить його, їй потрібно отримати негайне підтвердження іпотеки, щоб укласти угоду з продавцем.

Спеціалізована фінансова технологічна компанія XYZ пропонує іпотеку з низькою ставкою. Щоб переконатися, що вона є досяжною для Сінді, фінтех-компанія перевіряє її можливість отримання іпотеки на основі історії транзакцій на її банківському рахунку. Фінтех-компанія запитує дані про транзакції у банку Сінді. Банк запитує у Сінді підтвердження її особи та згоду на обмін даними. Після цього банк передає дані фінтех-компанії для обробки. Все це займає всього кілька секунд, і фінтех-компанія може надати миттєвий результат Сінді.

Фінансові установи спираються на довіру, і для банків надзвичайно важливо, щоб їх клієнти мали довіру до них. Банки повинні забезпечити безпечне зберігання грошей клієнтів, правильну обробку транзакцій та захист особистих даних. При впровадженні цифрових технологій, таких як відкритий банкінг, це повинно сприяти покращенню взаємодії з клієнтами і створенню активної цифрової екосистеми з великим вибором програм та цифрових рішень. Однак, коли банки впроваджують відкриті банківські послуги, дуже важливо, щоб основні технології відкритого банкінгу допомагали збудувати, підтримувати і зміцнювати довіру клієнтів до свого

банку. Які технології необхідні для участі в відкритих банківських екосистемах і як ці технології можуть бути використані для встановлення довіри з партнерами та клієнтами?

Коли банки приєднуються до відкритих банківських екосистем, їм потрібно опанувати різні цифрові технології, зокрема API. API використовуються для обміну фінансовими даними у відкритій банківській екосистемі. З правильним використанням API та супутніми технологіями, ви можете викликати довіру як надійного учасника екосистеми серед партнерів та клієнтів.

Партнери: Для забезпечення довіри з партнерами важливо відповідно захищати API та дані, які передаються, дотримуючись найкращих практик; розробляти API відповідно до встановлених специфікацій і галузевих стандартів; і створювати гладкий досвід інтеграції.

Клієнти: Щоб встановити довіру з клієнтами, банки повинні виконувати їх навчання; показувати, що клієнти мають контроль над своїми даними; запитувати згоду клієнта на будь-який обмін даними або транзакції, ініційовані API; і створювати зручний досвід користувача для клієнтів.

Традиційно банки зосереджуються на строгому контролі над своїми продуктами та способами їх поставки, однак сучасні клієнти наразі все більше прагнуть розширеного вибору.

Клієнти бажають мати доступ до банківських продуктів у зручний для них час та спосіб, враховуючи їхні індивідуальні вподобання. Виявляється, що відповідь на ці потреби за допомогою традиційної закритої банківської структури є вкрай складною задачею. [13]

Зазвичай, банки структурують свою діяльність навколо концепцій продуктів і каналів. Продукти включають, наприклад, кредити, рахунки та платежі, а канали – це способи доставки цих продуктів клієнтам. Це призводить до створення простої трьохрівневої архітектури, відомої як банківський стек.

Це означає, що банки створюють свої продукти та розповсюджують їх через власний канал. Цей підхід надає банкам жорсткий контроль над своїм банківським асортиментом. На схемі нижче можна візуалізувати рівень контролю, де помаранчевий колір позначає область контролю банку, тоді як чорний - це зовнішня сфера його впливу.

Часто продукти і канали з'єднані на рівні IT (наприклад, великі моноліти), але вони також мають важливий зв'язок на рівні бізнесу. Володіння каналом дозволяє контролювати інтерфейс клієнта та надає банкам можливість здійснювати перехресний продаж продуктів зі свого портфеля. Володіння цифровими продуктами, окрім каналу, забезпечує жорсткий контроль над ланцюжком створення вартості.

API – це, мовляв, засувка-блискавка між каналом і продуктом, яка вводить удосконалену структуру та розділення між каналом і продуктом. Проте поки блискавка залишається зачиненою, дві її половини – канал і продукт – нерозривно сполучені. Це підкреслює важливість внутрішніх API, які забезпечують поліпшену внутрішню структуру. Справжня цінність засувки-блискавки полягає в її можливості відкриватися, що приводить нас до наступного етапу.

Після створення внутрішніх API настав час відкрити зв'язок між каналом і продуктом, і це відомо як відкритий банкінг. Це робить банківський продукт, найнижчий рівень в банківській ієрархії, доступним через API навіть для зовнішніх третіх сторін. Це дозволяє зовнішнім сторонам приносити внесок в банківський стек різними способами, що має серйозні наслідки для бізнесу. Нові бізнес-моделі стають можливими, коли продукти банку можуть розповсюджуватися через сторонні канали, або коли продукти третіх сторін легко розповсюджувати серед власних клієнтів, можливо, для заповнення прогалин в продуктовому портфелі.

Звісно, відкритий банківський сервіс включає в себе більше, ніж лише технічні аспекти внутрішніх API, оскільки відкриття API між каналом і

продуктом для третіх сторін вимагає юридичного, договірного або, в деяких випадках, навіть нормативного контексту.

З цією структурою і зрозумілим інтерфейсом для різних компонентів банківського стеку банки можуть:

1. Відкрити свій стек і запропонувати сумісні рішення від третіх сторін.

2. Легко інтегрувати рішення сторонніх розробників в свій банківський стек як зі сторони каналу, так і зі сторони продукту.

3. Продовжувати розробляти власні банківські продукти та поширювати їх через фінтех-рішення.

4. Або стати розповсюджувачами продуктів і послуг, створених фінтех-компаніями.

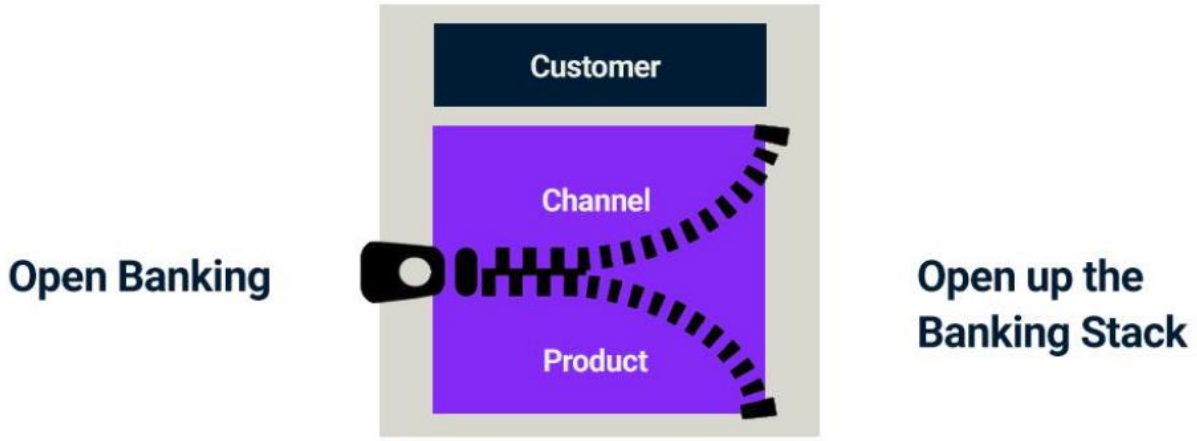

Рис. 2.2. Вигляд API з точки зору клієнта, канала і продукта

2.2. Дослідження засобів для роботи з даними

2.2.1. REST API

Існує різноманітність видів API, що створює труднощі для новачків у розробці, які намагаються розрізнити кожен з них. Один із цих видів, відомий

як Representational State Transfer (REST), представляє собою архітектурний стиль програмного забезпечення, який розробники використовують для створення веб-інтерфейсів. REST API забезпечують прості та стандартизовані інтерфейси, що дозволяють отримувати доступ до даних, вмісту, алгоритмів, медіа та інших цифрових ресурсів через веб-адреси. Фактично кажучи, REST API є найбільш поширеними видами API, що активно використовуються в Інтернеті сьогодні.

Для досягнення статусу RESTful API необхідно дотримуватись шести основних обмежень:

Уніфікований інтерфейс: Для забезпечення уніфікованого інтерфейсу потрібно застосовувати численні архітектурні обмеження, які регулюють взаємодію компонентів. Крім того, ресурси повинні бути ідентифіковані унікальними URL-адресами.

Клієнт-серверний підхід: Використання уніфікованого інтерфейсу розділяє відповідальність між клієнтом і сервером. Клієнтська сторона стосується інтерфейсу та формування запитів, в той час як серверна сторона відповідає за доступ до даних, управління робочим навантаженням та безпеку. Це розділення дозволяє незалежно розвивати і вдосконалювати обидва компоненти.

Операції без стану: Всі необхідні дані для розуміння і обробки запиту повинні бути включені в нього. Сервер не повинен зберігати інформацію про стан клієнта.

Кешування ресурсів: Відповіді на запити повинні містити мітки для позначення ресурсів, які можна кешувати або не можна.

Багаторівнева система: REST допускає створення ієрархічних структур, в яких кожен компонент не має доступу до рівня вище або нижче в ієрархії, ніж його власний рівень.

Код на вимогу: REST API дозволяє завантажувати та виконувати код на вимогу, такий як аплети або скрипти. Сервери можуть повертати статичні

представлення ресурсів у форматі XML або JSON, а також виконуваний код для клієнта, якщо це потрібно.

Для отримання розуміння того, як працюють REST API, важливо визначити поняття "ресурси". Ресурсом може бути будь-яка інформація, наприклад, документ, зображення, колекція інших ресурсів або навіть абстрактний об'єкт. REST використовує ідентифікатор ресурсу для ідентифікації конкретного об'єкта, що бере участь у взаємодії між компонентами. [11]

Метод – це тип запиту, який ви надсилаєте серверу. Існують чотири основних методи доступу до ресурсів, які пов'язані з REST API:

GET: Цей метод дозволяє серверу знайти та надіслати вам дані, які ви запитали.

PUT: Використання методу 'PUT' дозволяє серверу оновити запис у базі даних.

POST: Цей метод дозволяє серверу створити новий запис у базі даних.

DELETE: Використання методу 'DELETE' дозволяє серверу видалити запис із бази даних.

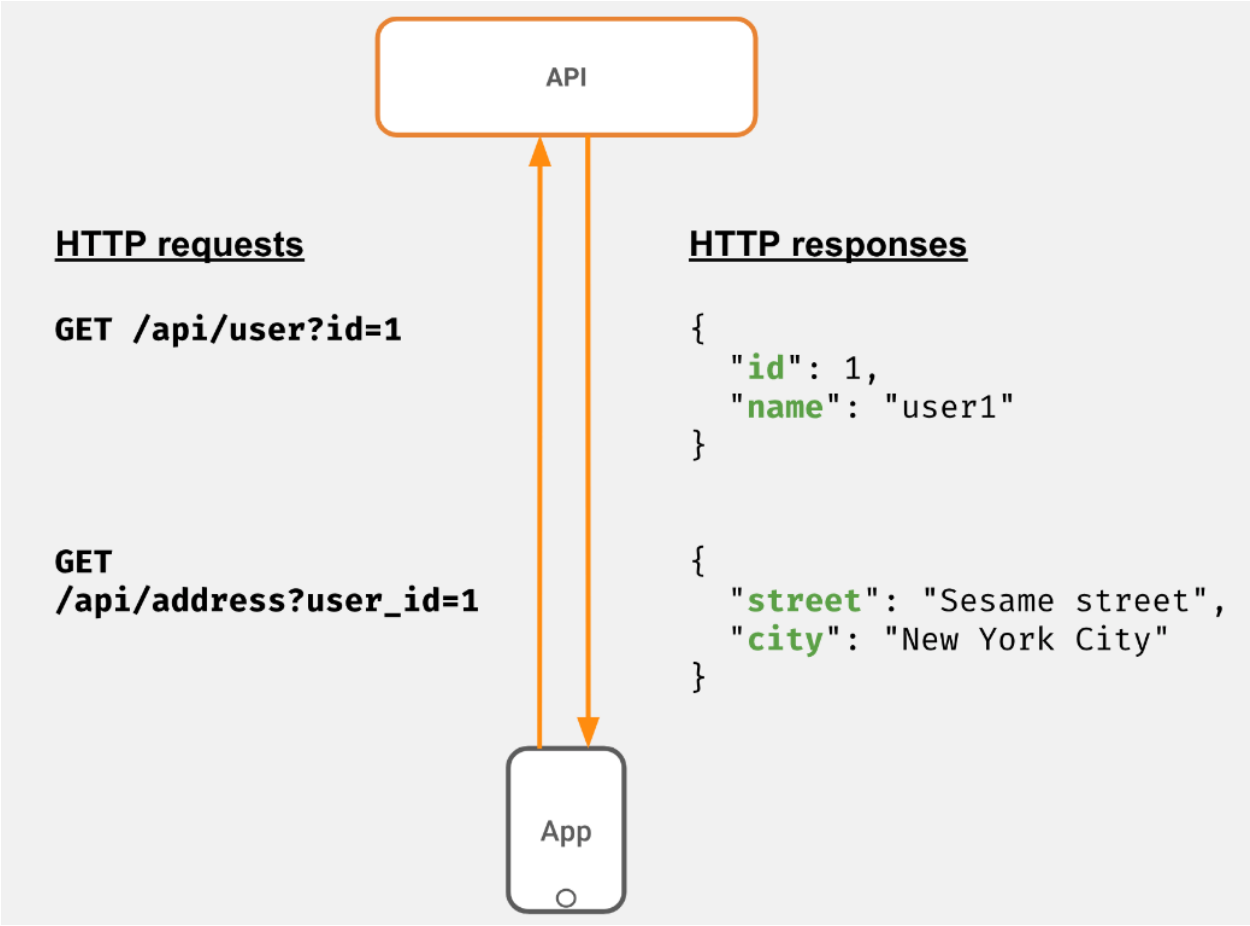

Рис. 2.3. Принцип роботи REST API

2.2.2. MongoDB

MongoDB – це система управління базами даних (СУБД) із відкритим вихідним кодом, яка спеціалізується на документно-орієнтованому підході до зберігання даних. Вона призначена для ефективного зберігання великих обсягів інформації. MongoDB відноситься до класу NoSQL баз даних, що означає, що дані в ній не зберігаються у формі табличних структур, як це зазвичай властиво реляційним базам даних.

Розробка та управління MongoDB здійснюється компанією MongoDB, Inc. за ліцензією SSPL (Server Side Public License). Ця СУБД була запущена в лютому 2009 року та підтримує офіційні драйвери для багатьох популярних мов програмування, включаючи C, C++, C#, .Net, Go, Java, Node.js, Perl, PHP,

Python, Motor, Ruby, Scala та Swift. Тому ви маєте можливість розробляти додатки на вибір за допомогою будь-якої з цих мов.

На сьогодні MongoDB активно використовується великою кількістю компаній, таких як Facebook, Nokia, eBay, Adobe, Google і багато інших, для зберігання та управління великими обсягами даних.

Розглянемо процес роботи MongoDB під поверхнею. MongoDB функціонує як сервер баз даних, де дані зберігаються. Для пояснення, середовище MongoDB надає вам можливість запустити сервер та створити кілька баз даних за його допомогою.

Оскільки MongoDB є базою даних типу NoSQL, інформація зберігається у вигляді колекцій і документів. Таким чином, база даних, колекції та документи взаємодіють між собою, як показано нижче:

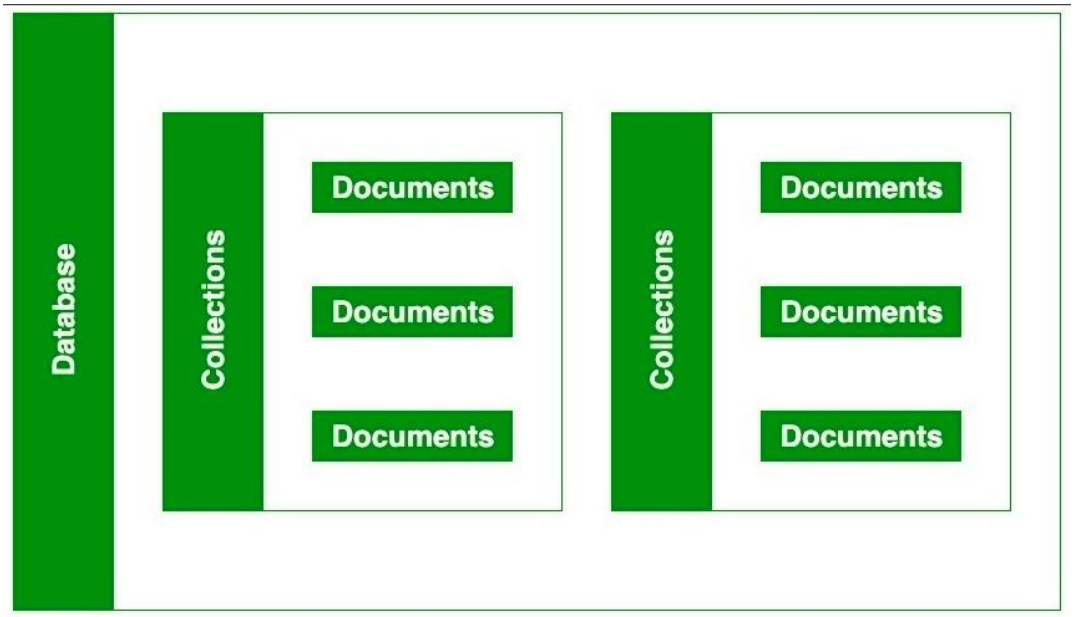

Рис. 2.4. Взаємодія елементів в MongoDB

База даних MongoDB містить колекції, подібно до того, як у базі даних MySQL є таблиці. У MongoDB можна створити кілька баз даних та кілька колекцій.

Кожна колекція містить документи, які містять дані для зберігання в MongoDB. Одна колекція може містити багато документів, і вони не обов'язково мають однаковий формат, оскільки MongoDB не вимагає суворих схем.

Документи створюються за допомогою полів, що представляють собою пари ключ-значення, подібні до стовпців у традиційних реляційних базах даних. Значення полів можуть бути різних типів BSON, таких як числа з рухомою комою, рядки, булеві значення і т. д.

MongoDB використовує формат документів BSON для зберігання даних. BSON відображається як "Binary JSON" і є бінарним представленням даних JSON. Сервер MongoDB перетворює дані JSON у формат BSON для більш ефективного зберігання та обробки.

MongoDB дозволяє вкладати дані в документи, що дозволяє створювати складні взаємозв'язки між даними та зберігати їх в одному документі. Це робить операції з даними дуже ефективними порівняно з SQL, де для отримання даних з різних таблиць потрібно писати складні SQL-запити. Треба враховувати, що максимальний розмір BSON-документа - 16 МБ. [12]

Зауваження: MongoDB Server дозволяє вам запускати багато баз даних.

Наприклад, уявімо, що у нас є база даних з назвою "Library". У цій базі даних містяться дві колекції, і в межах кожної колекції ми можемо знайти два документи. В кожному з цих документів ми зберігаємо наші дані за допомогою полів, як це показано на зображенні нижче:

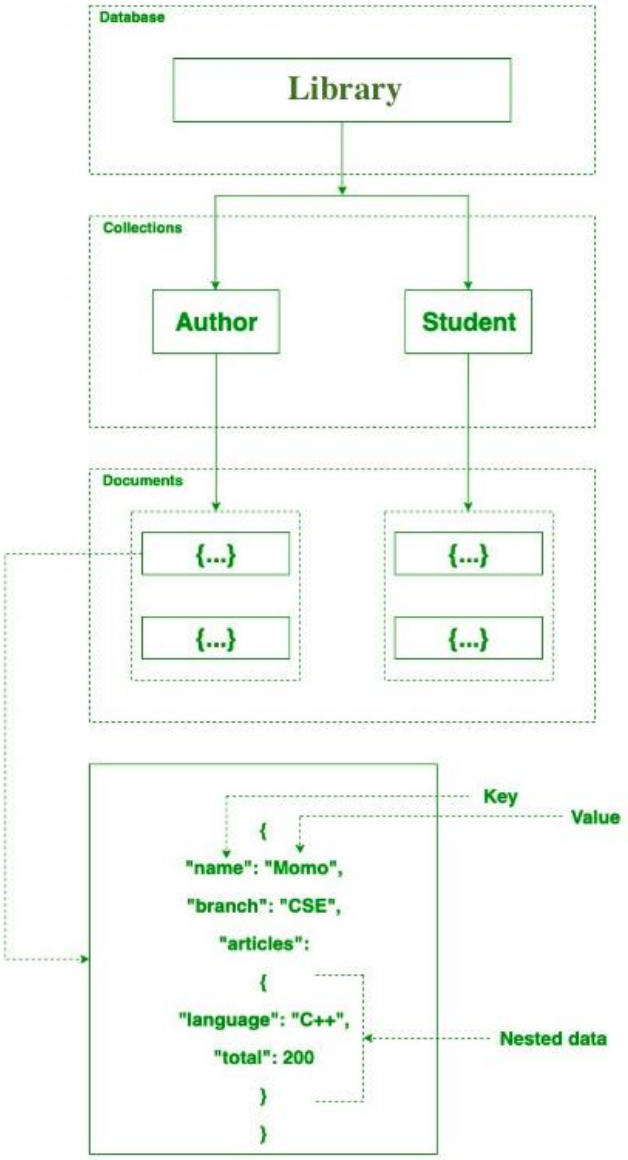

Рис. 2.5. Структура бази даних MongoDB

Характеристики MongoDB:

Гнучка схема бази даних: MongoDB дозволяє мати базу даних без жорсткої схеми. Це означає, що в одній колекції можуть бути збережені різні типи документів, і ці документи можуть містити різну кількість полів та даних. У відміну від реляційних баз даних, MongoDB не вимагає, щоб всі документи в колекції були однаковою структури. Ця гнучкість дозволяє працювати з даними більш адаптивно.

Орієнтована на документи: Всі дані в MongoDB зберігаються у документах, які мають пари ключ-значення. Відмінно від реляційних баз даних, де дані зберігаються в таблицях з рядками і стовпцями, MongoDB використовує документи для представлення даних. Кожен документ також має унікальний ідентифікатор об'єкта, який служить для його ідентифікації.

Індексування: MongoDB надає можливість індексувати кожне поле в документах за допомогою первинних та вторинних індексів. Це полегшує швидкий доступ до даних та виконання запитів. Без індексів база даних шукає дані за запитом, перевіряючи кожен документ, що може призвести до великих затрат часу та ресурсів.

Масштабованість: MongoDB дозволяє горизонтально масштабувати систему за допомогою шардингу. Цей підхід передбачає розподіл даних на кілька серверів, де великий обсяг даних розділяється на фрагменти за допомогою ключа шарду. Ці фрагменти рівномірно розподіляються між різними шардами, розташованими на різних фізичних серверах. Це дозволяє додавати нові сервери до робочої бази даних для підтримки зростання обсягу даних.

Реплікація: MongoDB забезпечує високий рівень доступності та надійності завдяки можливості реплікації. В цьому випадку створюються декілька копій даних, які відправляються на інші сервери. Це означає, що якщо один сервер виходить з ладу, то дані можна отримати з інших серверів, що забезпечує неперервну роботу системи.

Агрегація: MongoDB надає можливість виконувати операції над згрупованими даними і отримувати один результат або обчислений результат. Це схоже на операції GROUP BY в SQL. MongoDB пропонує три різні методи для агрегації даних: конвеєр агрегації, функцію map-reduce та одноразові методи агрегації.

Висока продуктивність: MongoDB славиться високою продуктивністю і стійкістю даних. Це досягається завдяки різноманітним функціям, таким як

масштабування, індексування, реплікація та інші. MongoDB дозволяє обробляти великий обсяг даних ефективно та швидко.

2.3. Аналіз технологій для створення клінтської частини додатка

2.3.1. JavaScript programming language

JavaScript – це легка, кросплатформова, інтерпретована мова програмування, яку також відомо як мова сценаріїв для веб-сторінок. Вона відома своєю широкою використаністю в розробці веб-сайтів та застосунків. JavaScript є мовою, що підтримує слабке типізування (динамічну типізацію), і її можна використовувати як для клієнтської, так і для серверної розробки. Вона підтримує імперативний та декларативний підходи. JavaScript включає стандартну бібліотеку об'єктів, таких як масиви, роботу з датами та математичні функції, а також основні мовні конструкції, такі як оператори, керуючі структури та інші мовні елементи.

На стороні клієнта: забезпечує об'єкти для управління браузером та об'єктною моделлю документа (DOM). Наприклад, клієнтські розширення дозволяють додатку додавати елементи на HTML-сторінку та реагувати на події користувача, такі як клацання мишею, введення даних у форму та навігація по сторінці. Для розробки клієнтської частини існують корисні бібліотеки, такі як AngularJS, ReactJS, VueJS та багато інших.

Серверна частина: забезпечує об'єкти, необхідні для виконання JavaScript на сервері. Розширення на стороні сервера дозволяють додатку взаємодіяти з базою даних і забезпечують постійність інформації між викликами додатку або взаємодію з файлами на сервері. Один з найвідоміших фреймворків, який використовується сьогодні, - це Node.js.

Імперативна мова – в цьому типі мови більше уваги приділяється тому, як саме виконується завдання. Ця мова просто контролює послідовність

обчислень і включає в себе процедурний підхід до програмування, а також об'єктно-орієнтований підхід, оскільки в режимі очікування асинхронного виконання ми розглядаємо, що робити після виклику асинхронної операції.

Декларативне програмування – в цьому типі мови основна увага зосереджена на тому, як отримати бажаний результат, замість того, як саме це робити. Головна мета – описати бажаний результат без прямих інструкцій щодо виконання, як це робиться, наприклад, у функції зі стрілкою. [14]

JavaScript може бути включений до HTML-файлу двома методами:

1. Внутрішній JavaScript: Ми можемо додати JavaScript безпосередньо до нашого HTML-файлу, вставивши код всередину тегу <script>. Тег <script> може розміщуватися як всередині тегу <head>, так і всередині тегу <body> відповідно до потреб.

2. Зовнішній JavaScript: Ми можемо написати JavaScript-код у відокремленому файлі з розширенням .js, а потім підключити цей файл до HTML-файлу, вставивши його всередину тегу <head> того HTML-файлу, в який ми хочемо додати цей код.

Синтаксис:

```
<script>
  // JavaScript Code
</script>
```

Згідно з нещодавніми результатами опитування, проведеного на Stack Overflow, JavaScript визнано найпопулярнішою мовою програмування на планеті.

З розвитком технологій браузерів і переходом JavaScript на сервер, завдяки таким інструментам, як Node.js та інші фреймворки, JavaScript виявився набагато потужнішим. Ось деякі можливості, які надає JavaScript:

JavaScript був винайдений перш за все для взаємодії з Document Object Model (DOM). Перед появою JavaScript веб-сайти були в основному статичними, але завдяки JS стали можливими динамічні веб-сайти.

У JavaScript функції є об'єктами, які можуть мати властивості і методи, подібно до інших об'єктів. Функції можна передавати як аргументи в інші функції.

JavaScript може працювати з датою і часом.

Він також може виконувати перевірку форм, навіть якщо форми створюються за допомогою HTML.

Однією з важливих особливостей JavaScript є те, що для його використання не потрібен компілятор.

JavaScript вважається легким завдяки декільком основним причинам. Він вимагає низького обсягу процесорного часу, має простий та мінімалістичний синтаксис та відсутність строгих типів даних. У JavaScript все розглядається як об'єкт, що спрощує його вивчення, особливо для тих, хто знайомий із синтаксисом C++ або Java.

Ця легка мова не перевантажує процесор, а також не створює великого навантаження на операційну пам'ять. JavaScript може працювати у браузері, навіть зі складними парадигмами та логікою, і при цьому споживає менше ресурсів, ніж багато інших мов. Наприклад, Node.js, що є варіантом JavaScript, не тільки виконує обчислення швидше, але й ефективно використовує ресурси порівняно з аналогами, такими як Dart або Java.

Крім того, JavaScript має обмежений набір вбудованих бібліотек та фреймворків порівняно з іншими мовами програмування, що може бути однією з причин його легкості. Проте ця обмеженість також може вимагати використання зовнішніх бібліотек та фреймворків.

2.3.2. Бібліотека REACT

React.js – це відкритий JavaScript фреймворк та бібліотека, розроблена Facebook, яка використовується для швидкого створення інтерактивних веб-

додатків та інтерфейсів з меншою кількістю коду порівняно зі стандартним JavaScript.

У React ви створюєте ваші додатки, розбиваючи їх на багаторазові компоненти, які можна уявити як незалежні будівельні блоки. Ці компоненти представляють окремі частини кінцевого інтерфейсу, і разом вони формують повну картину користувацького додатку.

Роль React у додатку полягає в обробці рівня представлення, подібно до "виду" у патерні модель-вид-контролер (MVC), забезпечуючи ефективний рендеринг. Замість того, щоб розглядати весь інтерфейс як одне ціле, React заохочує розробників ділити його на окремі компоненти, що спрощує побудову і модифікацію інтерфейсу. Фреймворк ReactJS об'єднує швидкість та ефективність JavaScript і надає більш ефективний спосіб роботи з DOM для швидшого рендерингу сторінок та створення динамічних та адаптивних веб-додатків.

Зазвичай ви відкриваєте веб-сторінку, введуючи URL-адресу в своєму веб-браузері. Ваш браузер відправляє запит на сервер, щоб отримати цю веб-сторінку, і потім відображає її. Якщо ви переходите за посиланням на іншу сторінку веб-сайту, браузер надсилає новий запит на сервер, щоб отримати цю нову сторінку.

Цей процес завантаження між вашим браузером (клієнтом) і сервером повторюється для кожної нової сторінки або ресурсу, до якого ви намагаєтеся отримати доступ на веб-сайті. Такий підхід до завантаження веб-сайтів працює добре, але є ситуації, коли це не є оптимальним. Для веб-сайтів, які залежать від великої кількості даних, повторне завантаження сторінок може призвести до незручностей для користувачів.

Крім того, при зміні даних у традиційних JavaScript-додатках, необхідно вручну оновлювати DOM, щоб відобразити зміни. Вам потрібно визначити, які дані змінилися, і власноруч внести ці зміни до DOM. Це може призвести до перезавантаження всієї сторінки.

React використовує інший підхід, дозволяючи створювати односторінкові додатки (SPA). Односторінкова програма завантажує лише один HTML-документ під час першого запиту. Потім вона оновлює лише певні частини веб-сторінки, такі як вміст або тіло, за допомогою JavaScript.

Цей підхід відомий як клієнтська маршрутизація, оскільки клієнту не потрібно перезавантажувати всю веб-сторінку, щоб отримати нову сторінку при кожному новому запиті. Замість цього React перехоплює запит і змінює ті частини, які потребують оновлення, без повного перезавантаження сторінки. Цей підхід забезпечує кращу продуктивність і більш динамічний досвід користувача.

React використовує віртуальний DOM, який є копією фактичного DOM. Віртуальний DOM React миттєво оновлюється, щоб відобразити зміни стану даних, і порівнює його з реальним DOM, щоб визначити, які саме зміни відбулися. [15]

На основі цього React визначає, як найкраще оновити реальний DOM шляхом впровадження цих змін без повного його перетворення. В результаті компоненти та інтерфейс React дуже швидко відображають зміни, оскільки не потребують повного перезавантаження сторінки при кожному оновленні.

У відміну від інших фреймворків, таких як Angular, React не нав’язує жорстких обмежень стосовно структури коду або організації файлів. Це означає, що розробники та команди можуть вільно встановлювати правила та організовувати проект так, як вони вважають за потрібне. Завдяки такій гнучкості React дозволяє використовувати його в будь-якому обсязі та застосовувати його так, як вимагають конкретні завдання.

React може бути використаний для створення окремої кнопки, кількох елементів інтерфейсу або навіть усього користувацького інтерфейсу програми. Ви можете поступово інтегрувати його у вже існуючий проект, надаючи йому інтерактивність, або створювати повноцінні потужні додатки на React з нуля, в залежності від ваших конкретних потреб.

2.4. Огляд використаних середовищ розробки

Середовища розробки є важливими інструментами для успішного створення та управління проектами. Вони надають розробникам зручні та контрольовані умови для створення, тестування та розгортання програмних продуктів. Ось деякі ключові аспекти, для чого потрібні середовища розробки для проектів:

1. Розробка програмного забезпечення: Середовища розробки надають зручні інструменти для створення коду програми, включаючи текстовий редактор, компілятори, інтерпретатори, та інші необхідні ресурси.

2. Тестування: Розробники використовують середовища для проведення тестів, перевірки роботи програми та виявлення помилок. Вони можуть створювати тестові набори, налагоджувати код та перевіряти, чи функції працюють належним чином.

3. Керування версіями: Системи керування версіями, такі як Git, інтегруються в середовища розробки для відстеження змін в коді, спрощення спільної роботи та відновлення попередніх версій програми.

4. Відлагодження: Середовища розробки дозволяють розробникам знайти та виправити помилки в коді за допомогою інструментів відлагодження, таких як точки зупинки, відстеження змін змінних та виведення діагностичних повідомлень.

5. Підтримка мов програмування: Різні мови програмування вимагають спеціалізованих середовищ для розробки. Середовища надають інструменти для розробки коду на конкретній мові, включаючи автодоповнення, відстеження синтаксису та інші.

6. Розгортання та управління проектом: Середовища розробки можуть включати інструменти для автоматизації розгортання програми на серверах або хмарних платформах. Вони також надають можливість керування завданнями та проектами.

7. Спільна робота: Завдяки середовищам розробки команди можуть спільно працювати над проектами, вносячи зміни до спільного репозиторію та вирішуючи конфлікти.

8. Зручність та продуктивність: Відповідно до потреб розробників, середовища розробки можуть налаштовуватися, надавати шаблони, гарячі клавіші та інші зручності для підвищення продуктивності.

Узагальнюючи, середовища розробки надають розробникам інфраструктуру та інструменти для створення, тестування та управління програмними проектами, що важливо для розробки високоякісного програмного забезпечення.

2.4.1. Середовище розробки Intellij Idea

Для розробки серверної частини додатку було використано Intellij Idea.

IntelliJ є однією з найвпливовіших та найпопулярніших інтегрованих середовищ розробки (IDE) для мови програмування Java. Цю потужну IDE розроблено та підтримує компанія JetBrains, і вона доступна у двох різних виданнях: Community та Ultimate. IntelliJ надає багато можливостей для швидкого розробки та покращення якості коду.

IDE вказує на інтегроване середовище розробки, що означає поєднання кількох інструментів, які спрощують розробку програмного забезпечення, роблять її більш надійною та менш схильною до помилок. Порівняно зі звичайним текстовим редактором, IDE має наступні переваги:

Інтеграція із корисними інструментами, такими як компілятор, інструменти налагодження, система контролю версій, засоби збірки, різноманітні фреймворки, інструменти для оптимізації коду тощо.

Здатність навігації в коді, автодоповнення коду, проведення рефакторингу та генерація коду, що прискорюють процес розробки.

Підтримка модульного, інтеграційного та покриття коду тестування за допомогою різних плагінів.

Велика кількість доступних плагінів, які дозволяють розширити функціональність IDE та додаткові можливості.

IntelliJ IDEA володіє кількома найефективнішими функціями завершення коду в мові Java. Її передбачальний алгоритм може точно передбачити, який код розробник намагається ввести і автоматично завершити його, навіть у випадку, якщо розробник не знає точну назву конкретного класу, члена або іншого ресурсу.

IntelliJ IDEA дійсно розуміє ваш код та контекст розробки, і це розуміння робить його унікальним серед інших інтегрованих середовищ розробки для Java.

Розумне завершення коду – ця функція пропонує найбільш відповідні символи, які можуть бути використані в поточному контексті.

Ланцюжкове завершення коду – це розширена функція завершення коду, яка надає список доступних символів через методи або геттери в поточному контексті.

Завершення статичного члена – це дозволяє використовувати статичні методи або константи та автоматично додавати необхідні оператори імпорту для уникнення помилок компіляції.

Виявлення дублікатів – швидко виявляє продубльовані фрагменти коду та надсилає повідомлення або пропозиції користувачеві.

Перевірки та швидкі виправлення – коли IntelliJ виявляє можливі помилки, у вас з'являється маленька лампочка в тому ж рядку. Натиснувши на неї, ви отримуєте список пропозицій. [16]

IntelliJ IDEA дотримується принципу забезпечення розробникам можливості кодувати без відволікань, і це обумовлено тим, що на екрані зазвичай видно лише редактор, зі спеціальними ярликами для доступу до інших функцій, не пов'язаних з програмуванням.

Ця орієнтація на редактор означає, що ви можете легко перевіряти додаткову інформацію, не виходячи з поточного контексту завдяки швидким спливаючим вікнам.

Також IntelliJ IDEA пропонує великий набір комбінацій клавіш для швидкого доступу до різних функцій, включаючи перемикання між вікнами інструментів.

Плюс до всього цього, у вас є вбудований налагоджувач, який дозволяє вам відлагоджувати вашу програму безпосередньо в самому середовищі розробки.

IntelliJ IDEA надає широкий спектр інструментів, включаючи контроль версій, інструменти збирання, виконувача тестів та засіб покриття коду для популярних тестових фреймворків, декомпілятор для класів Java, термінал, інструменти бази даних, підтримку Docker і багато інших корисних функцій.

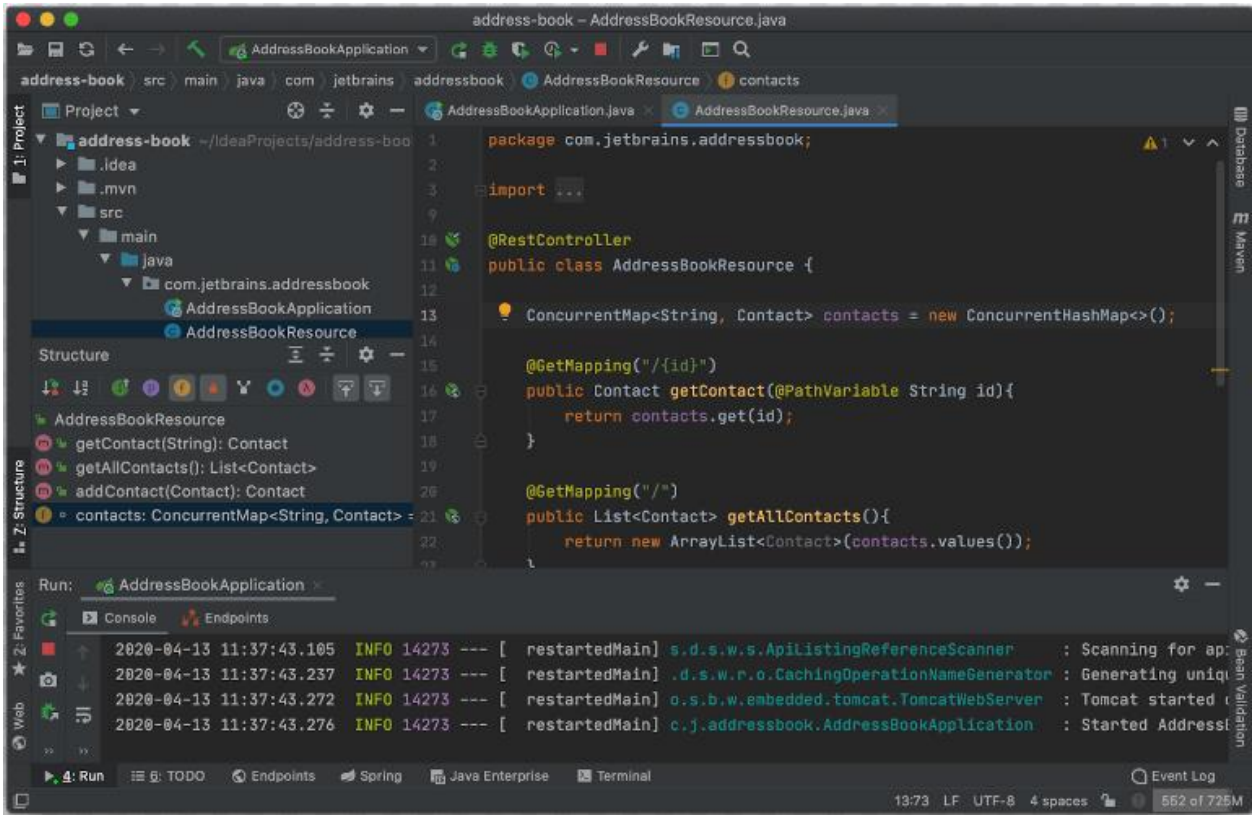

Рис. 2.6. Інтерфейс IntelliJ IDEA

2.4.2. Середовище розробки Visual Studio Code

Для розробки клієнтської частини додатка, найкращим вибором було використовувати Visual Studio Code.

Visual Studio Code – це поєднання простого редактора вихідного коду та потужних інструментів розробника, таких як IntelliSense для завершення коду та налагодження.

На перший погляд, це просто редактор, який спрощує вам життя. Простий цикл редагування, компіляції та налагодження означає менше витрат часу на налаштування середовища та більше часу на втілення ваших ідей. Visual Studio Code має швидкий редактор вихідного коду, ідеально підходить для щоденного використання, з підтримкою сотень мов програмування. Він надає підсвічування синтаксису, автоматичний відступ, інтуїтивно зрозумілі комбінації клавіш та можливість налаштування для зручного кодування.

Для більш серйозної розробки вам потрібні інструменти, які розуміють код глибше. Visual Studio Code пропонує вбудовану підтримку IntelliSense для автоматичного завершення коду, розширеного розуміння семантичного коду та можливості навігації та рефакторингу коду.

І, якщо робота над кодом стає важчою, налагодження може допомогти вам розібратися. Visual Studio Code включає інтерактивний налагоджувач, який дозволяє вам крок за кроком переглядати код, перевіряти змінні, оглядати стеки викликів і виконувати команди в консолі.

Окрім цього, VS Code інтегрується з інструментами збірки та сценаріїв для автоматизації різних завдань, що значно полегшує щоденний робочий процес. Він також підтримує Git, що дозволяє вам працювати з керуванням версіями прямо з редактора, включаючи перегляд змін та подачу їх на розгляд. [17]

Visual Studio Code надає розширену вбудовану підтримку розробки на Node.js, використовуючи як JavaScript, так і TypeScript, заснований на тих же

основних технологіях, які використовуються в Visual Studio. У VS Code також існують відмінні інструменти для веб-технологій, таких як JSX/React, HTML, CSS, SCSS, Less і JSON.

З архітектурної точки зору Visual Studio Code комбінує в собі найкраще з веб-технологій, нативних рішень і мовних технологій. За допомогою Electron, VS Code поєднує в собі веб-технології, такі як JavaScript і Node.js, із швидкістю та гнучкістю нативних додатків. Платформа VS Code використовує покращену версію того самого редактора на основі HTML, який був розроблений для використання у хмарному редакторі "Monaco", і включає компоненти, використані в інструментах Internet Explorer F12 та інших проектах. Крім цього, VS Code має архітектуру служби інструментів, яка дозволяє інтегрувати його з різними технологіями, що використовуються у Visual Studio, включаючи Roslyn для .NET, TypeScript та систему налагодження Visual Studio, і багато інших.

Visual Studio Code також пропонує загальнодоступну модель розширень, яка дозволяє розробникам створювати та використовувати розширення, а також налаштовувати свій власний досвід редагування, збирання та налагодження.

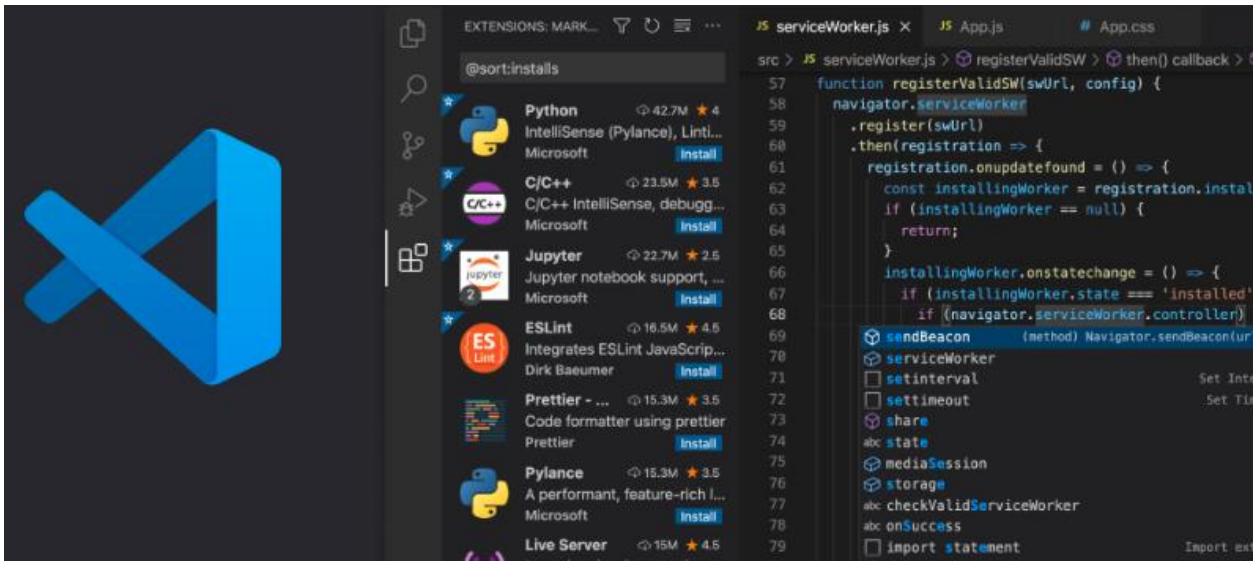

Рис. 2.7. Зовнішній вигляд Visual Studio Code

ВИСНОВОК ДО РОЗДІЛУ 2

На основі аналізу технологій для створення серверної та клієнтської частини додатку, а також середовищ розробки, можна зробити наступний висновок:

Для створення серверної частини додатку було обрано мову програмування Java та бібліотеку Spring. Ця комбінація надає надійність та широкі можливості для створення веб-додатку. Крім того, використання Open Banking API дозволить взаємодіяти з банківськими даними та послугами. Для роботи з даними використовуватимуться REST API та база даних MongoDB. REST API забезпечить стандартизовану взаємодію між клієнтською та серверною частинами, а MongoDB дозволить зберігати та керувати даними ефективно.

Для розробки клієнтської частини додатку обрано мову програмування JavaScript та бібліотеку React. JavaScript є широко використовуваною мовою для розробки веб-додатків, а React допоможе створити користувацький інтерфейс взаємодії з серверною частиною додатку.

Зазначені технології та інструменти обрані з урахуванням їхньої придатності для завдань проекту та можливості покращити продуктивність розробки. Такий вибір допоможе створити надійний та ефективний додаток для досягнення поставлених цілей.

РОЗДІЛ 3

РОЗРОБКА ДОДАТКУ

3.1. Створення календаря з подіями

В даному підрозділі буде описано як створити одих з головних відображуваних елементів додатку – календар, в який можна додавати особисті чи корпоративні події.

Для створення багатофункціонального календаря буде використано плагін для React Big Calendar.

React Big Calendar – це популярна бібліотека з відкритим вихідним кодом для створення компонентів календаря в додатках, що базуються на React. Вона надає гнучкий та налаштований спосіб відображення подій та графіку в форматі календаря. Ця бібліотека часто використовується для створення додатків, які працюють із подіями, систем назначень або будь-яких додатків, де необхідно візуальне відображення даних, пов'язаних з часом.

Основні можливості React Big Calendar включають:

- Налаштованість: React Big Calendar дозволяє легко налаштовувати вигляд та поведінку календаря. Ви можете визначати власні стилі подій, вспливаючі підказки та інші візуальні елементи.
- Підтримка перетягування та випадання: Користувачі можуть взаємодіяти з подіями за допомогою дій перетягування та випадання, що робить інтуїтивно зрозумілим оновлення дат та часу подій.

- Види за місяць, тиждень та день: Календар підтримує різні види, включаючи місяць, тиждень та день, що надає користувачам різні перспективи на їх графік.
- Змінювані події: Події можна змінювати взаємодією з користувачем, надаючи зручний спосіб змінювати тривалість подій.
- Вбудовані види та панель інструментів: Бібліотека постачається з вбудованими видами навігації (місяць, тиждень, день) та панеллю інструментів для легкої навігації між різними періодами часу.
- Локалізація: React Big Calendar підтримує локалізацію, що дозволяє відображати інформацію про дати та час різними мовами та форматами.
- Інтеграція з React: Оскільки React Big Calendar розроблено спеціально для додатків на React, вона безпроблемно інтегрується з компонентами React та слідує архітектурі компонентів React. [21]

Для того, щоб використовувати React Big Calendar потрібно встановити плагін за допомогою команди: *npm install —save react-big-calendar*. *Npm install moment* – команда допоможе перевіряти, маніпулювати та форматувати дати.

Спочатку потрібно створити файл у папці проекту *src* з кодом, який представлений нижче.

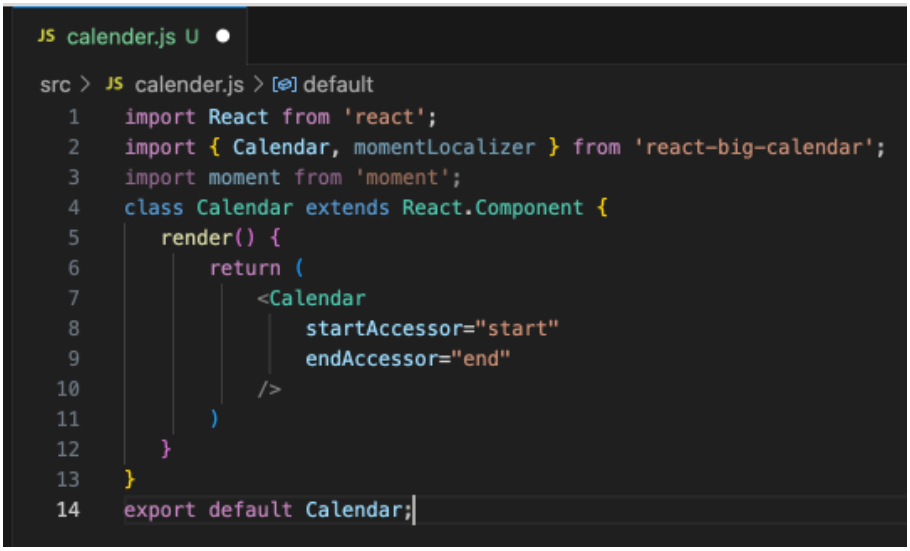

Рис. 3.1. Фрагмент js коду з основними властивостями React Big Calendar

React Big Calendar пропонує широкий спектр можливостей через свої реквізити, і ми використовуємо певні з них. У попередньому фрагменті коду ми вже використали дві основні властивості:

- „startAccessor”: ця властивість визначає дату та час початку події.
- „endAccessor”: ця властивість визначає кінцеву дату та час події.

Різні типи подій повинні відображатися різним кольором. Наприклад, особисті події підображаються синім кольором, а робочі – червоним.

Ми потребуємо функцію, яка обробляє відповідь API GET для отримання списку свят. Ця функція буде відображати робочі події "червоним" кольором на календарі. Крім того, вона буде обробляти список листів, що були створені користувачем, і відображатиме їх у календарі кольором за замовчуванням. Наразі ми реалізуємо обидві ці функції у календарі. [22]

На даному етапі ми зосередимось на функції render(), де будемо встановлювати дані для обох видів свят і відповідно заповнювати списки, які використовуються в атрибутах великого календаря React.

Великий календар React надає деякі змінні подій за замовчуванням, перелічені нижче.

Event {

title: string,

start: Date,

end: Date,

allDay?: boolean

resource?: any,

}

Ми налаштуємо робочі події та інформацію щодо особистих подій відповідно до вказаної вище змінної події. У цьому переліку ми об'єднаємо наші дані щодо особистих та робочих подій. Тепер прийшов час простим способом впровадити зв'язані дані щодо свят і кодів листів у Календарі.

```
function() {
    const holidays = []
this.state.holidaysList.forEach((holiday) => {
    let start = moment(holiday.for_date).toDate()
    holidays.push({ id: holiday.id, title: holiday.occasion,
    start: start, end: start, color: holiday.color, resource: holiday.is_restricted, type: 'holiday', allDay: 'true' })
})
const leaves = []
this.state.absentiesList.forEach((leave) => {
    let start_at = (new Date(leave.start_at))
    let end_at = (new Date(leave.end_at))
    leaves.push({ id: leave.id, title: leave.username, start: start_at, end: end_at, color: leave.color, type: 'leave', allDay: 'true' })
})
const list = [...holidays, ...leaves]
return (
 <Calendar
    events={list}
    startAccessor="start"
    endAccessor="end"
    defaultDate={moment().toDate()}
    eventPropGetter={event => {
        const eventData = list.find(ot => ot.id === event.id);
        const backgroundColor = eventData && eventData.color;
        return { style: { backgroundColor } };
    }}
```

Рис. 3.2. Фрагмент основного коду для відображення календаря

Ми включили три нові параметри у події календаря:

- events: Цей параметр пов'язує список подій.
- eventPropGetter: Додатково визначає функцію, яка повертає об'єкт className або атрибути стилю для застосування до вузла події.

- defaultDate: Тут ми використовуємо moment().toDate(), що забезпечує поточну дату.

Готовий вигляд календаря можна побачити нижче. Представлено 3 формата: місячний, тижневий і денний.

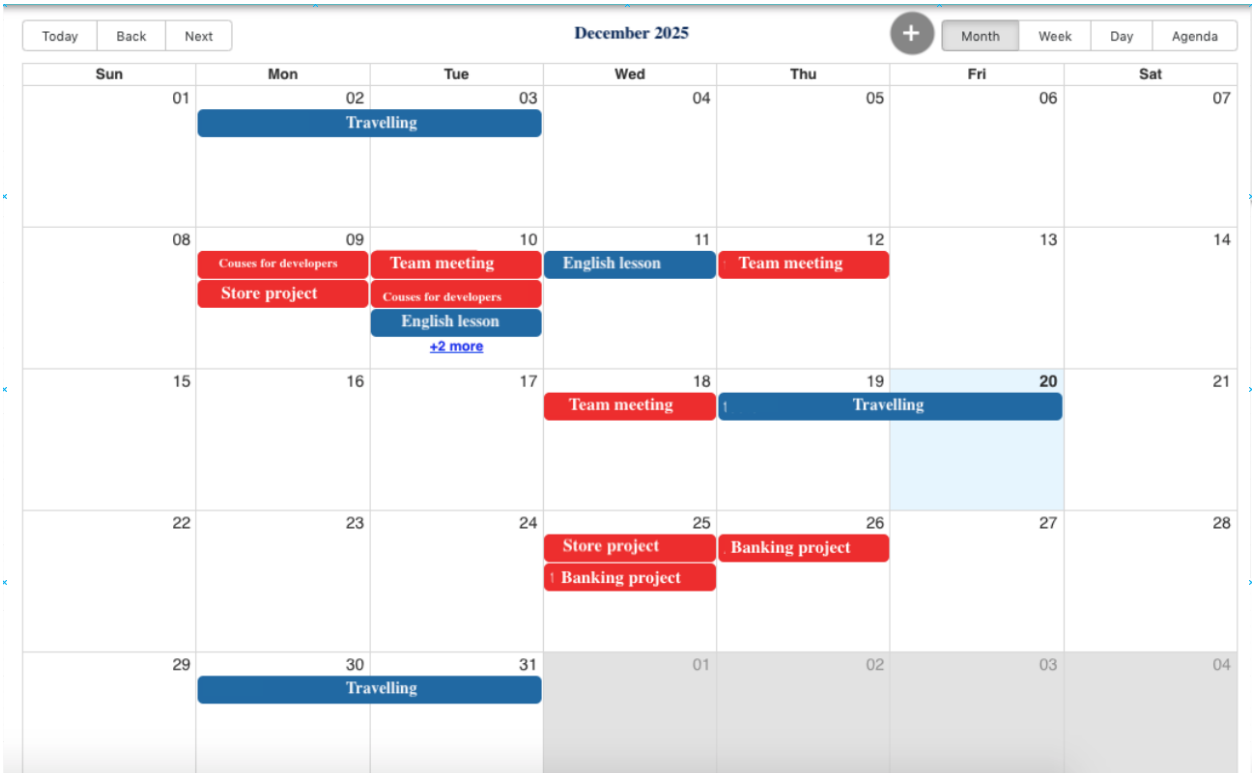

Рис. 3.3. Місячний формат календаря

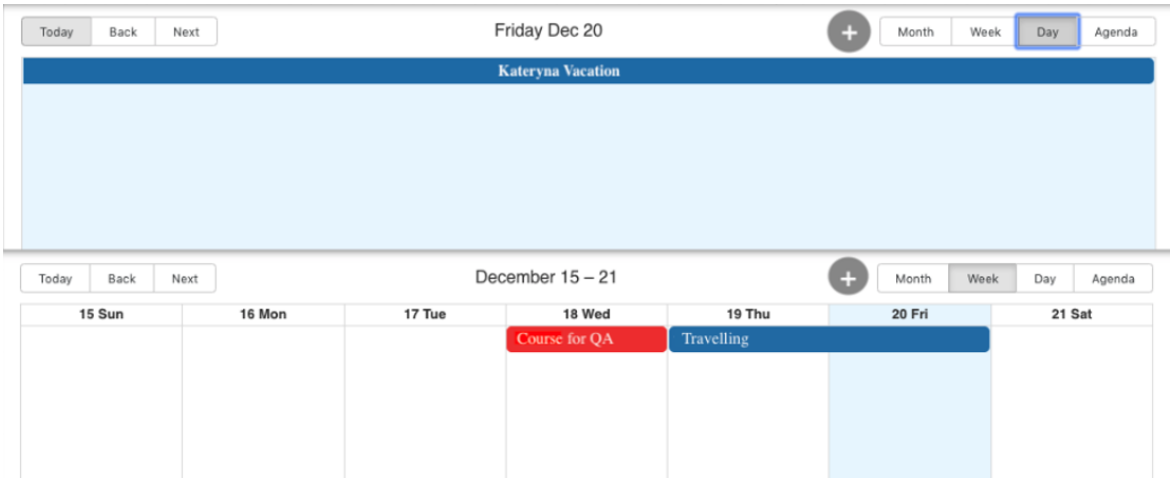

Рис. 3.4. Денний та тижневий формат календаря

3.2. Імплементація доступу до банківського рахунку

Наступним кроком є написання логіки для можливості мати доступ до банківського рахунку одразу з додатку. Це можливо за допомогою різних сервісів і відкритого API. В цій роботі було використано Stripe – інтернет-сервіс, який надає можливість компаніям і приватним особам отримувати платежі онлайн. Він також пропонує клієнтські бібліотеки (JavaScript і мобільні застосунки) і серверні бібліотеки (Java, Ruby, Node.js і т.д.).

Stripe готує обширні набори бібліотек для розробників, які відповідають за зовнішнє та бек-енд виконання оплат. Серед функцій цих бібліотек можна відзначити наявність об'єктів, які забезпечують:

- пройдіть процедуру автентифікації за допомогою стандартного або підключеного облікового запису Stripe;
- зручно надсилайте різноманітні запити до Stripe, такі як створення платежу, запити на повернення коштів, перекази та інші, і аналізуйте відповіді, що надходять;
- обробляйте веб-хуки для отримання сповіщень про різні події, такі як платежі, виплати, повернення коштів і т.д. Кожна зазначена подія в обліковому записі викликає відправку запиту від Stripe до зазначеного раніше маршруту на вашому сервері. Це дозволяє вашому додатку отримувати повідомлення про будь-які зміни, які ви хочете слідкувати. [24]

Stripe також надає компоненти інтерфейсу користувача, які можна легко імпортувати безпосередньо в ваш інтерфейс. Як видно в демонстраційній програмі, ми будемо використовувати готове до використання поле введення від Stripe для управління введенням номерів кредитних карток.

Управління платежами є складним завданням через обробку важливих даних користувачів, а саме їхніх номерів кредитних карток. Повністю обробку кредитних карток відзначає система Stripe. Таким чином, наша програма має менші вимоги до безпеки та відповідності. Платежі здійснюються безпосередньо через інтерфейс до Stripe API. Проте ми б хотіли вести відстеження всієї історії платежів, що пройшли через нашу платформу. Таким чином, потрібен механізм, який дозволяє нам одночасно зберігати платіжну інформацію в нашій базі даних, не передаючи при цьому конфіденційні дані на серверну частину.

На щастя, Stripe володіє системою "Намірів оплати", що дає нам можливість досягти цього.

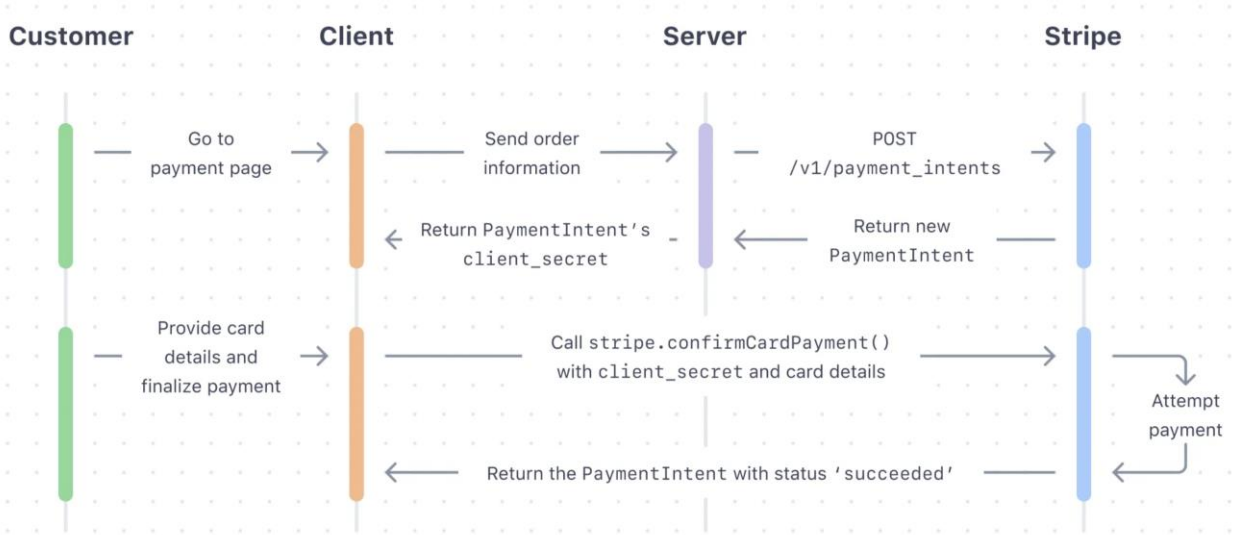

Рис. 3.5. Принцип роботи Stripe

На рисунку вище проілюстровано основний принцип роботи Stripe:

Крок 1: Клієнтська програма або зовнішній інтерфейс надсилає запит до серверної частини з повідомленням, що користувач готовий оплатити певну суму за конкретне замовлення. Запит містить всю необхідну інформацію про замовлення, таку як ціна та деталі товару.

Крок 2: Сервер викликає API Stripe для створення "Наміру платежу". Stripe генерує цей намір і передає його назад на серверну частину.

Крок 3: Сервер повертає `client_secret`, який міститься в об'єкті "Payment Intent", до зовнішнього інтерфейсу. Цей код дозволяє Stripe ідентифікувати, що платіж, здійснений через цього клієнта, відповідає наміру платежу, що був створений відповідним сервером.

Крок 4: Клієнт виконує оплату, користуючись інструментами розробки інтерфейсу Stripe. Запит на платіж, включаючи `client_secret`, надсилається до Stripe для виконання необхідних операцій.

Крок 5: Клієнту повертається статус із Stripe. [26]

Для реалізації серверної частини, нам потрібно написати рівень контроллера, серверний рівень та клас POJO для взаємодії java сервіса з базою даних.

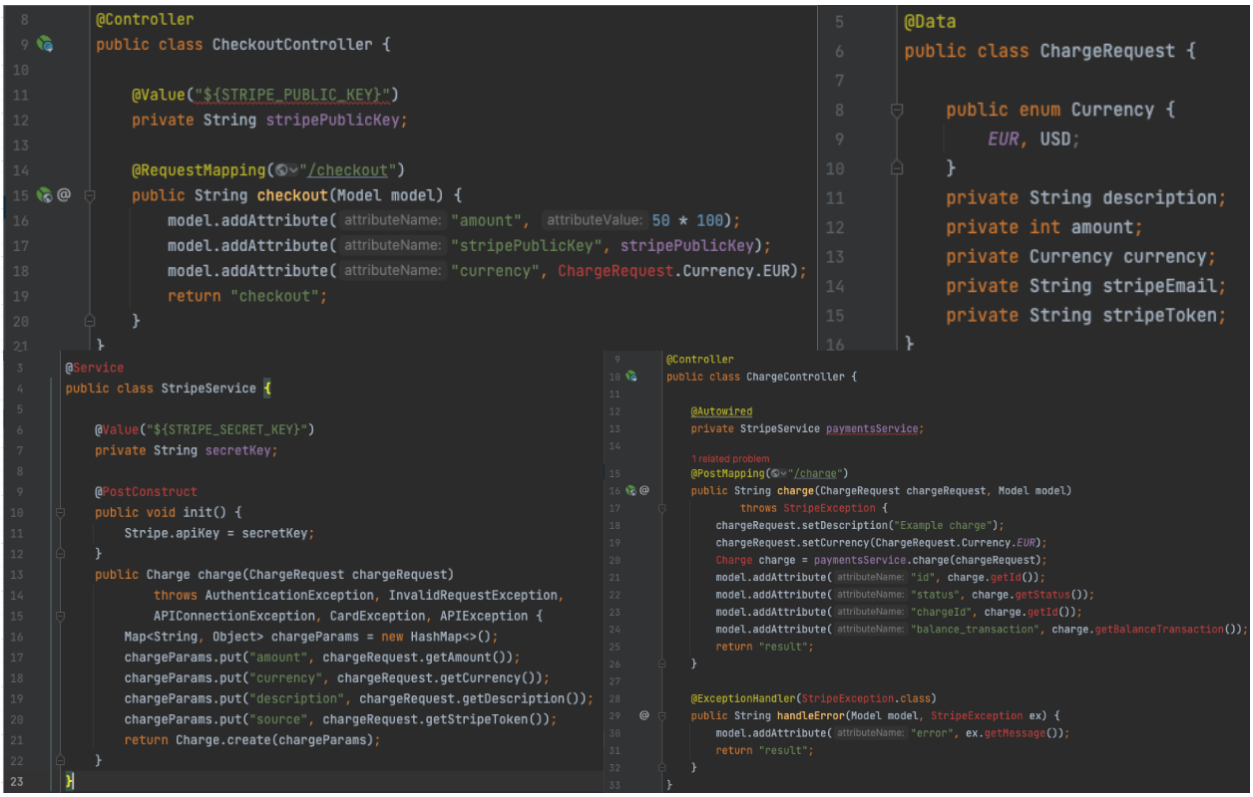

Рис. 3.6. Реалізація доступу до банківського рахунку на java

Розпочнемо зі створення контролера для підготовки моделі із необхідною інформацією для форми оформлення замовлення. Спочатку скопіюємо тестову версію нашого відкритого ключа з інформаційної панелі Stripe та визначимо STRIPE_PUBLIC_KEY як змінну середовища. Потім використаємо це значення у полі `stripePublicKey`. Також ми встановимо валюту та суму тут вручну лише для демонстрації. У реальній програмі ми можемо встановити ідентифікатор продукту/розпродажу, який можна використовувати для отримання фактичних значень. Щодо API-ключів Stripe, ви можете визначити їх як змінні середовища для кожної програми. Як і у випадку з будь-яким паролем чи конфіденційною інформацією, найкраще утримувати секретний ключ подалі від системи контролю версій.

Для обробки на серверній стороні нам необхідно визначити обробник запиту POST, який використовується формою оформлення замовлення.

Клас StripeService використовується для виконання фактичної операції заряду в Stripe. Як показано у CheckoutController, поле `secretKey` заповнюється значенням зі змінної середовища STRIPE_SECRET_KEY, яке ми скопіювали з інформаційної панелі Stripe. Після ініціалізації служби цей ключ використовується для усіх наступних операцій у Stripe. Об'єкт, який повертається бібліотекою Stripe, представляє операцію заряду і містить корисні дані, такі як ідентифікатор операції. [25]

Потрібно створити контролер, який прийматиме POST-запит від форми оформлення замовлення та відправлятиме платіж до Stripe через наш StripeService. Важливо відзначити, що параметр "ChargeRequest" автоматично ініціалізується на основі параметрів запиту, таких як "amount", "stripeEmail" і "stripeToken", які надсилаються з форми.

При успішній операції ми додаємо статус, ідентифікатор операції, ідентифікатор стягнення та ідентифікатор транзакції балансу до моделі, щоб подальше відображення їх користувачеві.

ExceptionHandler взаємодіє з винятками типу StripeException, які можуть виникнути під час операції заряджання. Якщо необхідно більше

деталей при обробці помилок, можна додати окремі обробники для підкласів StripeException, таких як CardException, RateLimitException або AuthenticationException.

На рисунку нижче можна побачити клієнтську частину додатку. Зображено обліковий запис користувача та секцію, в яку можна додати особисті банківські картки та слідкувати за кількістю грошей, які доступні на рахунку.

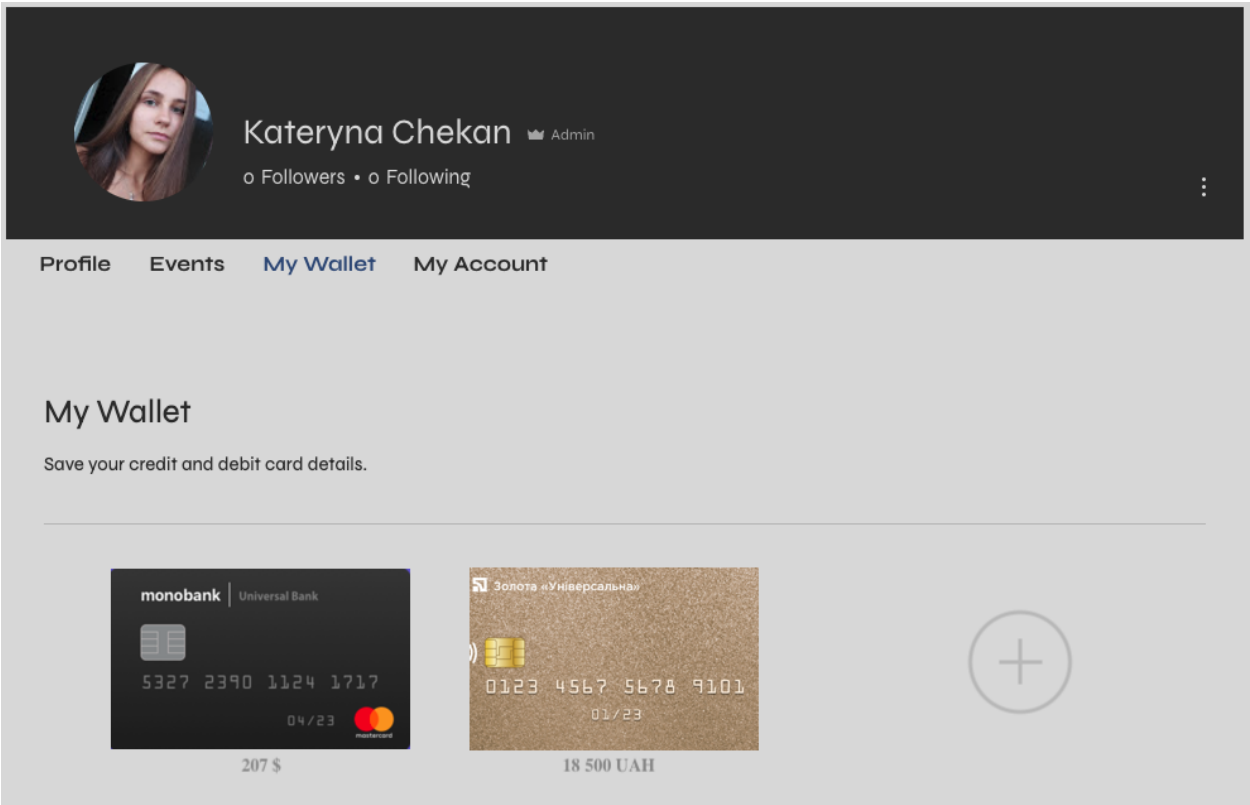

Рис. 3.7. Вигляд сторінки з банківськими картами користувача

3.3. Розробка сховища даних для проєкту

Для маштабування додатка потрібно зберігати дані користувачів у базі даних. Для цього було обрано нереляційне сховище MongoDB. Щоб спроситити встановлення та конфігурація бази даних на комп'ютер, було вирішено використовувати Docker Container.

3.3.1. Використання Docker Container

Контейнер Docker представляє собою автономний блок, який включає програму та всі необхідні конфігурації та залежності. Можна уявити його як великий zip-файл, що містить усе необхідне для запуску програми на будь-якій операційній системі або обладнанні. Docker служить інструментом для управління та запуску цих контейнерів. Контейнери забезпечують ізоляцію всього, що необхідно для виконання програми в її власному середовищі. Docker впровадив цей концепт, представивши Docker Engine, що дозволяє контейнерам працювати в різних операційних системах. Благодаря цьому двигуну контейнер може функціонувати однаково ефективно в Linux, macOS та Windows.

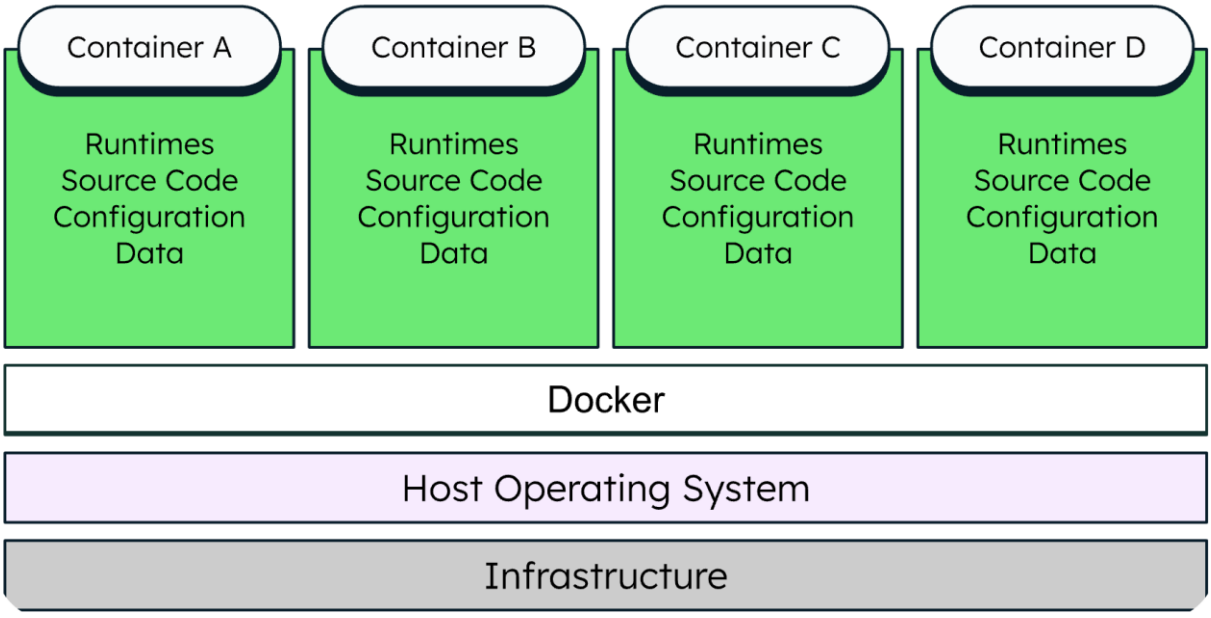

Рис. 3.8. Представлення структури механізму Docker

Контейнери операційні власними ізольованими процесами у визначеному просторі користувача. Це має перевагу значного зменшення

ресурсів, необхідних для роботи, порівняно з віртуальними машинами, які вимагають власної повної операційної системи. Таким чином, загальна вартість запуску програми у формі контейнера суттєво менша, ніж її виконання на віртуальній машині. [28]

Існує багато способів використання MongoDB у контейнерному середовищі. Обов'язково потрібно встановити Docker Desktop для роботи. Інструкції щодо встановлення можна знайти на їх веб-сайті.

Для запуску контейнера MongoDB за допомогою Docker потрібно використати наступну команду:

export MONGODB_VERSION=6.0-ubi8

docker run --name mongodb -d mongodb/mongodb-community-server:$MONGODB_VERSION

Ця команда запускає сервер MongoDB, керований версією 6.0, на базі Red Hat UBI у відокремленому режимі (у фоновому процесі). Рекомендується використовувати тег для конкретизації версії MongoDB для забезпечення узгодженості. Якщо вам необхідний доступ до сервера MongoDB з іншої локальної програми, вам потрібно відкрити порт за допомогою параметру -p.

docker run --name mongodb -d -p 27017:27017 mongodb/mongodb-community-server:$MONGODB_VERSION

Використовуючи цей спосіб, ви зможете підключитися до свого екземпляра MongoDB за адресою mongodb://localhost:27017. Можна керувати цим, скориставшись Compass, графічним інтерфейсом користувача MongoDB для візуалізації та аналізу ваших даних.

Будь-які дані, що були створені під час життєвого циклу цього контейнера, будуть видалені після його зупинки та видалення. Це можна здійснити за допомогою команд Docker stop та rm.

docker stop mongodb && docker rm mongodb

Якщо ви прагнете зберегти дані на своєму локальному комп'ютері, ви можете приєднати том, скориставшись параметром -v.

docker run --name mongodb -d -p 27017:27017 -v $(pwd)/data:/data/db mongodb/mongodb-community-server:$MONGODB_VERSION

Якщо ви зупините та перезапустите контейнер, усі введені раніше дані залишаться незмінними.

3.3.1. Реалізація доступу до бази даних через REST API

Для розробки з Spring Boot та базою даних MongoDB нам необхідні наступні API:

1. Spring Data MongoDB
2. Spring Boot

Існують два підходи до підключення до бази даних MongoDB - використання MongoRepository і MongoTemplate.

Для цього проекту ми використовуємо систему збірки Maven, і ось бібліотеки, які були включені в залежності:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

Спочатку потрібно створити POJO класи, які слугують відображенням структури бази даних.

POJO (Plain Old Java Object) – це клас у Java, який відповідає простим принципам об'єктно-орієнтованого програмування і не залежить від конкретного фреймворку чи специфікації. Основна ідея POJO полягає в тому, щоб мати простий, стандартний Java-клас без додаткових вимог чи обов'язкових елементів, які ставилися б конкретними фреймворками чи бібліотеками. [27]

Нижче наведено основні POJO класи додатку: User, Address, Card, Event.

```java
@Document
@Data
public class Address {

    @Id
    private String id;
    private String country;
    private String street;
    private String house;
    private String postcode;

}
```

```java
@Document
@Data
public class Event {

    @Id
    private String id;
    private String name;
    private String description;
    private String eventType;
    private Address address;
    private float expenseAmount;
    private Date startDate;
    private Date endDate;
    private Date creationDate = new Da
}
```

```java
@Document
@Data
public class User {

    @Id
    private String id;
    private String firstName;
    private String lastName;
    private int age;
    private String email;
    private char[] password;
    private String phoneNumber;
    private String mainPhoto;
    private Address address;
    private List<Event> events;
    private List<Card> bankingCards;
```

```java
@Document
@Data
public class Card {

    @Id
    private String id;
    private String number;
    private String balance;
    private String image;
    private String bank;

}
```

Рис. 3.9. POJO – класи

Потрібно звернутися до репозиторію Spring Data MongoDB для отримання доступу до наших даних. MongoRepository від Spring Data надає нам загальні функції, які можна легко інтегрувати та використовувати. Нижче визначено інтерфейс для нашого сховища. Також потрібно створити рівень контролеру, через який користувач може отримати доступ до сервісної частини за допомогою REST API. На рисунку показано як створити метод для отримання данних користувача по ID.

```java
@Repository
public interface UserRepository extends MongoRepository<User, String> {
}
```

```java
@RestController
@RequestMapping(value = "/users")
public class UserController {

    private final Logger LOG = LoggerFactory.getLogger(getClass());

    private final UserRepository userRepository;

    public UserController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @PreAuthorize("hasRole('ROLE_USER')")
    @RequestMapping(value = "/{userId}", method = RequestMethod.GET)
    public User getUser(@PathVariable String userId) {
        LOG.info("Getting user with ID: {}.", userId);
        return userRepository.findOne(userId);
    }

}
```

Рис. 3.10. Реалізація можливості доступу до бази даних

Для того, щоб отримати результат користувача по ідентифікатору, клієнту потрібно мати роль ‘ROLE_USER’ та використовуючи HHTP метод GET викликати даний URL: *GET* *http://localhost:8080/users/ID1*.

Результатом буде слугувати відповідь з статус кодом 200 і форматом JSON. JSON (JavaScript Object Notation) – це легкий формат обміну даними, який є текстовим та незалежним від мови програмування. Він використовується для представлення структурованих даних та передачі їх між комп'ютерами. JSON є популярним форматом для взаємодії між веб-серверами та клієнтами, а також для збереження та обміну даними в багатьох програмних середовищах. Вигляд відповіді серверу:


```
{
    "id":"eacde886-6cf2-43dd-b766-0a3cc50fff62",
    "firstName":"Kateryna",
    "lastName":"Chekan",
    "age":22,
    "email":"chekankate@gmail.com",
    "phoneNumber":"1234567890",
    "mainPhoto":"main.jpg",
```

```
"address":{
        "id":"414617fe-5116-41a0-8f1b-352ba2536e7b",
        "country":"Ukraine",
        "street":"Shevchenka",
        "house":"34c",
        "postcode":"111-111"
        },
"events":[
{
                "id":"cb62f855-9ec9-4beb-9e52-eda4bb182322",
                "name":"Development Course",
                "description":"Course for Java Developers. Duration is 4 weeks.",
                "eventType":"WORKING",
                "address":{
                        "id":"842564dd-5a38-4fa4-973f-0dfbfadb0487",
                        "location":"ZOOM"
                        },
                "expenseAmount":150,
                "startDate":"2025-10-12",
                "endDate":"2025-11-12"
                }
],
"bankingCards":[{
                "id":"163ee173-dc18-4d22-91a3-8dc3f2bfbe4a",
                "number":"1234 1234 1234 1234",
                "balance":15000,
                "currency":"UAH"
                }
]
}
```


Всю інформацію користувач заповнює через клієнтську частину додатка за допомогою різних форм для введення.

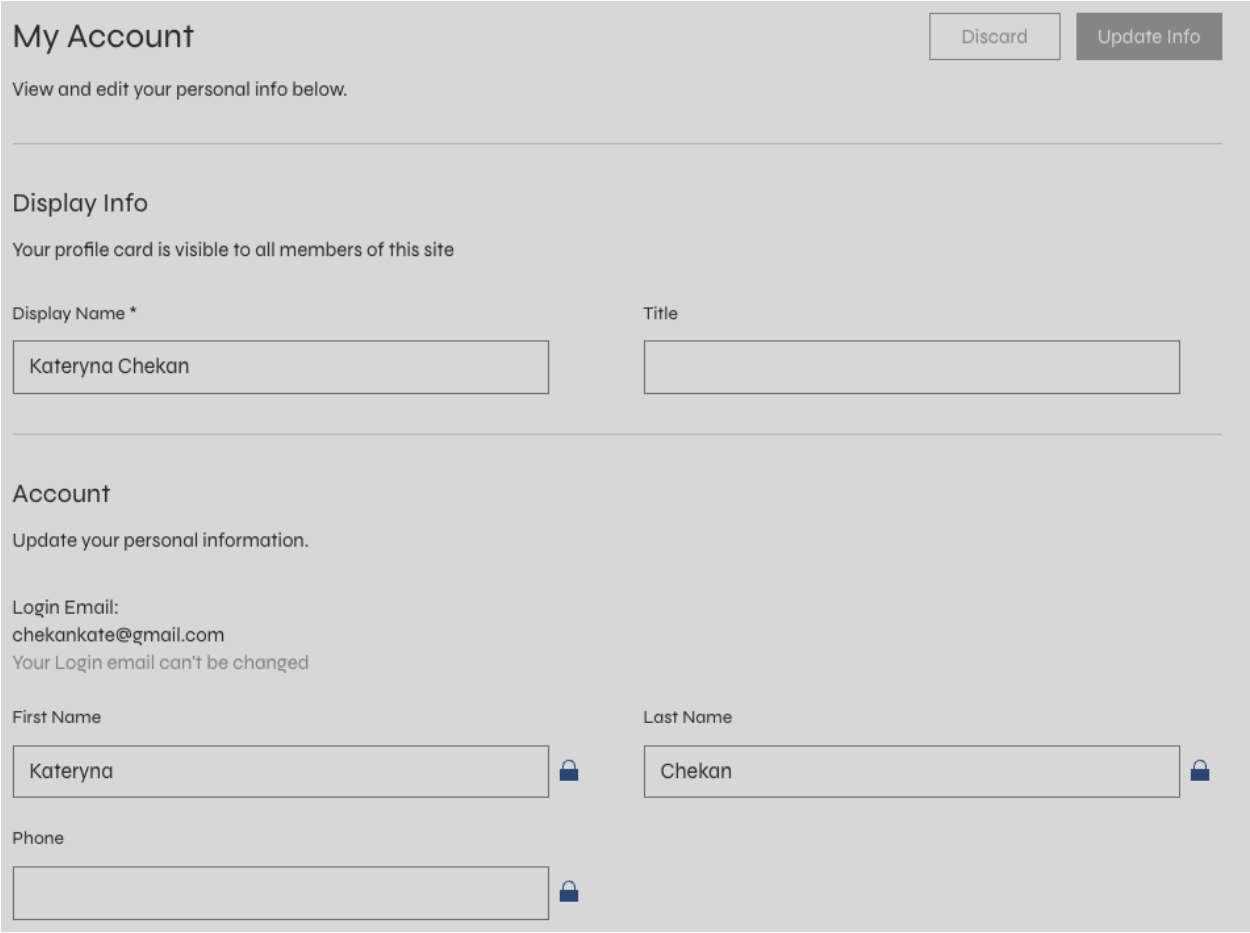

Рис. 3.11. Сторінка для заповнення даних користувача

3.4. Безпека додатку з використанням Spring Security

Включення системи безпеки (Security) в Java-додаток для управління часом і фінансами є критично важливим для забезпечення захисту даних та гарантування конфіденційності та цілісності інформації. Саме для цього було використано REST Security With JWT, Spring Security.

JWT (абревіатура від JSON Web Token) – це стандарт для використання токенів для автентифікації в Інтернеті загалом, а не лише в контексті REST-служб. На даний момент він знаходиться в статусі чернетки, описаному у RFC 7519. JWT є надійним та може містити різноманітну інформацію, при цьому залишаючись простим у використанні, навіть при відносно невеликому обсязі даних. Як і будь-який інший токен, JWT можна

використовувати для передачі ідентифікаційних даних автентифікованих користувачів між постачальником ідентифікаційної інформації та постачальником послуг (що можуть бути різними системами). Він також може включати всі необхідні відомості про користувача, такі як дані авторизації, тим самим звільняючи постачальника послуг від необхідності обходити базу даних чи зовнішні системи для перевірки ролей та дозволів для кожного запиту; всі ці дані можуть бути витягнуті безпосередньо з токена.

Отже, ось як працює механізм захисту JWT:

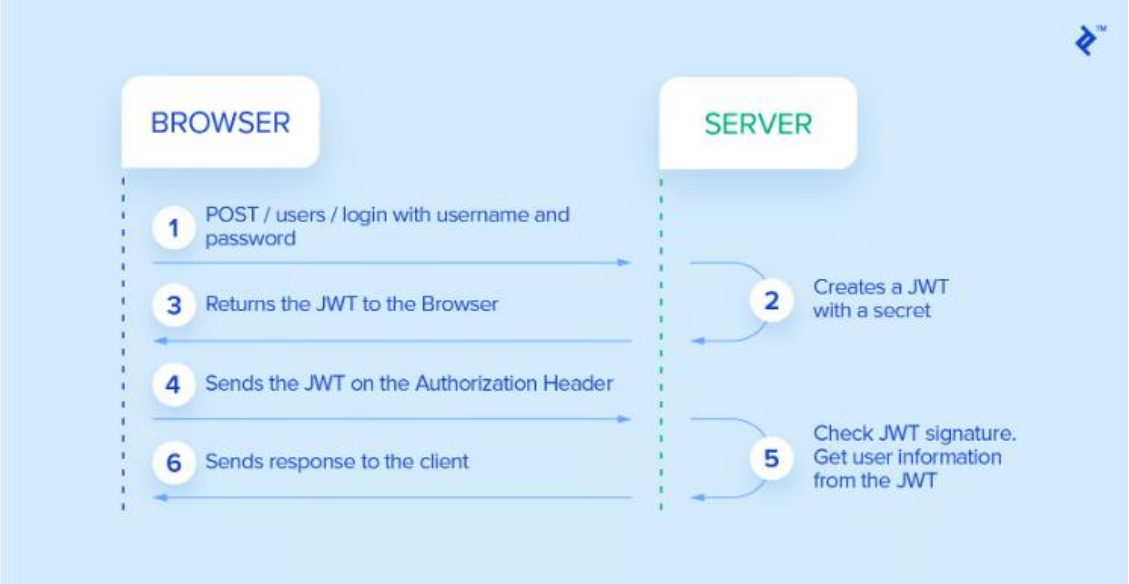

Рис. 3.12. Принцип роботи security з JWT токеном

- Клієнти отримують доступ до системи, передаючи свої облікові дані постачальнику ідентифікаційної інформації.
- Постачальник ідентифікаційної інформації перевіряє ці облікові дані; якщо все в порядку, він отримує дані користувача, створює JWT, який включає дані користувача та дозволи для доступу до послуг, а також встановлює термін дії JWT (який може бути обмеженим або необмеженим).

- Постачальник ідентифікаційної інформації підписує та, за необхідності, шифрує JWT, надсилаючи його клієнту як відповідь на початковий запит із обліковими даними.
- Клієнт зберігає JWT протягом обмеженого чи необмеженого періоду часу, залежно від терміну дії, встановленого постачальником ідентифікаційної інформації.
- Клієнт передає збережений JWT у заголовку авторизації для кожного запиту до постачальника послуг.
- Для кожного запиту постачальник послуг використовує JWT із заголовка авторизації, розшифровує його та, за необхідності, перевіряє підпис. Якщо всі перевірки пройдені успішно, він витягує дані користувача та дозволи, використовуючи лише ці дані для вирішення прийняття чи відхилення запиту клієнта. Зауважте, що для успішної роботи необхідно, щоб постачальники ідентифікаційної інформації та послуг мали взаєморозуміння щодо процесів шифрування, щоб служба могла перевірити підпис чи навіть розшифрувати зашифровані дані ідентифікації. [29]

Цей метод забезпечує велику гнучкість, одночасно зберігаючи дані в безпеці та полегшуючи розробку. Використовуючи цей підхід, можна з легкістю додавати нові серверні вузли до кластера постачальника послуг, ініціалізуючи їх здатністю лише перевіряти підпис та розшифровувати маркери за умови, що вони мають загальний секретний ключ. Реплікація сеансу, синхронізація бази даних або взаємодія між вузлами стає зайвою. Відпочивайте, насолоджуйтесь простотою та ефективністю.

Основним відмінником JWT від інших токенів є стандартизований формат вмісту токена. Ще однією рекомендованою практикою є включення JWT в заголовок авторизації через схему носія.

Для ефективної роботи служб REST потрібно використовувати інший підхід до авторизації порівняно з класичними багатосторінковими веб-сайтами.

Замість того, щоб починати процес автентифікації через переадресацію на сторінку входу при запиті захищеного ресурсу, сервер REST автентифікує кожен запит, використовуючи дані, включені безпосередньо в запит, такі як маркер JWT. У випадку невдалої автентифікації сервер просто відправляє відповідь з HTTP-кодом 401 (неавторизовано), і клієнти повинні мати відповідний обробник для цього; наприклад, браузер може відобразити динамічний елемент для введення ім'я користувача та паролю.

Як і очікувалося, у Spring Security Framework є багато готових до використання класів для "звичайних" механізмів авторизації, таких як файли cookie сесії, HTTP Basic і HTTP Digest. Проте він не має вбудованої підтримки для JWT, тому нам доведеться внести певні зміни, щоб забезпечити його функціональність. Для більш докладної інформації рекомендується переглянути офіційну документацію Spring Security.

Початково слід визначити фільтр безпеки Spring у файлі web.xml. Назва фільтра безпеки Spring повинна бути обов'язковою springSecurityFilterChain, щоб інші конфігурації Spring працювали належним чином.

Далі слід внести XML-декларацію компонентів Spring, що відносяться до безпеки. Для спрощення XML ми встановимо простір імен за замовчуванням як security, додаючи xmlns="http://www.springframework.org/schema/security" до кореневого елемента XML. Конфігурація XML має наступний вигляд:

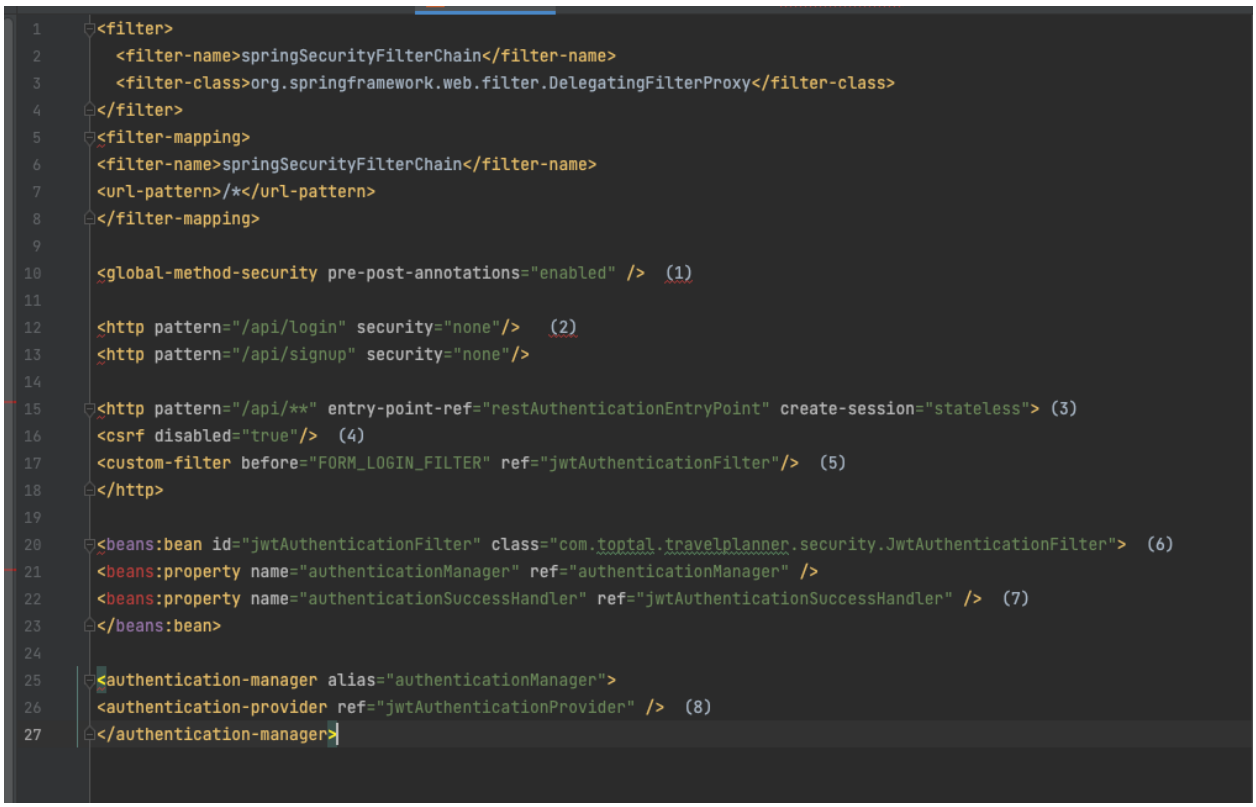

Рис. 3.13. Конфігурація security в xml файлі

1) У цьому рядку ми вмикаємо анотації @PreFilter, @PreAuthorize, @PostFilter, @PostAuthorize для будь-яких компонентів Spring у контексті.

2) Ми визначаємо кінцеві точки входу та реєстрації, щоб пропустити безпеку; навіть "анонімний" повинен мати можливість виконувати ці дві операції.

3) Далі ми встановлюємо ланцюжок фільтрів, застосований до всіх запитів, додаючи дві важливі конфігурації: посилання на точку входу та налаштування створення сеансу без стану (ми не хочемо, щоб сеанс створювався з метою безпеки, оскільки ми використовуємо маркери для кожного запиту).

4) Нам не потрібен захист від CSRF, оскільки наші токени захищені від нього.

5) Далі ми приєднуємо наш спеціальний фільтр автентифікації до попередньо визначеного ланцюжка фільтрів Spring, безпосередньо перед фільтром входу у форму.

6) Цей компонент є оголошенням нашого фільтра автентифікації; оскільки він розширює AbstractAuthenticationProcessingFilter Spring, нам потрібно оголосити його в XML, щоб зв'язати його властивості (автоматичне з'єднання тут не працює). Пізніше ми пояснимо, що робить фільтр.

7) Обробник успіху за замовчуванням AbstractAuthenticationProcessingFilter недостатньо хороший для цілей REST, оскільки він перенаправляє користувача на сторінку успішного виконання; тому ми створили свій власний тут.

8) Оголошення постачальника, створене authenticationManager, використовується нашим фільтром для автентифікації користувачів. [30]

Далі потрібно створити деякі java-класи для повного функціонування Spring Security з JWT tokens.

RestAuthenticationEntryPoint.java – забезпечує обробку автентифікації JWT, повертаючи HTTP-код 401 (Unauthorized), якщо автентифікація не вдається, і замінюючи стандартне перенаправлення Spring. Він є важливою точкою входу для автентифікації JWT, витягуючи маркер з заголовків запиту та делегуючи автентифікацію AuthenticationManager. Якщо маркер відсутній, генерується виняток, зупиняючи обробку запиту. Клас також має перевизначення для успішної автентифікації, уникаючи стандартного перенаправлення Spring.

JwtAuthenticationProvider.java – у класі використовується Spring AuthenticationManager за замовчуванням, але додається власний AuthenticationProvider для виконання фактичної автентифікації. AbstractUserDetailsAuthenticationProvider використовується для цього, вимагаючи повернення UserDetails на основі маркера JWT (у формі JwtAuthenticationToken). Якщо маркер недійсний, створюється виняток. У разі успішної автентифікації витягуються дані користувача з токену,

інформація про якого містить всі необхідні дані, включаючи ролі, без звернення до бази даних.

Нарешті, клас *JwtUtil.java* відповідає за розбір та генерацію маркерів на основі об'єкта користувача. Для цього використовується бібліотека jjwt. Метод generateToken() використовується для створення маркера, який повертається клієнтам на основі інформації про користувача.

ВИСНОВОК ДО РОЗДІЛУ 3

В результаті розділу, присвяченого розробці Java додатка для моніторингу часу та фінансів, було виявлено важливі етапи і компоненти, що додають значну функціональність та ефективність програмі. Зокрема, використано бібліотеку React Big Calendar для зручного та інтерактивного відображення подій у календарі, що робить користування додатком більш ефективним та зрозумілим.

Додаток також отримав можливість здійснення фінансових транзакцій через інтеграцію з Stripe, що робить його більш функціональним та готовим для реального використання. Інтеграція Docker container для MongoDB дозволяє забезпечити гнучкість та легкість управління базою даних, а його налаштування відповідно до потреб додатка додає стабільності та ефективності.

Одним із ключових аспектів розглянутого розділу є забезпечення безпеки. Використання Spring Security та JWT tokens забезпечує надійний рівень захисту для додатку, управління доступом та збереження конфіденційності користувачів та їх даних.

Загальний висновок полягає в тому, що використання розглянутих технологій та інструментів дозволило створити потужний та функціональний Java додаток, який забезпечує ефективний моніторинг часу та фінансів. Реалізація кожного етапу вибудовує структуру програми та додає цінність для кінцевого користувача, роблячи додаток зручним та безпечним у використанні.

ВИСНОВКИ

У ході дослідження предметної області було ретельно розглянуто необхідність ефективного керування особистим часом та фінансами для кожної людини. Підкреслено важливість володіння навичками Time Management та фінансового планування як ключового елементу успішного життя. У роботі наведено загальну інформацію щодо концепцій Time Management та керування фінансами, а також розглянуто різні методи та поради, які сприяють покращенню навичок управління власними ресурсами.

Для розробки додатку потрібно було ретельно обрати технології. Враховуючи технологічний аспект, для реалізації серверної частини обрано мову програмування Java та бібліотеку Spring, що забезпечує надійність та широкі можливості для створення веб-додатку. Використання Open Banking API дозволить взаємодіяти з банківськими даними, а REST API та база даних MongoDB забезпечать ефективну обробку і зберігання інформації.

Для клієнтської частини додатку вибрано мову програмування JavaScript та бібліотеку React, що гарантує створення зручного та ефективного інтерфейсу взаємодії з серверною частиною. Інтеграція з Stripe та використання Docker container для MongoDB додають функціональність та легкість управління фінансовими транзакціями та базою даних відповідно.

Спеціальна увага приділяється питанням безпеки, де використання Spring Security та JWT tokens гарантує надійний рівень захисту для користувачів та їхніх даних.

Загальний висновок підкреслює, що використання обраних технологій та інструментів дозволило створити потужний та функціональний додаток, який відзначається ефективним моніторингом часу та фінансів. Кожен етап розробки спрямований на покращення користувацького досвіду, забезпечуючи зручність та безпеку використання.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. The Importance of Time Management: Tips for Productivity [Електронний ресурс] – Режим доступу до ресурсу: https://www.freshbooks.com/hub/productivity/importance-of-time-management.
2. The Importance Of Keeping Track Of Expenses [Електронний ресурс] – Режим доступу до ресурсу: https://oboloo.com/blog/the-importance-of-keeping-track-of-expenses/.
3. Best Time Management Tools 2023 - What tools will really boost your productivity? [Електронний ресурс] – Режим доступу до ресурсу: https://www.spica.com/blog/best-time-management-tools.
4. Getting started with Google Calendar [Електронний ресурс] – Режим доступу до ресурсу: https://edu.gcfglobal.org/en/google-tips/getting-started-with-google-calendar/1/#.
5. The best expense tracker apps of 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://www.cnbc.com/select/best-expense-tracker-apps/.
6. Морган Хаузел, «The Psychology of Money: Timeless Lessons on Wealth, Greed, and Happiness», 2020
7. Браян Трейсі, «Time Management», Канада, 2006.
8. What is Java? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/topics/java.
9. Spring Framework [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/searchapparchitecture/definition/Spring-Framework.
10. Introduction to Spring Framework [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/docs/3.0.0.M4/reference/html/ch01s02.html.

11. What Is a REST API? Examples, Uses, and Challenges [Електронний ресурс] – Режим доступу до ресурсу: https://blog.postman.com/rest-api-examples/.

12. What is MongoDB – Working and Features [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/what-is-mongodb-working-and-features/.

13. What Is Open Banking? [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwareag.com/en_corporate/resources/api/article/open-banking.html.

14. Introduction to JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/introduction-to-javascript/.

15. What Is React.js? A Look at the Popular JavaScript Library [Електронний ресурс] – Режим доступу до ресурсу: https://kinsta.com/knowledgebase/what-is-react-js/.

16. Intellij Idea - Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/intellij_idea/intellij_idea_introduction.htm.

17. Why Visual Studio Code? [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/docs/editor/whyvscode.

18. Крейг Уоллс, «Spring in action», Англія, 2015.

19. Ерік Фрімен і Елізабет Робсон, «Вивчаємо програмування на JavaScript», США, 2014.

20. Фаулер Мартін, «NoSQL. Методологія розробки нереляційних баз даних», Англія, 2020.

21. Event Calendar using React [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/event-calendar-using-react/.

22. React Big Calendar [Електронний ресурс] – Режим доступу до ресурсу: https://www.scaler.com/topics/react/react-big-calendar/.

23. Adding Payments to your Web Application: An Introduction to Stripe Integration [Електронний ресурс] – Режим доступу до ресурсу:

https://medium.com/sipios/adding-payments-to-your-web-application-an-introduction-to-stripe-integration-3abd7354206a.

24. Implementing a Payment Gateway Integration in Java Full Stack Applications [Електронний ресурс] – Режим доступу до ресурсу: https://seldomindia.com/implementing-a-payment-gateway-integration-in-java-full-stack-applications/.

25. Payment Gateway Integration in Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/payment-gateway-integration-in-java.

26. Introduction to the Stripe API for Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/java-stripe-api.

27. Spring Boot MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/spring-boot-mongodb.

28. Docker and MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/compatibility/docker.

29. REST Security With JWT Using Java and Spring Security [Електронний ресурс] – Режим доступу до ресурсу: https://www.toptal.com/java/rest-security-with-jwt-spring-security-and-java.

30. OAuth 2.0 Resource Server JWT [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-security/reference/servlet/oauth2/resource-server/jwt.html.