

Міністерство освіти і науки України
Одеський національний політехнічний університет
Інститут комп'ютерних систем
Кафедра інформаційних систем

КУРСОВА РОБОТА

з дисципліни: «Технології створення програмних продуктів»
за темою: "Розробка органайзеру «Catch Everything»"

Частина 3

Виконав:
студент 3-го курсу
групи АА-181
Марков Д.Т.
Перевірив:
Блажко О. А.

Одеса-2020

Анотація

В курсовій роботі розглядається процес створення програмного продукту «Catch Everything». Робота виконувалась в команді з декількох учасників:

1. Волков С.В.;
2. Вознюк Д.В.;
3. Марков Д.

Тому в пояснювальній записці у розділах «Проектування» та «Конструювання» детальніше описано лише одну частину з урахуванням планів проведених робіт з розділу «Планування» з описом особливостей конструювання:

- структур даних реляційної моделі в системі керування базами даних pgAdmin4\$

- програмних модулів в інструментальному середовищі MVC з використанням фреймворку Spring boot та мови програмування Java.

Результати роботи розміщено на github-репозиторії за адресою:

<https://github.com/DavidMarkov/TCSP>

Перелік скорочень

ОС – операційна система

ІС – інформаційна система

БД – база даних

СКБД – система керування базами даних

ПЗ – програмне забезпечення

ПП – програмний продукт

UML – уніфікована мова моделювання

Зміст

1 Вимоги до програмного продукту	6
1.1 Визначення потреб споживача	6
1.1.1 Ієрархія потреб споживача	6
1.1.2 Деталізація матеріальної потреби	7
1.2 Бізнес-вимоги до програмного продукту	7
1.2.1 Опис проблеми споживача	7
1.2.1.1 Концептуальний опис проблеми споживача	7
1.2.1.2 Метричний опис проблеми споживача	7
1.2.2 Мета створення програмного продукту	8
1.2.2.1 Проблемний аналіз існуючих програмних продуктів	8
1.2.2.2 Мета створення програмного продукту	8
1.3 Вимоги користувача до програмного продукту	8
1.3.1 Історія користувача програмного продукту	8
1.3.2 Діаграма прецедентів програмного продукту	9
1.3.3 Сценаріїв використання прецедентів програмного продукту	9
1.4 Функціональні вимоги до програмного продукту	14
1.4.1 Багаторівнева класифікація функціональних вимог	14
1.4.2 Функціональний аналіз існуючих програмних продуктів	16
1.5 Нефункціональні вимоги до програмного продукту	17
1.5.1 Опис зовнішніх інтерфейсів	17
1.5.1.1 Опис інтерфейса користувача	17
1.5.1.1.1 Опис INPUT-інтерфейса користувача	17
1.5.1.1.2 Опис OUTPUT-інтерфейса користувача	18
1.5.1.2 Опис інтерфейсу із зовнішніми пристроями	21
1.5.1.3 Опис програмних інтерфейсів	21
1.5.1.4 Опис інтерфейсів передачі інформації	21
1.5.1.5 Опис атрибутів продуктивності	21
2 Планування процесу розробки програмного продукту	22
2.1 Планування ітерацій розробки програмного продукту	22
2.2 Концептуальний опис архітектури програмного продукту	22
2.3 План розробки програмного продукту	23
2.3.1 Оцінка трудомісткості розробки програмного продукту	23
2.3.2 Визначення дерева робіт з розробки програмного продукту	25
2.3.3 Графік робіт з розробки програмного продукту	26
2.3.3.1 Таблиця з графіком робіт	26

2.3.3.2 Діаграма Ганта.....	27
3 Проектування програмного продукту.....	30
3.1 Концептуальне та логічне проектування структур даних програмного продукту	30
3.1.1 Концептуальне проектування на основі UML-діаграми концептуальних класів.....	30
3.1.2 Логічне проектування структур даних	30
3.2 Проектування програмних класів	31
3.3 Проектування алгоритмів роботи методів програмних класів.....	31
3.4 Проектування тестових наборів методів програмних класів	31
4 Конструювання програмного продукту.....	47
4.1. Особливості інсталяції та роботи з СУБД	47
4.2. Створення структур Даних	49
4.2.1.Створення бази даних.....	49
4.3. Робота з IDEA	53
4.3.1Spring MVC:	53
4.3.2. Створення програмних класів.....	56
4.4. Особливості розробки алгоритмів методів програмних класів	59
4.5. Використання спеціалізованих програмних бібліотек	61
4.6. Модульне тестування програмних класів	63
5 Розгортання та валідація програмного продукту	69
5.1 Інструкція з встановлення програмного продукту	70
5.2 Інструкція з використання програмного продукту.....	70
5.3 Результати валідації програмного продукту	72
Висновки	73

1 Вимоги до програмного продукту

1.1 Визначення потреб споживача

1.1.1 Ієрархія потреб споживача

Відомо, що в теорії маркетингу потреби людини можуть бути представлені у вигляді ієрархії потреб ідей американського психолога Абрахама Маслоу включають рівні:

- фізіологія (вода, їжа, житло, сон);
- безпека (особиста, здоров'я, стабільність),
- приналежність (спілкування, дружба, любов),
- визнання (повага оточуючих, самооцінка),
- самовираження (вдосконалення, персональний розвиток).

На рисунку 1.1 представлена ієрархія потреби споживача. Хотілося б задовольнити фізіологічні потреби, використовуючи майбутній програмний продукт.

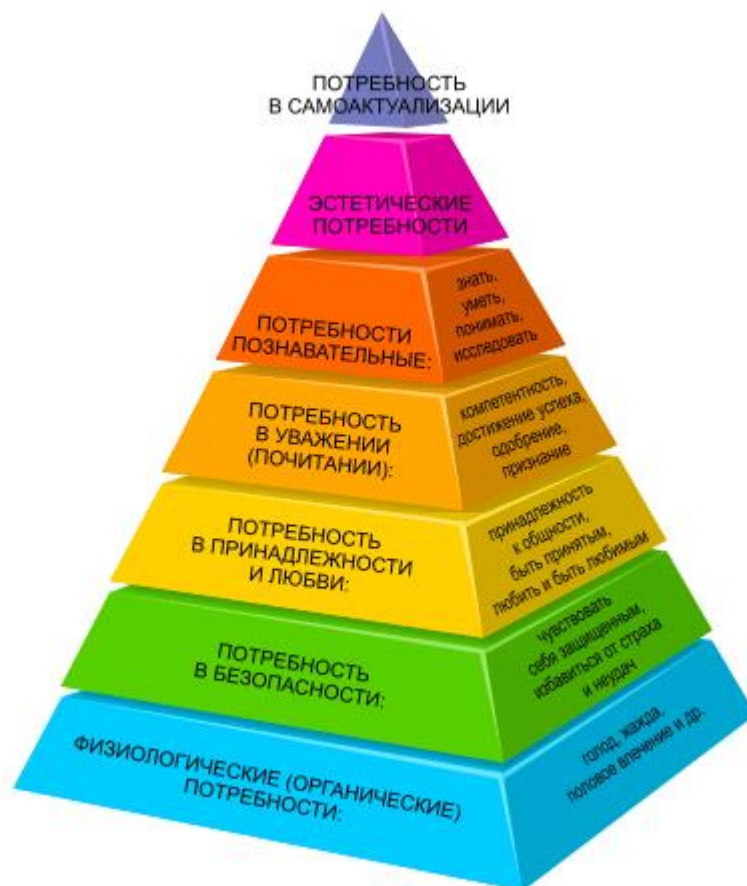


Рис. 1.1 – Приклад ієрархії потреби споживача

1.1.2 Деталізація матеріальної потреби

Для деталізації матеріальних потреб була розроблена MindMap, що представлена на рисунку 1.2.

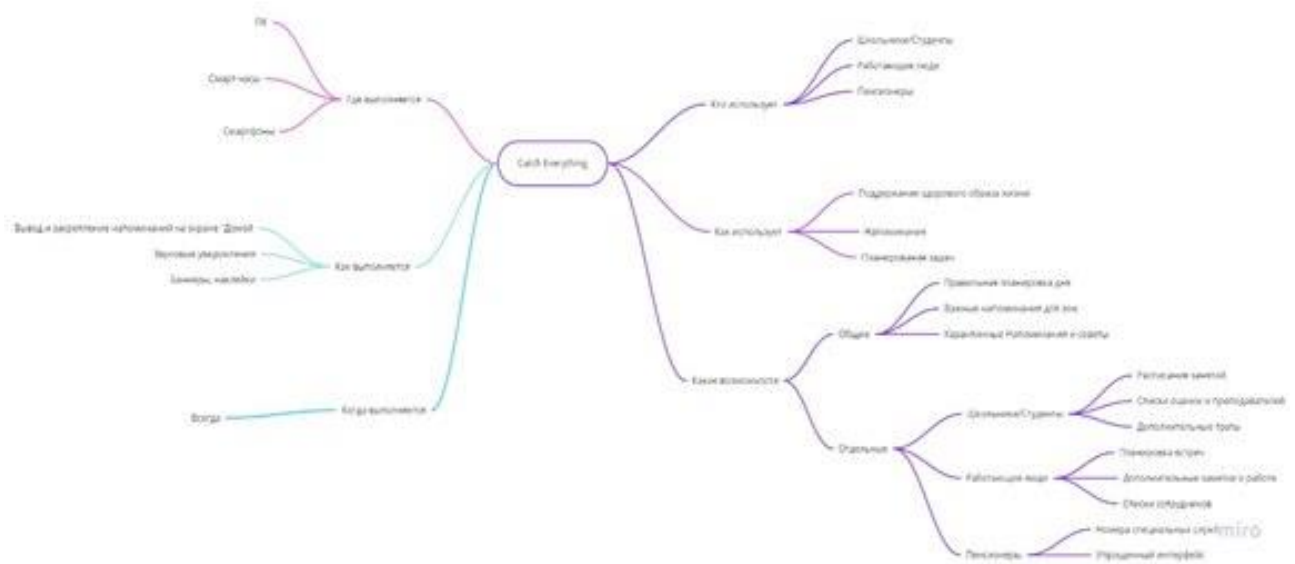


Рис. 1.2 – MindMap

1.2 Бізнес-вимоги до програмного продукту

1.2.1 Опис проблеми споживача

1.2.1.1 Концептуальний опис проблеми споживача

Неможливо запам'ятати:

1. Всі майбутні заплановані події;
2. З точністю стежити за водним балансом;
3. Адреса та номери телефонів та емейлів;
4. Розклад занять, рецепти блюд та т.д.

1.2.1.2 Метричний опис проблеми споживача

Метричний показник: Низький рівень продуктивності через неможливість запам'ятовування всієї інформації.

Рівень продуктивності LP (LP – the Level of Productivity) можна визначити як:

$$LP = N_x / N, \text{ де}$$

N_x – кількість потрібної інформації, яку користувач пам'ятає;

N – кількість всієї потрібної інформації.

1.2.2 Мета створення програмного продукту

1.2.2.1 Проблемний аналіз існуючих програмних продуктів

Таблиця 1.2.2.1 – Аналіз існуючих ПП

№	Назва продукту	Вартість	Ступінь готовності	Примітка
1	Органайзер Calendars	Безкоштовно/ платно	1	Просунуте додаток, підійде лише людям працюючим у великих компаніях
2	Щоденник Tappsk	Безкоштовно/ платно	1	Цікава система звичок, зручний інтерфейс
3	Список справ і завдань Todoist	Безкоштовно/ платно	1	Можливість розставляти пріоритети, захищений початковий екран

1.2.2.2 Мета створення програмного продукту

Метою роботи є створення органайзера з зручним інтуїтивно-зрозумілим користувацьким інтерфейсом. Підвищення продуктивності шляхом запису всієї необхідної інформації, яка при потребі буде виводиться у вигляді нагадувань.

1.3 Вимоги користувача до програмного продукту

1.3.1 Історія користувача програмного продукту

1. Як користувач, я можу управляти завданнями (створення, редагування, видалення).
2. Як користувач, я можу управляти подією (створення, редагування, видалення).
3. Як користувач, я можу управляти телефонною книгою (створення, редагування, видалення).
4. Як користувач, я можу управляти нотатками (створення, редагування, видалення).
5. Як користувач, я можу отримувати нагадування про карантинні заходи.
6. Як користувач, я можу управляти особистими записами (особистий щоденник).
7. Як користувач, я можу використовувати менеджер паролів, для збереження паролів.
8. Як користувач, я можу купити підписку на преміум, для відключення реклами.
9. Як гость, я можу зареєструватися у додатку.

1.3.2 Діаграма прецедентів програмного продукту

Діаграма прецедентів зображена на рисунку 1.3.2.

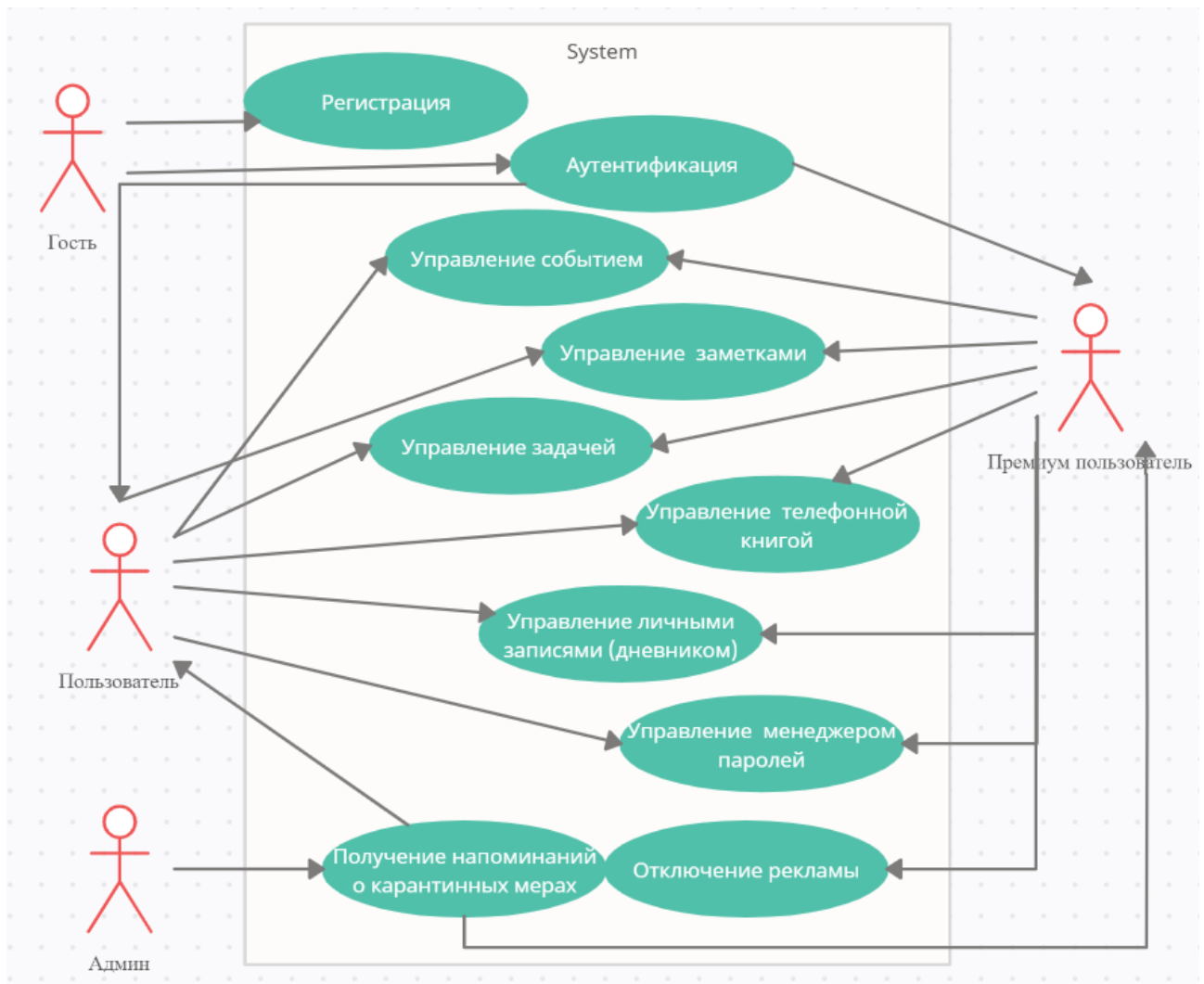


Рис. 1.3.2 – Діаграма прецедентів

1.3.3 Сценарії використання прецедентів програмного продукту

Сценарій №1.

Назва прецеденту: «Управління подією».

Передумова початку виконання сценарію: Потреба у створенні, редагуванні і видаленні події.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання нового події і можливість його редагування.

Приклад основного успішного сценарію прецеденту «Управління подією»:

1. Користувач передає ПП текстове повідомлення з точним зазначенням часу його настання для додавання події.

2. ПП зберігає подія і дає можливість редагувати або видалити його.
Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління подією»:

3.1 ПП виявляє, що користувач не вказав час настання події.

3.a.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №2.

Назва прецеденту: «Управління завданням».

Передумова початку виконання сценарію: Потреба у створенні, редагуванні і видаленні завдання.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання нового завдання і можливість її редагування.

Приклад основного успішного сценарію прецеденту «Управління завданням»:

2. Користувач передає ПП опис справ, які йому необхідно виконати для додавання завдання.

2. ПП зберігає завдання і дає можливість редагувати або видалити її.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління завданням»:

3.1 ПП виявляє, що користувач не вказав жодної справи.

3.a.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №3.

Назва прецеденту: «Отримання нагадування про карантинні заходи».

Передумова початку виконання сценарію: Запобігання поширенню вірусної інфекції.

Актор: ПП.

Гарант успіху: Оповіщення користувача.

Приклад основного успішного сценарію прецеденту «Отримання нагадування про карантинні заходи»:

1. ПП відправляє користувачеві текстове повідомлення про необхідність дотримання карантинних заходів з певним інтервалом часу.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Отримання нагадування про карантинні заходи»:

1.1. Користувач не отримує повідомлення про дотримання карантинних заходів через те, що відключив раніше цю функцію.

Сценарій №4.

Назва прецеденту: «Управління замітками».

Передумова початку виконання сценарію: Потреба у створенні, редагуванні і видаленні замітки.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання нової замітки і можливість її редагування.

-Приклад основного успішного сценарію прецеденту «Управління замітками»:

1. Користувач передає ПП текстове повідомлення для додавання замітки.
2. ПП зберігає замітку і дає можливість редагувати або видалити її.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління замітками»:

3.1 ПП виявляє, що користувач нічого не заповнив.

3.а.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №5.

Назва прецеденту: «Управління телефонною книгою».

Передумова початку виконання сценарію: Потреба в запису, редагування і видалення номера телефону.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання нового запису в телефонній книзі і можливість її редагування.

Приклад основного успішного сценарію прецеденту «Управління телефонною книгою»:

1. Користувач передає ПП контактні дані для додавання номера телефону.
2. ПП зберігає запис з номером телефону і дає можливість редагувати або видалити його.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління телефонною книгою»:

3.1 ПП виявляє, що користувач передав ПП не всі необхідні контактні дані.

3.а.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №6.

Назва прецеденту: «Управління особистими записами (щоденником)».

Передумова початку виконання сценарію: Потреба у веденні, редагуванні і видаленні особистих записів.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання новий особистий записи і можливість її редагування.

Приклад основного успішного сценарію прецеденту «Управління особистими записами (щоденником)»:

1. Користувач передає ПП текстове повідомлення із зазначенням часу його додавання для додавання особистої записи.

2. ПП зберігає особисту запис і дає можливість редагувати або видалити її.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління особистими записами (щоденником)»:

3.1 ПП виявляє, що користувач не вказав час.

3.а.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №7.

Назва прецеденту: «Управління менеджером паролів».

Передумова початку виконання сценарію: Потреба в запису, редагування і видалення паролів.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання нового пароля і можливість його редагування.

Приклад основного успішного сценарію прецеденту «Управління менеджером паролів»:

1. Користувач передає ПП умовне слово для додавання пароля.

2. ПП зберігає пароль і дає можливість редагувати або видалити його.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління менеджером паролів»:

3.1 ПП виявляє, що користувач нічого не заповнив.

3.а.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №8.

Назва прецеденту: «Відключення реклами».

Передумова початку виконання сценарію: Потреба в запису, редагування і видалення паролів.

Актор: Користувач.

Гарант успіху: Використання організатора без реклами.

Приклад основного успішного сценарію прецеденту «Відключення реклами»:

1. Користувач передає дані про свою карту і оплачує замовлення на покупку преміум.

2. ПП обробляє дані.

3. ПП змінює роль користувача на «Преміум-користувач» і відправляє квитанцію про оплату на пошту користувача.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Відключення реклами»:

3.1 ПП виявляє, що користувач передав ПП не всі дані карти.

3.a.1. ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

3.2. ПП виявляє, що у користувача недостатня сума на рахунку.

3.a.2. ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №9.

Назва прецеденту: «Аутентифікація».

Передумова початку виконання сценарію: Потреба у вході в свою особисту сторінку.

Актор: Гість.

Гарант успіху: Вхід на особисту сторінку.

Приклад основного успішного сценарію прецеденту «Аутентифікація»:

1. Користувач вводить логін і пароль.

2. ПП обробляє дані.

3. ПП вітає користувача про успішне вході на особисту сторінку

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Аутентифікація»:

3.1 ПП виявляє, що користувач передав ПП не коректний логін або пароль, або логін і пароль.

3.a.1. ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №10.

Назва прецеденту: «Реєстрація».

Передумови початку виконання сценарію: Потреба у створенні особистої сторінки.

Актор: Гість.

Гарантії успіху: Реєстрація користувача.

Приклад основного успішного сценарію прецеденту «Реєстрація»:

1. Користувач вводить необхідні дані для реєстрації.

2. ПП обробляє дані.

3. ПП вітає користувача про успішну реєстрацію.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Реєстрація»:

3.1 ПП виявляє, що користувач передав ПП не коректні дані.

3.а.1. ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

1.4 Функціональні вимоги до програмного продукту

1.4.1. Багаторівнева класифікація функціональних вимог

Багаторівнева класифікація функціональних вимог представлена в таблиці 1.4.1. На рисунку 1.4.1 опис ієрархічної класифікація функціональних вимог.

Таблиця 1.4.1. Багаторівнева класифікація функціональних вимог

Ідентифікатор функції	Назва функції
FR1	Управління подіями(д)/ завданнями(с)/ нотатками(Д) / паролями(д) / номерами(С)
FR1.1	Користувач переходить у розділ Події/Менеджер паролів і вибирає дію "Додати/змінити подію/пароль"
FR1.2	Користувач надсилає/редагує необхідні дані ПП для додавання/змінення елемента
FR1.3	ПП зберігає внесені дані
FR2	Управління Завданнями/Номерами
FR2.1	Користувач переходить у розділ Завданнями/Телефона книга і вибирає дію "Додати/змінити завдання/запис в телефонії книзі"
FR2.2	Користувач надсилає/редагує необхідні дані ПП для додавання/змінення елемента
FR2.3	ПП зберігає внесені дані
FR3	Управління Нотатками/Особистим щоденником
FR3.1	Користувач переходить у розділ Нотатки/Особистий щоденник і вибирає дію "Додати/змінити Нотаток/Запис в особистому щоденнику"
FR3.2	Користувач надсилає/редагує необхідні дані ПП для додавання/змінення елемента

Продовження таблиці 1.4.1. Багаторівнева класифікація функціональних вимог

FR3.3	ПП зберігає внесені дані
FR4	Реєстрація
FR4.1	Користувач переходить у розділ Реєстрація
FR4.2	Користувач надсилає необхідні дані ПП для реєстрації
FR4.3	ПП зберігає дані користувача
FR5	Аутентифікація
FR5.1	Користувач переходить у вікно «Вхід»
FR5.2	Користувач надсилає логін та пароль ПП для входу
FR5.3	ПП перевіряє логін та пароль на відповідність в БД
FR5.4	Користувач заходить на особисту сторінку
FR6	Отримання нагадування про карантинні заходи
FR6.1	ПП надсилає користувачеві текстове повідомлення про те, що карантинних заходів потрібно дотримуватися через певний проміжок часу
FR7	Вимкнення реклами
FR7.1	Користувач заходить в розділ «Придбати Преміум» і вибирає акцію «придбати преміум».
FR7.2	Користувач передає необхідні дані для придбання преміум
FR7.3	ПП обробляє дані
FR7.4	ПП змінює роль користувача на "Преміум користувач" і відправляє квитанцію про оплату поштою користувача.

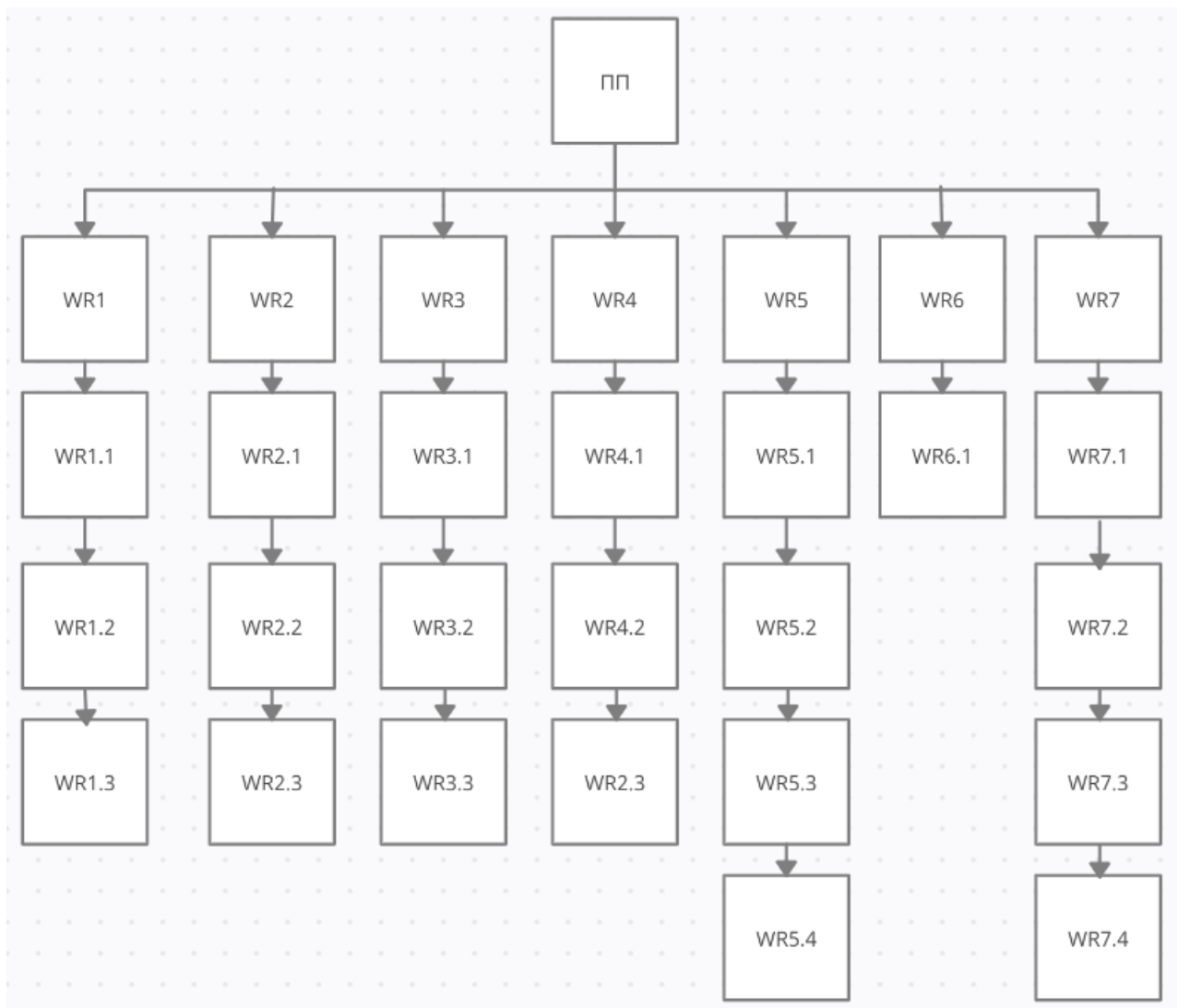


Рис. 1.4.1 – Опис ієрархічної класифікація функціональних вимог

1.4.2 Функціональний аналіз існуючих програмних продуктів

Таблиця 1.4.2 – Функціональний аналіз існуючих програмних продуктів

Ідентифікатор функції	onlinemschool.com	yaklass.ru	geogebra.org
FR1	≈	≈	≈
FR2	-	-	-
FR3	+	+	+
FR4	+	-	-
FR5	+	+	+
FR6	-	-	-
FR7	+	-	+

1.5 Нефункціональні вимоги до програмного продукту

1.5.1 Опис зовнішніх інтерфейсів

1.5.1.1 Опис інтерфейса користувача

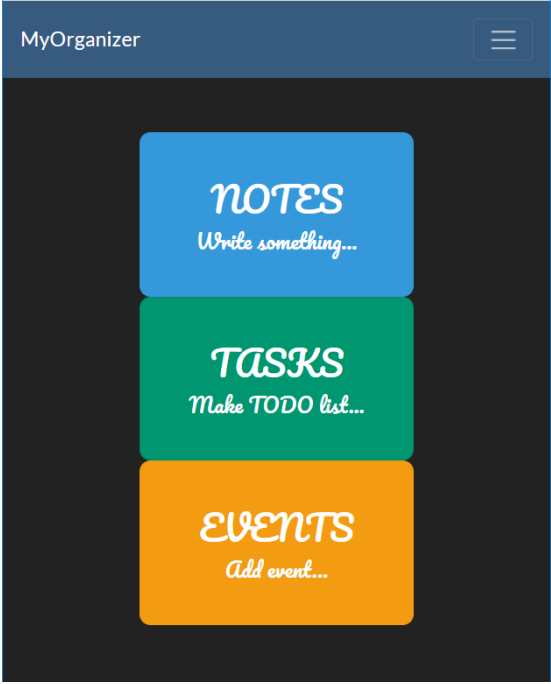
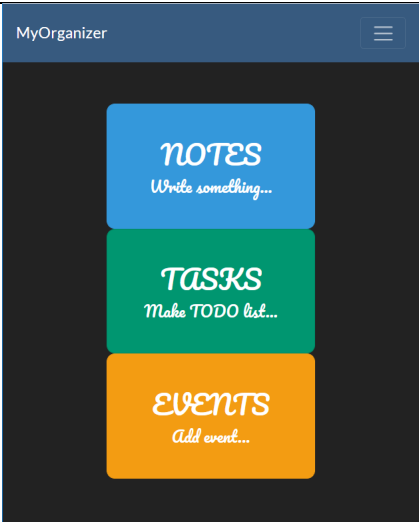
1.5.1.1.1 Опис INPUT-інтерфейса користувача

Таблиця 1.5.1.1.1 – Опис INPUT-інтерфейса користувача

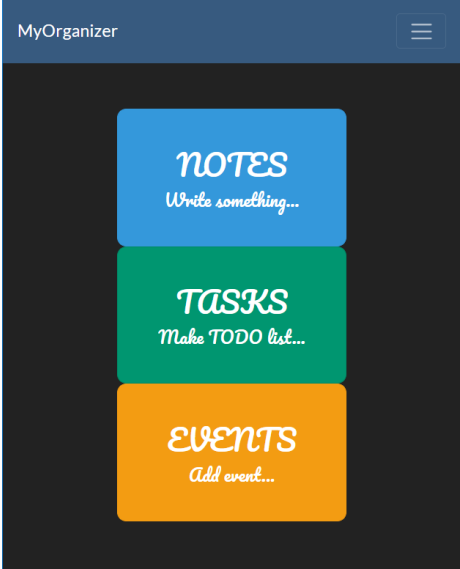

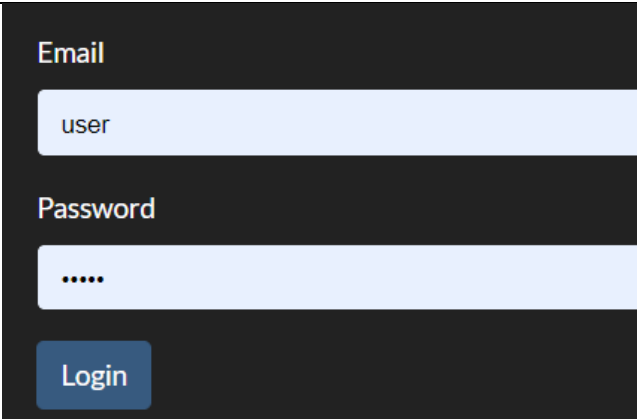
Ідентифікатор функції	Засіб INPUT потоку	Особливості використання
FR1	Стандартна клавіатура і миша, сенсорний екран"	
FR2	Стандартна клавіатура і миша, сенсорний екран	
FR3	Стандартна клавіатура і миша, сенсорний екран	
FR4	Стандартна клавіатура і миша, сенсорний екран	
FR5	Стандартна клавіатура і миша, сенсорний екран	
FR7	Стандартна клавіатура і миша, сенсорний екран	

1.5.1.1.2 Опис OUTPUT-інтерфейса користувача

Таблиця 1.5.1.1.2 – Опис OUTPUT-інтерфейса користувача

Ідентифікатор функції	Засіб OUTPUT потоку	Особливості використання
FR1	Графічний інтерфейс	
FR2		

Продовження таблиці 1.5.1.1.2 – Опис OUTPUT-інтерфейса користувача

FR3		
FR4		
FR5		

Продовження таблиці 1.5.1.1.2 – Опис OUTPUT-інтерфейса користувача

FR6	Графічний інтерфейс	
FR7	Графічний інтерфейс	

1.5.1.2 Опис інтерфейсу із зовнішніми пристроями

Таблиця 1.5.1.2 – Опис інтерфейсу із зовнішніми пристроями

Ідентифікатор функції	Зовнішній пристрій
FR1	Смартфон, ПК, годинник
FR2	Смартфон, ПК, годинник
FR3	Смартфон, ПК, годинник
FR4	Смартфон, ПК, годинник
FR5	Смартфон, ПК, годинник
FR6	Смартфон, ПК, годинник
FR7	Смартфон, ПК, годинник

1.5.1.3 Опис програмних інтерфейсів

Для реалізації більшості функцій програмного продукту необхідно:

- HTML.
- CSS.
- JavaScript.
- Java.

1.5.1.4 Опис інтерфейсів передачі інформації

Рекомендується використовувати

Провідні інтерфейси:

1. Ethernet
2. GigabitEthernet

Бездротові інтерфейси:

1. Wi-Fi

1.5.1.5 Опис атрибутів продуктивності

Ідентифікатор функції	Максимальний час реакції програмного продукту на дії користувача(секунди)
FR1	2
FR2	3
FR3	3
FR4	4
FR5	2
FR6	2
FR7	8

2 Планування процесу розробки програмного продукту

2.1 Планування ітерацій розробки програмного продукту

З метою забезпечення для вимог таких рекомендацій IEEE-стандарту, як необхідність, корисність при експлуатації, здійсненність функціональних вимог до ПП, визначте функціональні пріоритети, які будуть використані при плануванні ітерацій розробки ПП.

Таблиця 2.1 – Опис функціональних пріоритетів

Ідентифікатор функції	Функціональні залежності	Вплив на досягнення мети, %	Пріоритет функції
FR1	-		М
FR2	FR1		М
FR3	FR2		М
FR4	-		М
FR5	-		М
FR6	-		М
FR7	-		М

2.2 Концептуальний опис архітектури програмного продукту

Архітектура ПП представлена на рисунку 2.2.

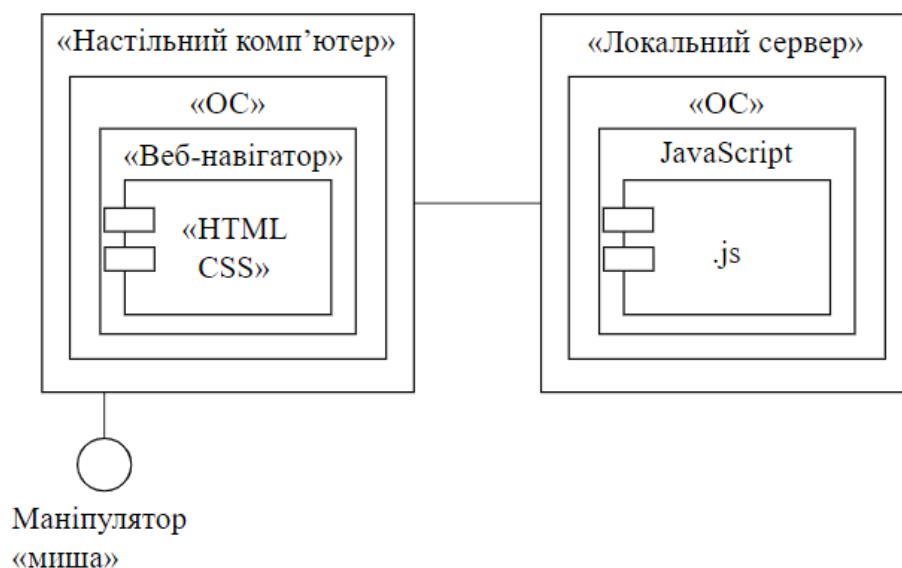


Рисунок 2.2 – Архітектура програмного продукту

2.3 План розробки програмного продукту

2.3.1 Оцінка трудомісткості розробки програмного продукту

Всі актори діляться на три типи: прості, середні і складні. Простий актор представляє зовнішню систему з чітко визначеним програмним інтерфейсом. Середній актор представляє або зовнішню систему, що взаємодіє з ПП за допомогою мережових протоколів, або особистість, що користується текстовим інтерфейсом (наприклад, алфавітно-цифровим терміналом). Складний актор представляє особистість, що користується графічним інтерфейсом. Загальна кількість акторів кожного типу помножується на відповідний ваговий коефіцієнт, потім обчислюється загальний ваговий показник (рис. 2.3.1.1).

Тип актора	Ваговий коефіцієнт
Простий	1
Середній	2
Складний	3

Рис. 2.3.1.1 – Вагові коефіцієнти акторів

Всі прецеденти діляться на три типи; прості, середні і складні в залежності від кількості кроків успішних сценаріїв (основних і альтернативних). Загальна кількість прецедентів кожного типу помножується на відповідний ваговий коефіцієнт, потім обчислюється загальний ваговий показник (рис. 2.3.1.2).

Тип прецедента	Кількість кроків сценарію	Ваговий коефіцієнт
Простий	≤ 3	5
Середній	4-7	10
Складний	> 7	15

Рис. 2.3.1.2 – Вагові коефіцієнти прецедентів

$$UCPP = 3 \cdot 1 + 8 \cdot 5 = 43$$

Технічна складність проекту (TCF – Technical Complexity Factor) обчислюється з урахуванням показників технічної складності (табл. 2.3.1.1). Кожному показнику присвоюється значення STi в діапазоні від 0 до 5: 0 означає

відсутність значимості показника для даного проекту, 5 - високу значимість).
Значення TCF обчислюється за формулою – $TCF = 0,6 + (0,01 * (ST_i * Вага_i))$

Таблиця 2.3.1.1 – TCF-таблиця

Показник	Опис показника	Sti	Вага
T1	Розподільна система	0	2
T2	Висока продуктивність	3	1
T3	Робота користувачів онлайн	3	1
T4	Складна обробка даних	1	-1
T5	Повторне використання коду	4	1
T6	Простота інсталювання	5	0.5
T7	Простота використання	5	0.5
T8	Переносимість	1	2
T9	Простота внесення змін	1	1
T10	Паралелізм	0	1
T11	Спеціальні вимоги безпеки	2	1
T12	Беспосередній доступ до системи зі сторони зовнішніх користувачів	0	1
T13	Спеціальні вимоги до навчання користувачів	0	1

$TCF = 0.79$

Рівень кваліфікації розробників (EF - Environmental Factor) обчислюється з урахуванням наступних показників (табл. 2.3.1.2).

Таблиця 2.3.1.2 – Рівень кваліфікації розробників

Показник	Опис показника	Sti	Вага
F1	Знайомство з технологією	2	1.5
F2	Досвід розробки додатків	0	0.5
F3	Досвід використання ООП	0	1
F4	Наявність провідного аналітика	2	0.5
F5	Мотивація	2	1
F6	Стабільність вимог	5	2
F7	Часткова зайнятість	2	-1
F8	Складні мови програмування	1	-1

EF = 1.01
UCP = 33.51

2.3.2 Визначення дерева робіт з розробки програмного продукту

При створенні дерева робіт (Work BreakDown Structure- WBS) використовується дерево функцій.

Кожна функція 1-го рівня ієрархії перетворюється в Work Package (WP)

Кожна функція 2-го рівня ієрархії перетворюється в Work Task (WT).

Для кожної задачі визначаються підзадачі - Work Sub Task (WST) з урахуванням базових процесів розробки програмних модулів: проектування, конструювання, модульне тестування, збірка та системне тестування(рис. 2.3.2).

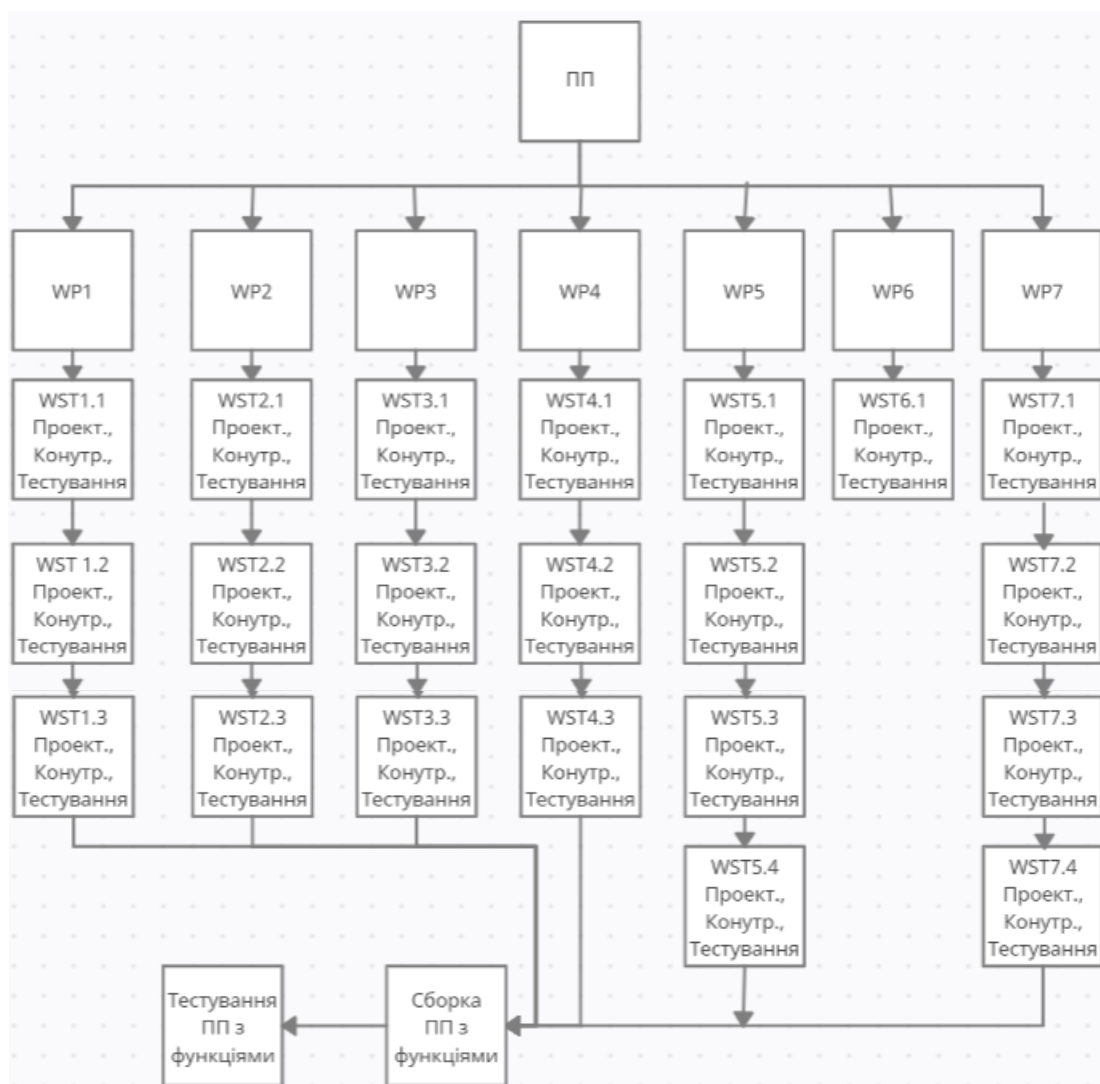


Рис. 2.3.2 – Дерево робіт з розробки ПП

2.3.3 Графік робіт з розробки програмного продукту

2.3.3.1 Таблиця з графіком робіт

Для кожної підзадачі визначається виконавець, що фіксується у вигляді таблиці, приклад якої представлено в таблиці 2.3.3.1.

Таблиця 2.3.3.1 – Таблиця графіку робіт

WST	Дата початку	Дні	Дата завершення	Виконавець
WST 1	17.10.20	3	20.10.20	Марков Д.
WST 1.1	20.10.20	3	23.10.20	Марков Д.
WST 1.2	23.10.20	4	27.10.20	Марков Д.
WST 1.3	27.10.20	3	30.10.20	Марков Д.
WST 2	30.10.20	3	02.11.20	Волков С.
WST 2.1	02.10.20	4	06.11.20	Волков С.
WST 2.2	06.10.20	3	09.11.20	Волков С.
WST 2.3	09.11.20	3	12.11.20	Волков С.
WST 3	12.11.20	4	16.11.20	Вознюк Д.
WST 3.1	17.10.20	3	20.10.20	Вознюк Д.
WST 3.2	20.10.20	3	23.10.20	Вознюк Д.
WST 3.3	23.10.20	4	27.10.20	Вознюк Д.
WST 4	27.10.20	3	30.10.20	Волков С.
WST 4.1	30.10.20	3	02.11.20	Волков С.
WST 4.2	02.10.20	4	06.11.20	Волков С.
WST 4.3	06.10.20	3	09.11.20	Волков С.
WST 5	09.11.20	3	12.11.20	Вознюк Д.
WST 5.1	12.11.20	4	16.11.20	Вознюк Д.
WST 5.2	17.10.20	3	20.10.20	Вознюк Д.
WST 5.3	20.10.20	3	23.10.20	Вознюк Д.
WST 5.4	23.10.20	4	27.10.20	Вознюк Д.
WST 6	27.10.20	3	30.10.20	Марков Д.
WST 6.1	30.10.20	3	02.11.20	Марков Д.
WST 7	02.10.20	4	06.11.20	Марков Д.
WST 7.1	06.10.20	3	09.11.20	Вознюк Д.
WST 7.2	09.11.20	3	12.11.20	Волков С.
WST 7.3	12.11.20	4	16.11.20	Волков С.
WST 7.4	17.10.20	3	20.10.20	Вознюк Д.

2.3.3.2 Діаграма Ганта

Діаграма Ганта (складається із смуг (вісь Y), орієнтованих уздовж осі часу (вісь X)). Кожна смуга – окрема підзадача в проекті, її кінці - моменти початку і завершення роботи, її протяжність - тривалість роботи. Мета діаграми - візуально показати послідовність процесів та можливість паралельного виконання робіт(таблиця 2.3.3.2.1).

Таблиця 2.3.3.2.1. Діаграма Ганта

W ST	Дата поча тку	Д ні	Дата заверш ення	Викона вель	17. 10	20. 10	23. 10	27. 10	30. 10	02. 11	06. 11	09. 11	12. 11
W ST 1	17.1 0.20	3	20.10.2 0	Марко в Д.									
W ST 1.1	20.1 0.20	3	23.10.2 0	Марко в Д.									
W ST 1.2	23.1 0.20	4	27.10.2 0	Марко в Д.									
W ST 1.3	27.1 0.20	3	30.10.2 0	Марко в Д.									
W ST 2	30.1 0.20	3	02.11.2 0	Волков С.									
W ST 2.1	02.1 0.20	4	06.11.2 0	Волков С.									
W ST 2.2	06.1 0.20	3	09.11.2 0	Волков С.									
W ST 2.3	09.1 1.20	3	12.11.2 0	Волков С.									
W ST 3	12.1 1.20	4	16.11.2 0	Возню к Д.									
W ST 3.1	17.1 0.20	3	20.10.2 0	Возню к Д.									

Продовження таблиці 2.3.3.2.1. Діаграма Ганта

W ST 3.2	20.1 0.20	3	23.10.2 0	Возню к Д.									
W ST 3.3	23.1 0.20	4	27.10.2 0	Возню к Д.									
W ST 4	27.1 0.20	3	30.10.2 0	Волков С.									
W ST 4.1	30.1 0.20	3	02.11.2 0	Волков С.									
W ST 4.2	02.1 0.20	4	06.11.2 0	Волков С.									
W ST 4.3	06.1 0.20	3	09.11.2 0	Волков С.									
W ST 5	09.1 1.20	3	12.11.2 0	Возню к Д.									
W ST 5.1	12.1 1.20	4	16.11.2 0	Возню к Д.									
W ST 5.2	17.1 0.20	3	20.10.2 0	Возню к Д.									
W ST 5.3	20.1 0.20	3	23.10.2 0	Возню к Д.									
W ST 5.4	23.1 0.20	4	27.10.2 0	Возню к Д.									
W ST 6	27.1 0.20	3	30.10.2 0	Марко в Д.									
W ST 6.1	30.1 0.20	3	02.11.2 0	Марко в Д.									
W ST 7	02.1 0.20	4	06.11.2 0	Марко в Д.									

Продовження таблиці 2.3.3.2.1. Діаграма Ганта

W ST 7.1	06.1 0.20	3	09.11.2 0	Возню к Д.									
W ST 7.2	09.1 1.20	3	12.11.2 0	Волков С.									
W ST 7.3	12.1 1.20	4	16.11.2 0	Волков С.									
W ST 7.4	17.1 0.20	3	20.10.2 0	Возню к Д.									

3 Проектування програмного продукту

3.1 Концептуальне та логічне проектування структур даних програмного продукту

3.1.1 Концептуальне проектування на основі UML-діаграми концептуальних класів

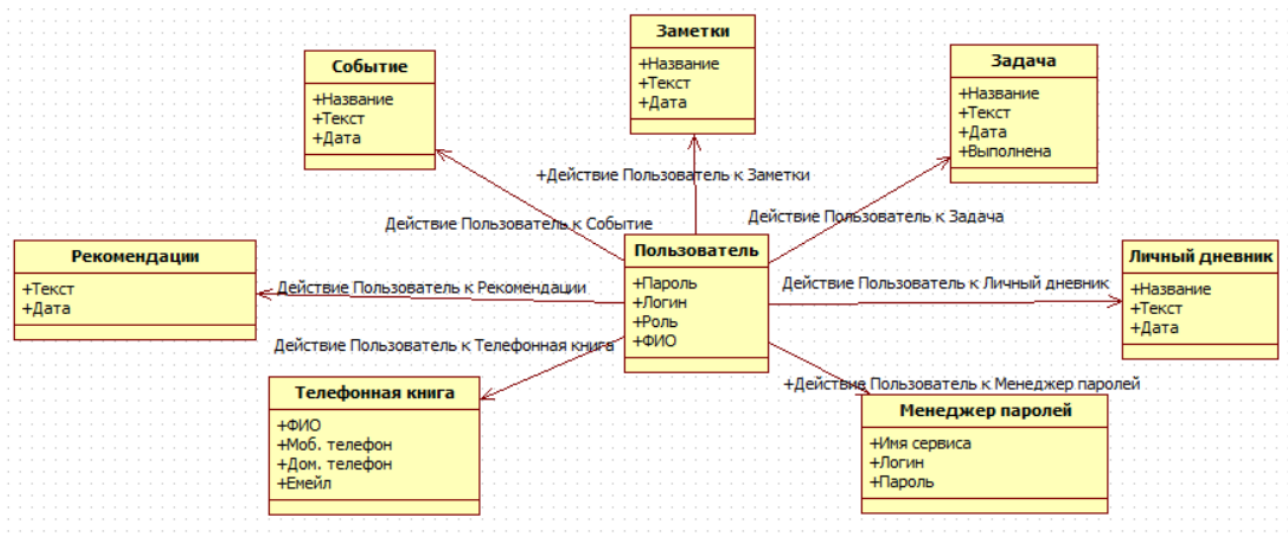


Рис. 3.1.1 – UML-діаграма концептуальних класів

3.1.2 Логічне проектування структур даних

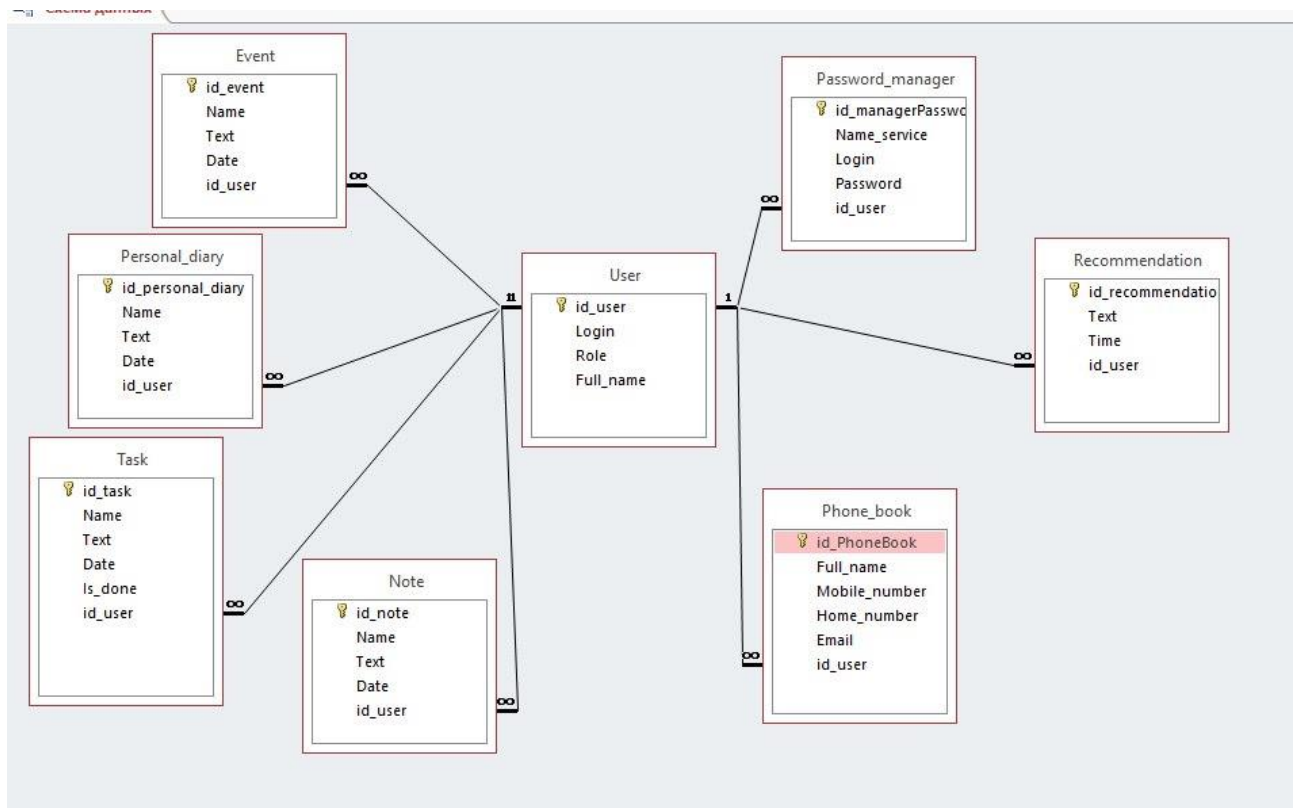


Рис. 3.1.2 – UML-діаграма структурних класів

3.2 Проектування програмних класів

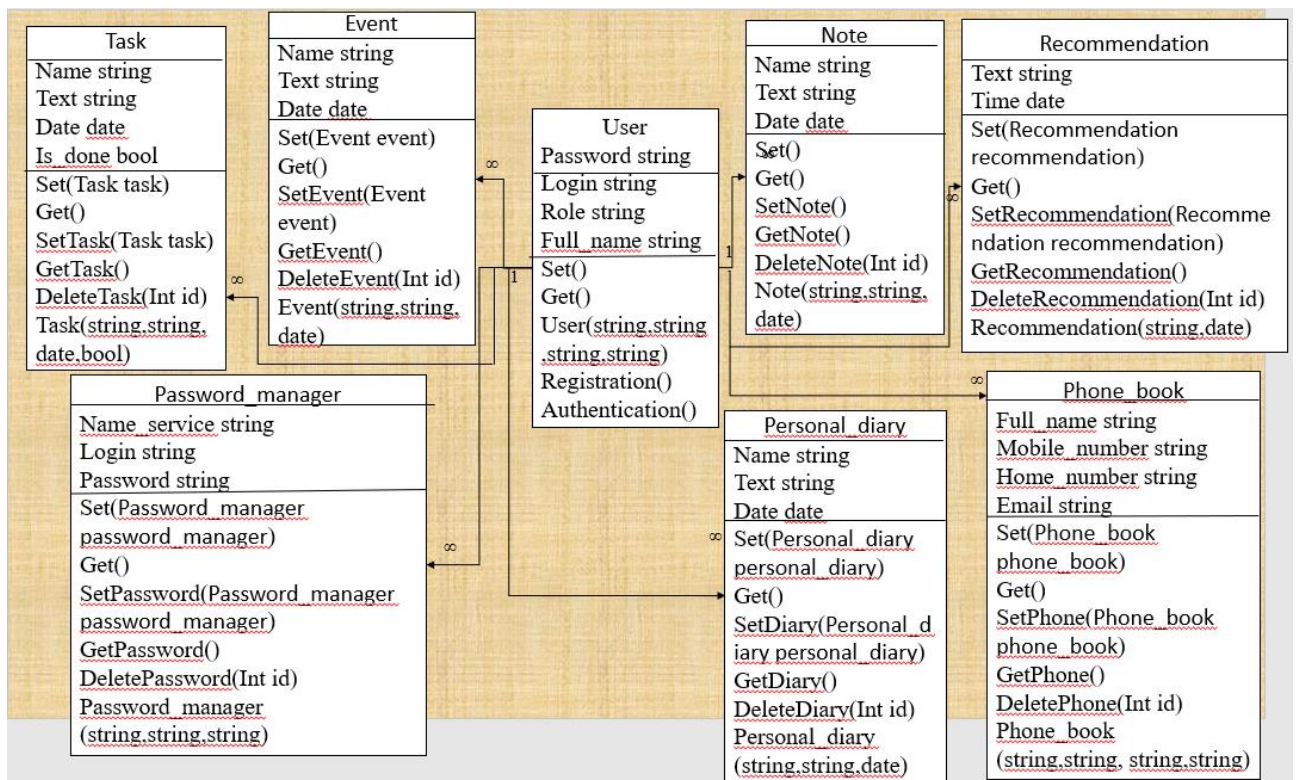


Рис. 3.2 – Діаграма програмних класів

3.3 Проектування алгоритмів роботи методів програмних класів

Алгоритм методу Authentication(string Login, string password) зображений на рисунку 3.3.1.

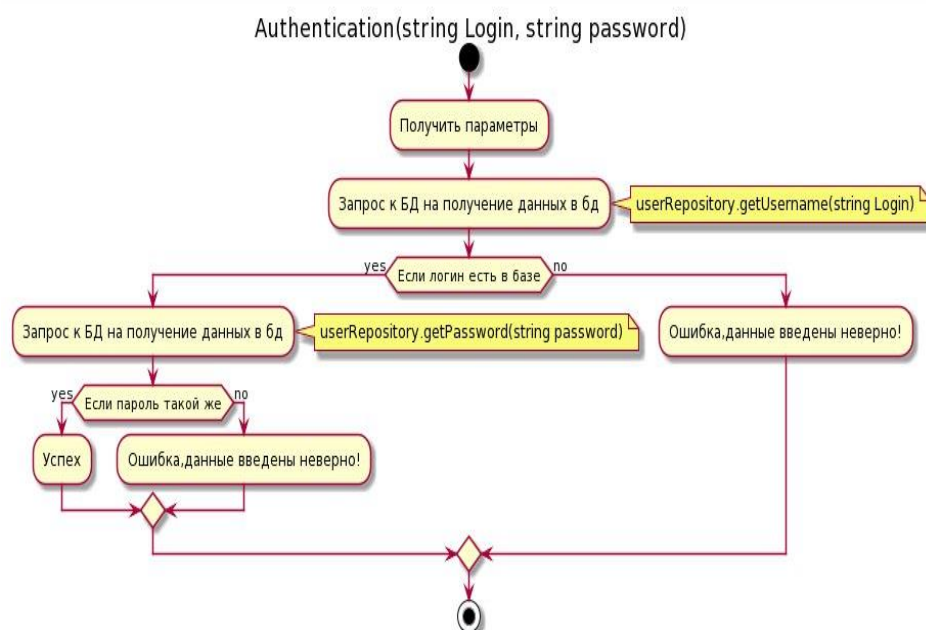


Рисунок 3.3.1 – Алгоритм методу Authentication(User user)

Код до метода Authentication(string Login, string password):

```
@startuml
start
title Authentication(string Login, string password)
:Получить параметры;
:Запрос к БД на получение данных в бд;
note right
userRepository.getUsername(string Login)
end note
if (Если логин есть в базе) then (yes)
:Запрос к БД на получение данных в бд;
note right
userRepository.getPassword(string password)
end note
if (Если пароль такой же) then (yes)
:Успех;
else (no)
:Ошибка, данные введены неверно!;
endif
else (no)
:Ошибка, данные введены неверно!;
endif
stop
@enduml
```

Алгоритм метода getEvent() изображений на рисунку 3.3.2.

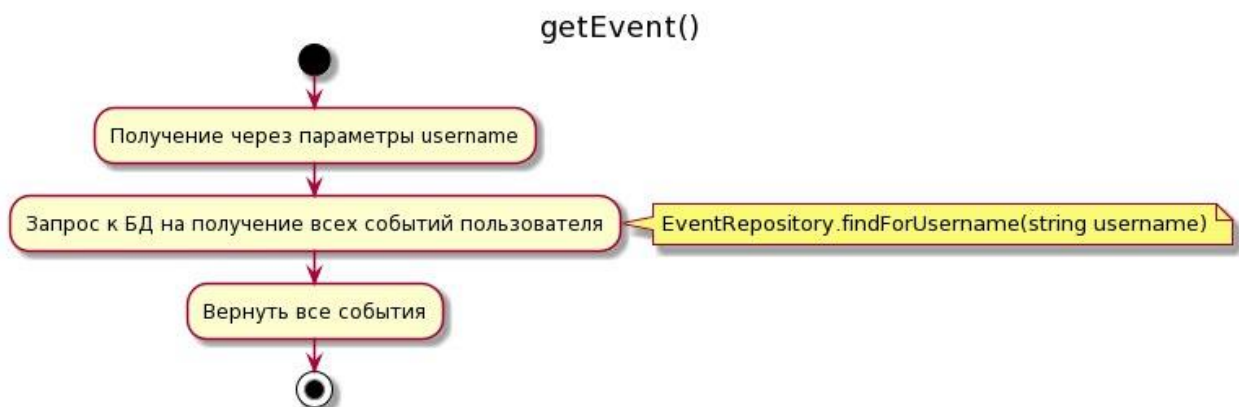


Рисунок 3.3.2 – Алгоритм метода Registration(User user)

Код до метода `getEvent()`:

```
@startuml
start
title getEvent()
:Получение через параметры username;
:Запрос к БД на получение всех событий пользователя;
note right
EventRepository.findForUsername(string username)
end note
:Вернуть все события;
stop
@enduml
```

Алгоритм метода `getNote()` изображений на рисунку 3.3.3.

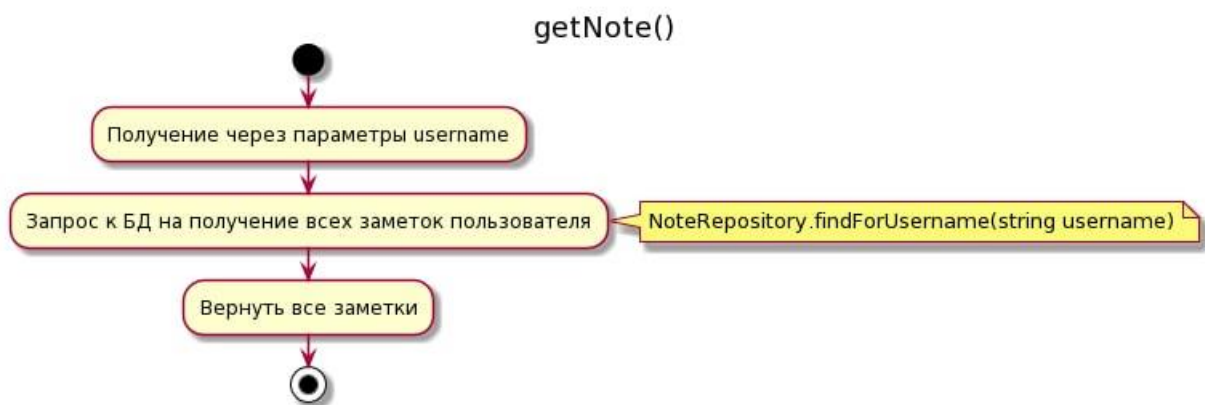


Рисунок 3.3.3 – Алгоритм метода `getNote()`

Код до метода `getNote()`:

```
@startuml
start
title getNote()
:Получение через параметры username;
:Запрос к БД на получение всех заметок пользователя;
note right
NoteRepository.findForUsername(string username)
end note
:Вернуть все заметки;
stop
@enduml
```

Алгоритм метода `getRecommendation()` изображен на рисунке 3.3.4.

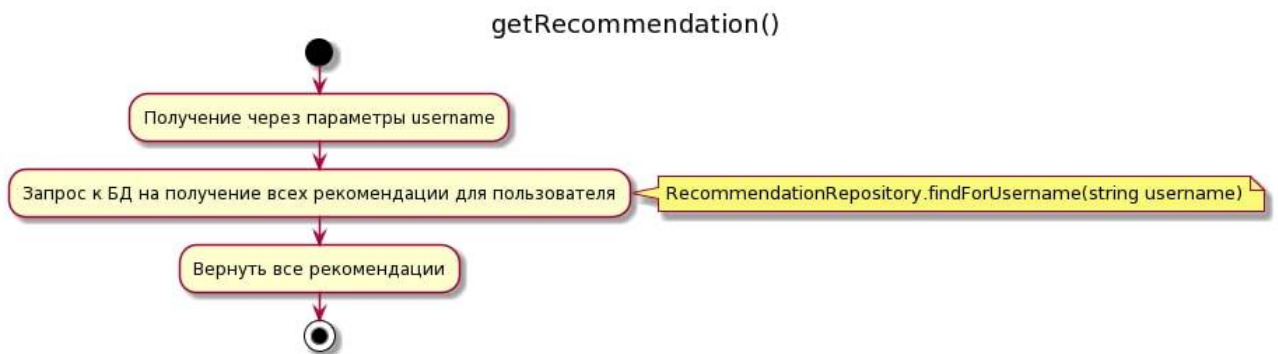


Рисунок 3.3.4 – Алгоритм метода `getRecommendation()`

Код для метода `getRecommendation()`:

```
@startuml
start
title getRecommendation()
:Получение через параметры username;
:Запрос к БД на получение всех рекомендаций для пользователя;
note right
RecommendationRepository.findForUsername(string username)
end note
:Вернуть все рекомендации;
stop
@enduml
```

Алгоритм метода `getDiary()` изображен на рисунке 3.3.5.



Рисунок 3.3.5 – Алгоритм метода getDiary()

Код до метода getDiary():

```
@startuml
start
title getDiary()
:Получение через параметры username;
:Запрос к БД на получение всех личных записей пользователя;
note right
DiaryRepository.findForUsername(string username)
end note
:Вернуть все личные записи пользователя;
stop
@enduml
```

Алгоритм метода getPasswordManadger() изображений на рисунку 3.3.6.



Рисунок 3.3.6 – Алгоритм метода getPasswordManadger()

Код до метода getPasswordManadger():

```
@startuml
start
title getPasswordManadger()
:Получение через параметры username;
:Запрос к БД на получение всех записей с менеджера паролей пользователя;
note right
PasswordManadgerRepository.findForUsername(string username)
end note
:Вернуть все записи менеджера паролей пользователя;
Stop
@enduml
```

Алгоритм метода setNote(Note note) изображений на рисунку 3.3.7.

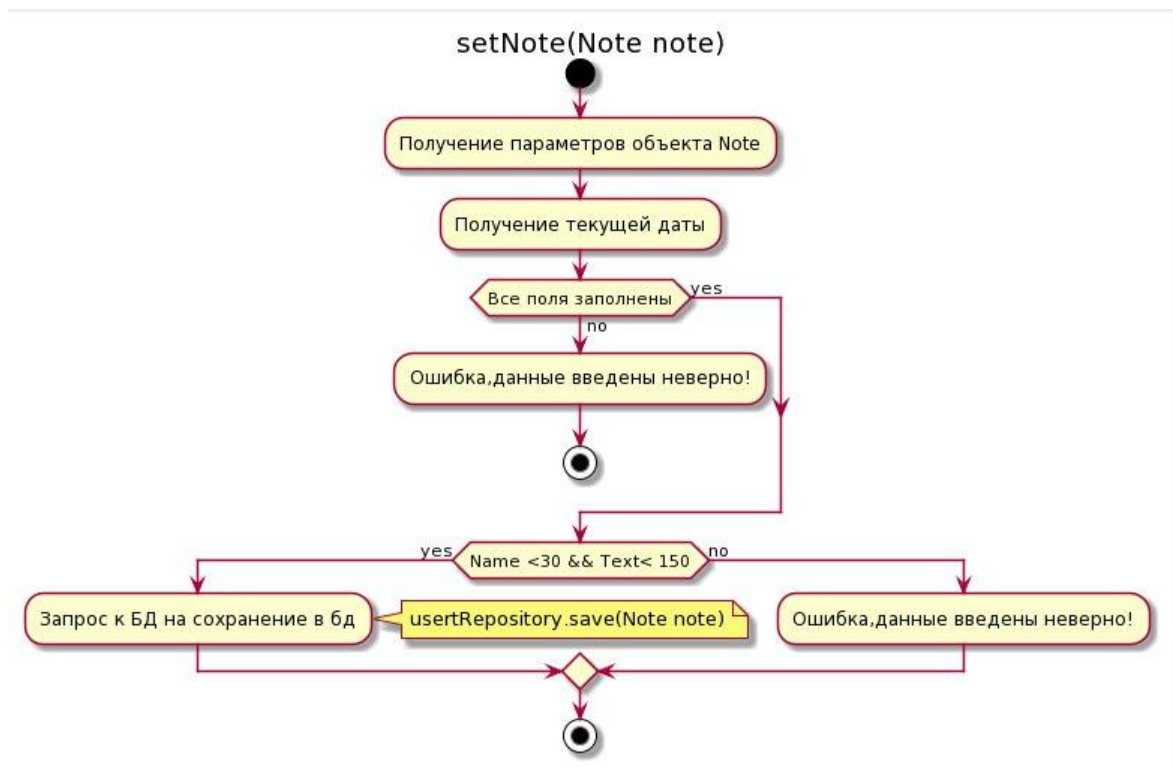


Рисунок 3.3.7 – Алгоритм метода setNote(Note note)

Код до метода setNote(Note note):

```

@startuml
start
title setNote(Note note)
:Получение параметров объекта Note;
:Получение текущей даты;
if (Все поля заполнены) then (yes)
else (no)
:Ошибка, данные введены неверно!;
stop
endif
if (Name < 30 && Text < 150) then (yes)
:Запрос к БД на сохранение в бд;
note right
usertRepository.save(Note note)
end note
else (no)
:Ошибка, данные введены неверно!;
endif
stop @enduml
    
```

Алгоритм метода setEvent(Event event) изображений на рисунку 3.3.8.

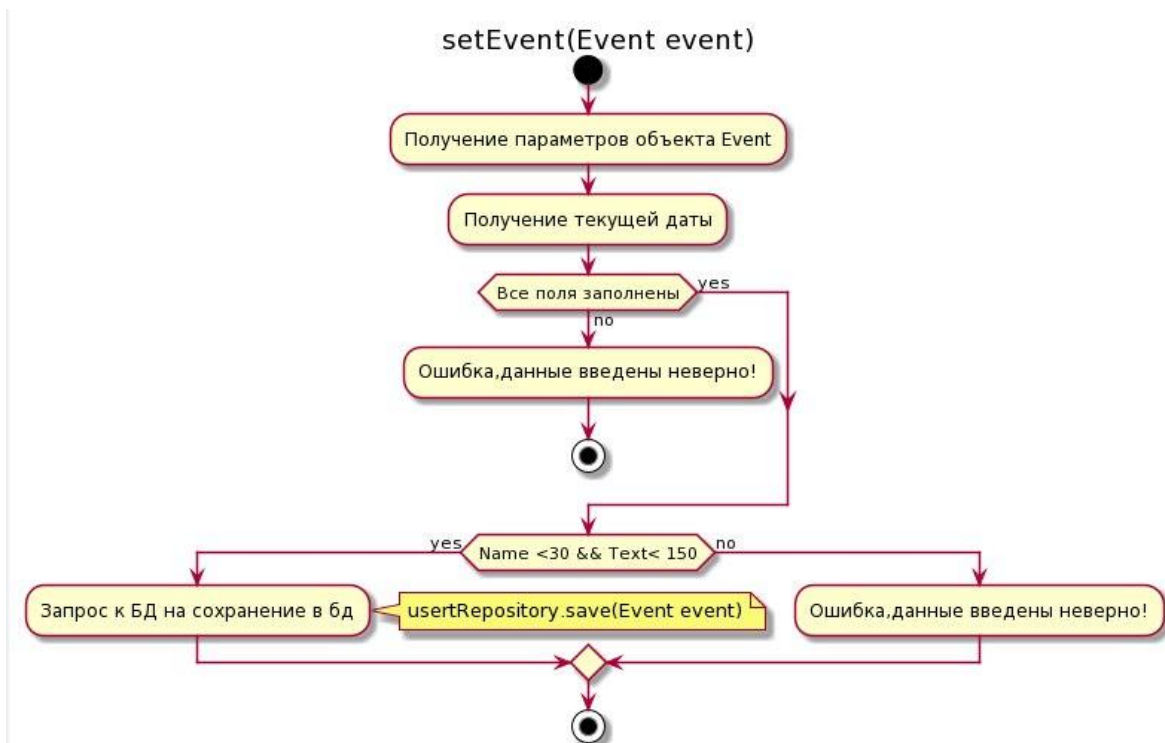


Рисунок 3.3.8 – Алгоритм метода setEvent(Event event)

Код до метода setEvent(Event event):

```

@startuml
start
title setEvent(Event event)
:Получение параметров объекта Event;
:Получение текущей даты;
if (Все поля заполнены) then (yes)
else (no)
:Ошибка, данные введены неверно!;
stop
endif
if (Name <30 && Text < 150) then (yes)
:Запрос к БД на сохранение в бд;
note right
userRepository.save(Event event)
end note
else (no)
:Ошибка, данные введены неверно!;
endif
stop
@enduml
  
```

Алгоритм метода setRecommendation(Recommendation recommendation) изображений на рисунку 3.3.9.

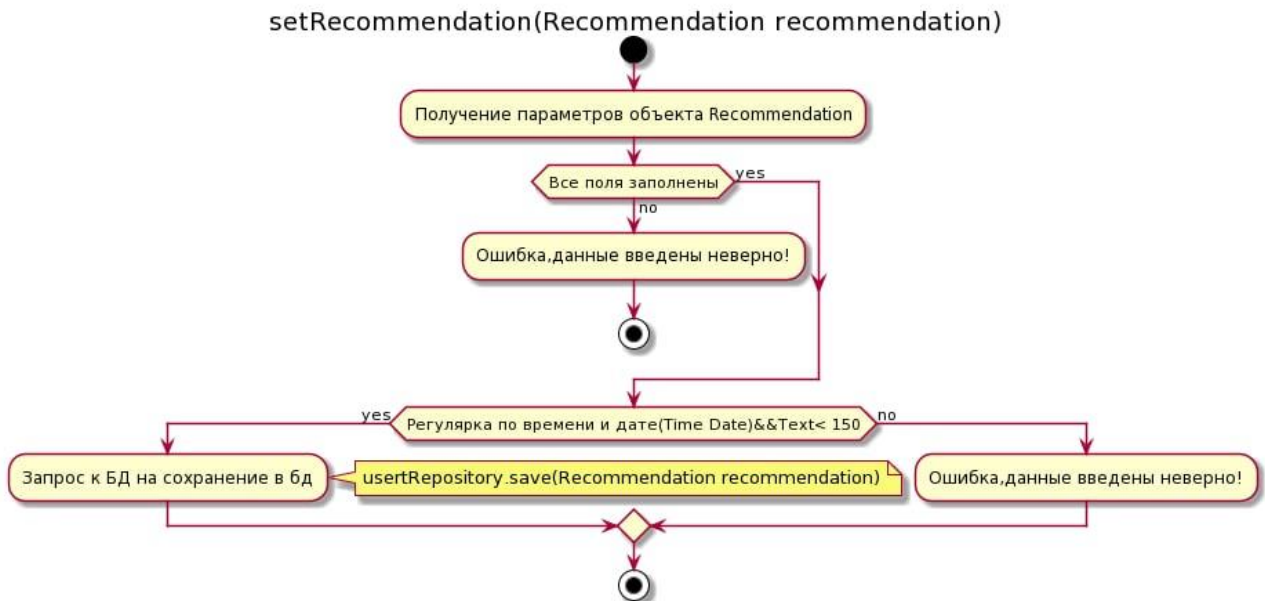


Рисунок 3.3.9 – Алгоритм метода setRecommendation(Recommendation recommendation)

Код до метода setRecommendation(Recommendation recommendation):

```

@startuml
start
title setRecommendation(Recommendation recommendation)
:Получение параметров объекта Recommendation;
if (Все поля заполнены) then (yes)
else (no)
:Ошибка, данные введены неверно!;
stop
endif
if (Регулярка по времени и дате(Time Date)&&Text< 150) then (yes)
:Запрос к БД на сохранение в бд;
note right
userRepository.save(Recommendation recommendation)
end note
else (no)
:Ошибка, данные введены неверно!;
endif
stop
@enduml
  
```

Алгоритм метода setPersonalDiary(PersonalDiary personalDiary)
зображений на рисунку 3.3.10.

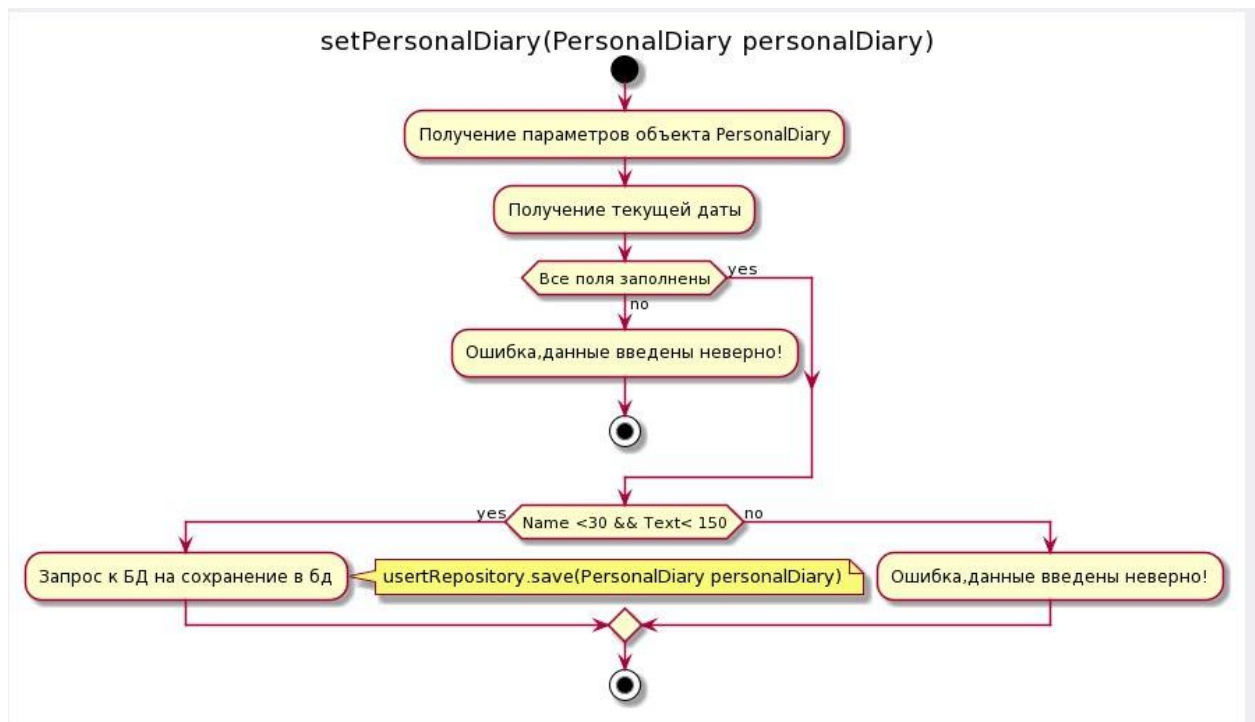


Рисунок 3.3.10 – Алгоритм метода setPersonalDiary(PersonalDiary personalDiary)
Код до метода setPersonalDiary(PersonalDiary personalDiary):

```

@startuml
start
title setPersonalDiary(PersonalDiary personalDiary)
:Получение параметров объекта PersonalDiary;
:Получение текущей даты;
if (Все поля заполнены) then (yes)
else (no)
:Ошибка, данные введены неверно!;
stop
endif
if (Name <30 && Text < 150) then (yes)
:Запрос к БД на сохранение в бд;
note right
userRepository.save(PersonalDiary personalDiary)
end note
else (no)
:Ошибка, данные введены неверно!;
endif
stop@enduml@enduml
  
```

Алгоритм метода setPasswordManager(PasswordManager passwordManager) изображений на рисунку 3.3.11.

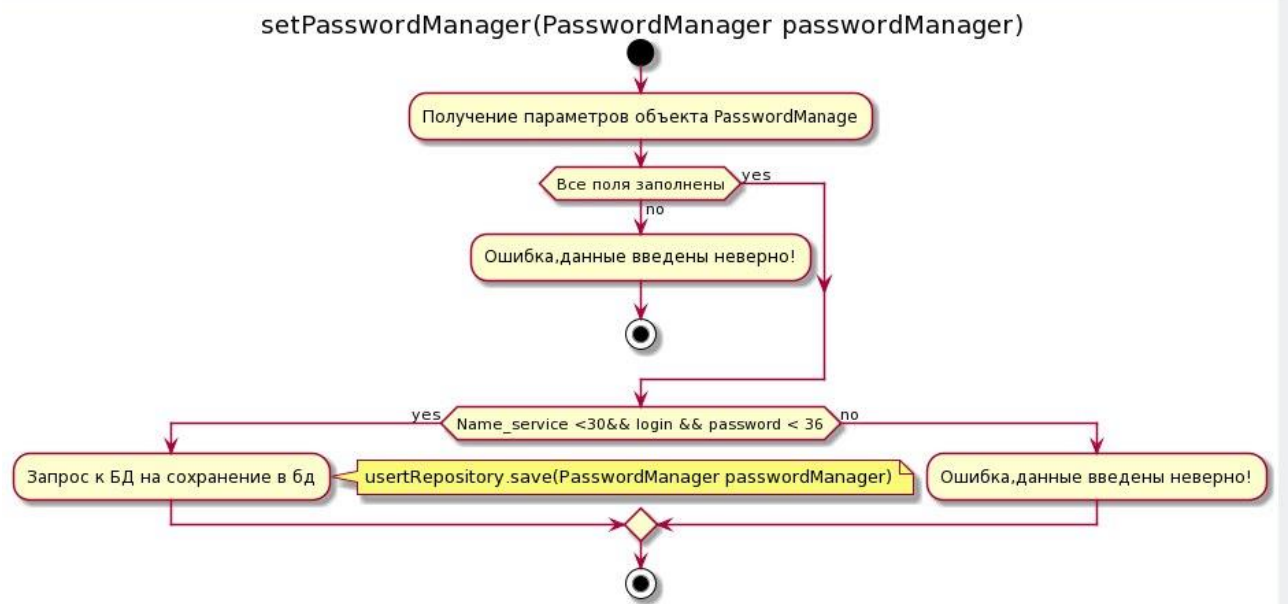


Рисунок 3.3.11 – Алгоритм метода setPasswordManager(PasswordManager passwordManager)

Код до метода setPasswordManager(PasswordManager passwordManager):

```

@startuml
start
title setPasswordManager(PasswordManager passwordManager)
:Получение параметров объекта PasswordManage;
if (Все поля заполнены) then (yes)
else (no)
:Ошибка, данные введены неверно!;
stop
endif
if (Name_service < 30 && login && password < 36) then (yes)
:Запрос к БД на сохранение в бд;
note right
userRepository.save(PasswordManager passwordManager)
end note
else (no)
:Ошибка, данные введены неверно!;
endif
stop
@enduml

```


Алгоритм метода DeleteEvent(int id) изображен на рисунке 3.3.12.



Рисунок 3.3.12 – Алгоритм метода DeleteEvent(int id)

Код до метода DeleteEvent(int id):

```
@startuml
start
title DeleteEvent(int id)
:Получение параметров;
:Запрос к БД на удаления события по id;
note right
EventRepository.deleteId(int Id)
end note
:Успешное удаление события;
stop
@enduml
```

Алгоритм метода DeleteNote(int id) изображен на рисунке 3.3.13.

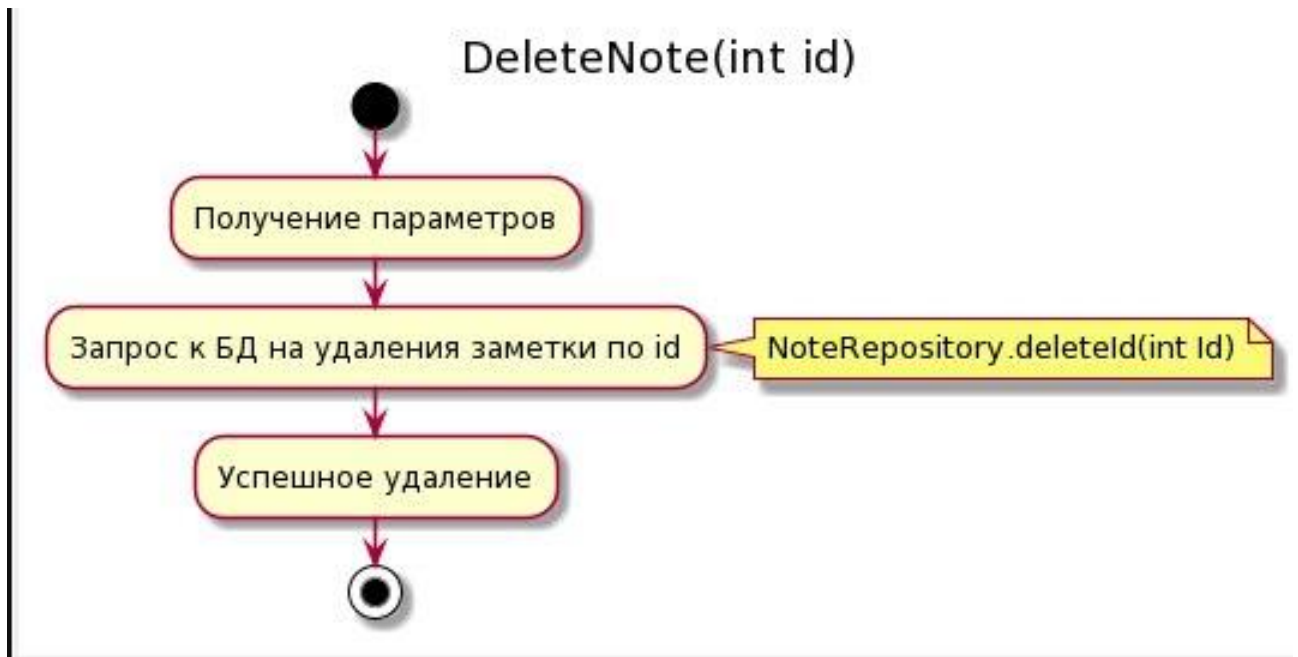


Рисунок 3.3.13 – Алгоритм методу DeleteNote(int id)

Код до методу DeleteNote(int id):

```

@startuml
start
title DeleteNote(int id)
:Получение параметров;
:Запрос к БД на удаления заметки по id;
note right
NoteRepository.deleteId(int Id)
end note
:Успешное удаление;
Stop @enduml@enduml
  
```

Алгоритм методу DeleteRecommendation (int id) изображений на рисунку

3.3.14.



Рисунок 3.3.14 – Алгоритм методу DeleteRecommendation (int id)

Код до методу DeleteRecommendation (int id):

```

@startuml
start
title DeleteRecommendation(int id)
:Получение параметров;
:Запрос к БД на удаления рекомендации по id;
note right
RecommendationRepository.deleteId(int Id)
end note
:Успешное удаление;
stop
@enduml

```

Алгоритм методу DeletePasswordManager(int id) изображений на рисунку 3.3.15.



Рисунок 3.3.15 – Алгоритм методу DeletePasswordManager(int id)
Код до методу DeletePasswordManager(int id):

```

@startuml
start
title DeletePasswordManager(int id)
:Получение параметров;
:Запрос к БД на удаления по id;
note right
RecommendationRepository.deleteId(int Id)
end note
:Успешное удаление;
stop
@enduml

```

Алгоритм методу DeletePersonalDiary (int id) зображений на рисунку 3.3.16.



Рисунок 3.3.16 – Алгоритм методу DeletePersonalDiary (int id)

Код до методу DeletePersonalDiary (int id):

```
@startuml
start
title DeletePersonalDiary(int id)
:Получение параметров;
:Запрос к БД на удаления по id;
note right
DeletePersonalDiaryRepository.deleteId(int Id)
end note
:Успешное удаление; stop
@enduml
```

3.4 Проектування тестових наборів методів програмних класів

У відповідності із планами на розробки програмного продукту між всіма учасниками проектної команди необхідно розподілити розробку функцій (методів класів) та відповідних їм програмних модулів.

Для кожної функції (методу класу) необхідно створити тестові набори (Рис 3.4.1-3.4.5.), використовуючи будь-який метод чорного ящика та представити їх у вигляді таблиці.

Название функции	Номер теста	Описание вводимых значений	Описание ожидаемого результата
Authentication(string password, string login)	1	user_name = user1 password = Qwerty	Пользователь успешно вошел в свой аккаунт
Authentication(string password, string login)	2	user_name = Qwerty password = user1	Ошибка: «Логин или пароль введен неверно»
Authentication(string password, string login)	3	user_name = qser1 password = Werty	Ошибка: «Логин или пароль введен неверно»

Рис. 3.4.1 Таблица тестів1

Название функции	Номер теста	Описание вводимых значений	Описание ожидаемого результата
SetEvent(user)	10	Name = Собрание Text = «Собрание в школе» Date = 12/01/2021	Событие успешно создано
SetEvent(user)	11	Name = Собрание Text = «Собрание в школе» Date = 1/12/2020	Ошибка: «Выберите другую дату»
SetNote(user)	12	Name = «Книги» Text = «Необходимые книги для прочтения...»	Заметка успешно сохранена

Рис. 3.4.2 Таблица тестів2

Название функции	Номер теста	Описание вводимых значений	Описание ожидаемого результата
SetRecommendation(user)	13	Text = «Зарядка» Date = 5/12/20	Рекомендация успешно создано
SetDiary(user)	14	Name = «Дне/в\ник»	Ошибка: «Пожалуйста введите корректное название дневника»
DeleteNote(id)	15	*Выбираем любую заметку*	Заметка успешно удалена

Рис. 3.4.3 Таблица тестів3

Название функции	Номер теста	Описание вводимых значений	Описание ожидаемого результата
DeleteRecommendation(id)	16	*Выбираем любую рекомендацию*	Рекомендация успешно удалена
DeleteDiary(id)	17	*Выбираем дневник*	Дневник успешно удален

Рис. 3.4.4 Таблица тестів4

Название функции	Номер теста	Описание вводимых значений	Описание ожидаемого результата
SetPassword(password manager)	19	Name = «Работа» Login = user2 Password = Qwery1	Пароль успешно сохранен
GetDiary()	20	*Выбираем дневник*	Дневник открылся

Рис. 3.4.5 Таблица тестів5

4 Конструювання програмного продукту

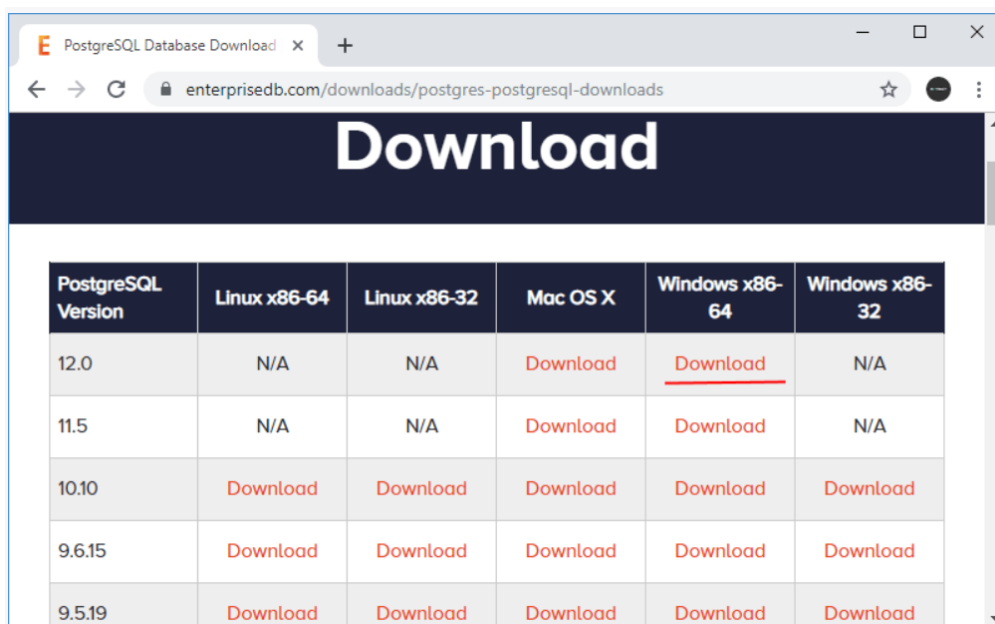
4.1. Особливості інсталяції та роботи з СУБД

PostgreSQL - вільна об'єктно-реляційна система управління базами даних.

З тих пір вийшло безліч версій postgresql. Поточною версією є версія 12.

PostgreSQL підтримується для всіх основних операційних систем - Windows, Linux, MacOS.

На сторінці <https://www.postgresql.org/download/> можна знайти посилання на завантаження різних дистрибутивів для різних операційних систем. Зокрема, для завантаження дистрибутива для Windows треба перейти на сторінку <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads> і вказати всі необхідні опції для завантаження: версію postgres і операційну систему.



PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
12.0	N/A	N/A	Download	Download	N/A
11.5	N/A	N/A	Download	Download	N/A
10.10	Download	Download	Download	Download	Download
9.6.15	Download	Download	Download	Download	Download
9.5.19	Download	Download	Download	Download	Download

Рис. 4.1.1 Посилання на завантаження різних дистрибутивів для різних операційних систем

При установці запам'ятаємо пароль, так як він буде потрібно для підключення до сервера. Потім потрібно буде встановити порт, по якому буде запускатися сервер. Можна залишити порт за замовчуванням:

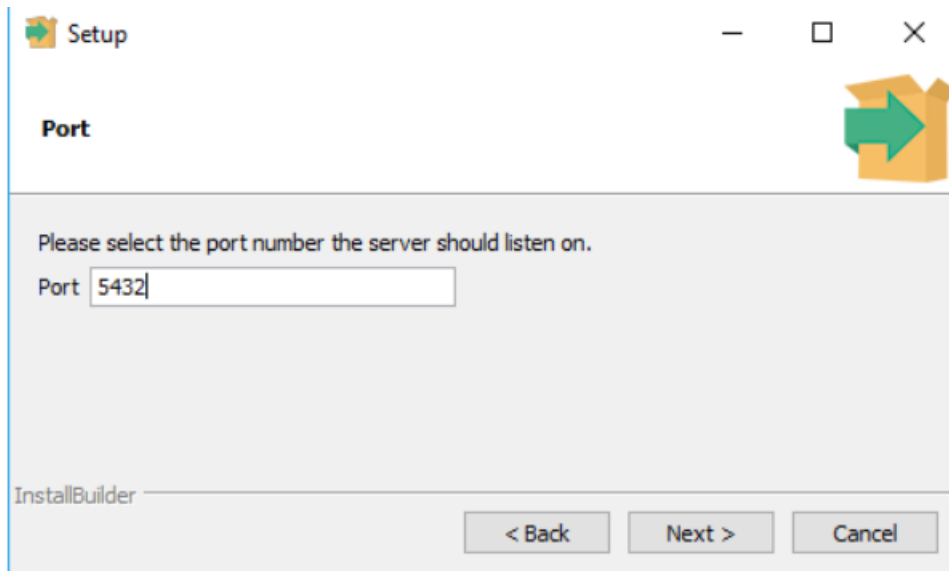


Рис. 4.1.2 Встановлення стандартного порту 5342

І після завершення установки ми побачимо наступне вікно, і для виходу натиснемо на кнопку Finish:

Таким чином, сервер PostgreSQL встановлений, і ми можемо починати з ним працювати.

Для спрощення адміністрування на сервері postgresql в базовий комплект установки входить такий інструмент як pgAdmin. Він являє графічний клієнт для роботи з сервером, через який ми в зручному вигляді можемо створювати, видаляти, змінювати бази даних і управляти ними. Так, на Windows після установки ми можемо знайти значок pgAdmin в меню Пуск і запустити його:

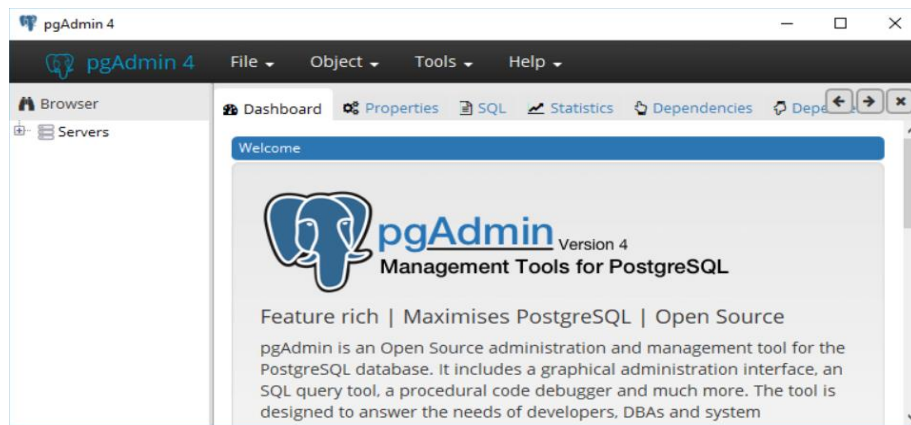
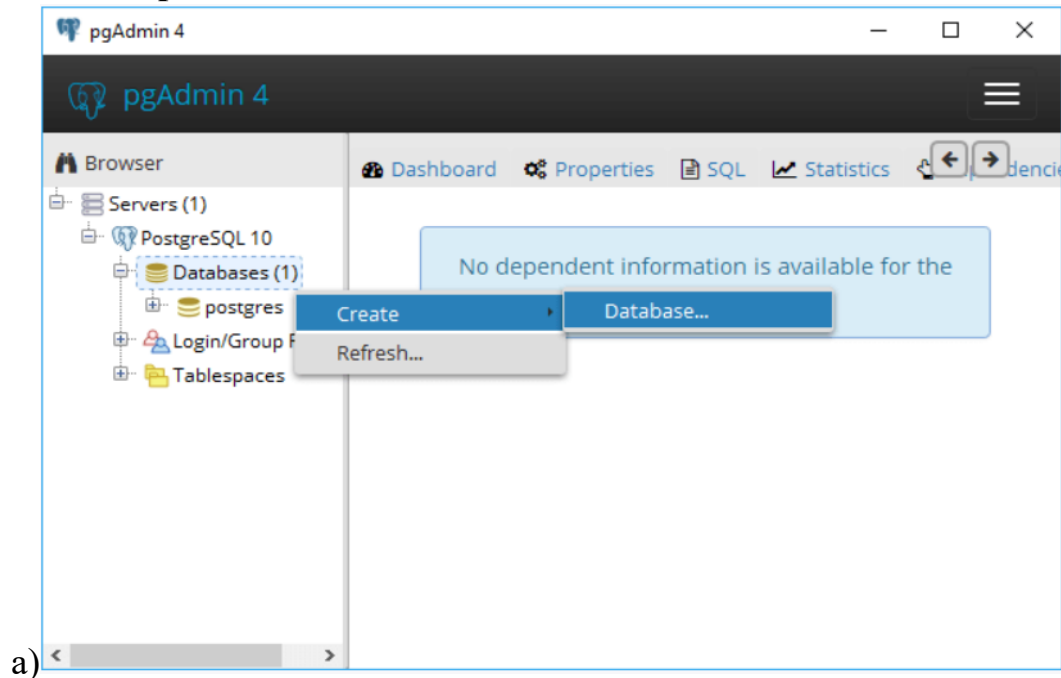


Рис. 4.1.3 Стартове вікно pgAdmin

4.2. Створення структур Даних

4.2.1. Створення бази даних.



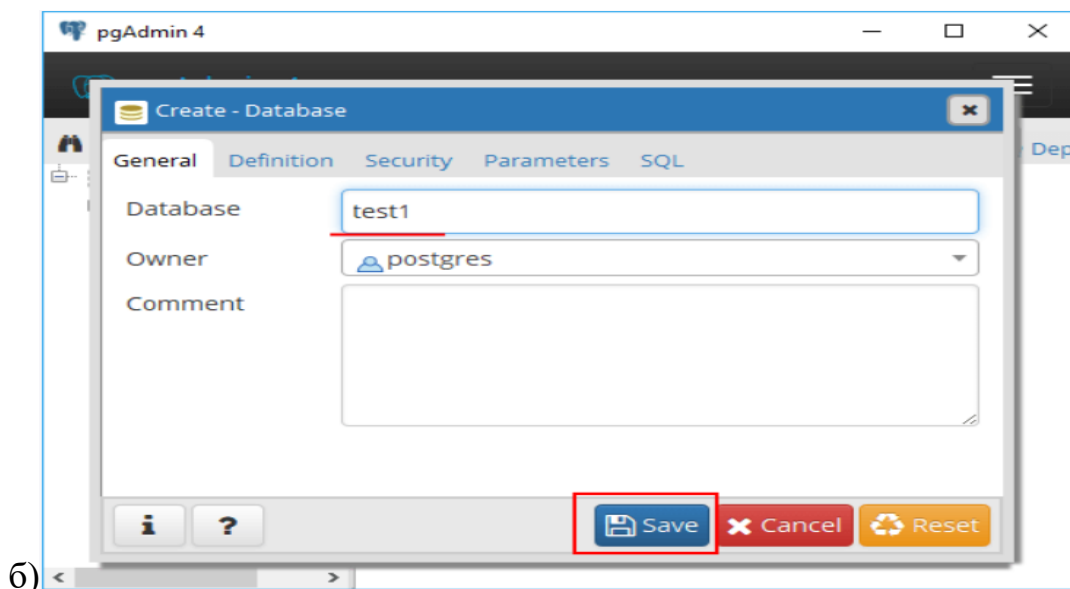


Рис.4.2.1.1 а) Створення БД; б) Введення необхідних параметрів БД

4.2.2. Створення таблиць.

За допомогою анотації `@entity` програма розуміє, що треба створити в базі даних таблиці з параметрами та зв'язками описаними у класі з анотацією `@entity`.

Таблиця 4.2.2.1. Опис анотації

Анотація	Опис
@NoArgsConstructor @AllArgsConstructor	Создание пустого конструктора, конструктора включающего все final поля, либо конструктора включающего все возможные поля
@Builder	Неализация паттерна bulder, <u>Singular</u> – используется для объектов в единственном экземпляре (добавления элемента в коллекции и т.п.)
@Data	Генерация всех служебных методов, заменяет сразу команды <code>@ToString</code> ,

	<u>@EqualsAndHashCode</u> , <u>Getter</u> , <u>Setter</u> , <u>@RequiredArgsConstructor</u>
@Entity	Эта аннотация указывает Hibernate, что данный класс является сущностью (entity bean). Такой класс должен иметь конструктор по-умолчанию (пустой конструктор).
@Table	С помощью этой аннотации мы говорим Hibernate, с какой именно таблицей необходимо связать (map) данный класс. Аннотация @Table имеет различные атрибуты, с помощью которых мы можем указать <i>имя таблицы, каталог, БД и уникальность столбцов в таблице БД</i> .
@Id	С помощью аннотации @Id мы указываем <i>первичный ключ (Primary Key)</i> данного класса.

Продовження таблиці 4.2.2.1 Опис анотації

@GeneratedValue	Эта аннотация используется вместе с аннотацией @Id и определяет такие параметры, как strategy и generator.
@Column	Аннотация @Column определяет к какому столбцу в таблице БД относится конкретное поле класса (атрибут класса).

<p>Наиболее часто используемые атрибуты аннотации @Column такие:</p>	<ul style="list-style-type: none"> • name Указывает имя столбца в таблице • unique Определяет, должно ли быть данное значение уникальным • nullable Определяет, может ли данное поле быть NULL, или нет. • length Указывает, какой размер столбца (например количество символов, при использовании String).
----------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Зразок коду, який відноситься до класу Event:

```
package pl.sda.finalProject.myOrganizer.entity;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;
import java.time.LocalDate;
import java.time.LocalDateTime;

@Entity
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Event {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```

    private String name;

    private LocalDate creationDate;

    private LocalDate eventDate;

    private LocalTime eventTime;

    private int minutesBefore;

    private int hoursBefore;

    private int daysBefore;

    @ManyToOne
    private MyUser user;
}

```

4.3. Робота з IDEA

Наш програмний продукт використовує спосіб відображення сайтів засобами HTML, CSS, Java, Thymeleaf, Spring, Maven та СУБД PostgreSQL.

Для роботи в зв'язці з вищеперерахованими засобами найбільш підходить інтегроване середовище розробки програмного забезпечення IntelliJ IDEA .

Чому саме IntelliJ IDEA :

- Розумне автодоповнення, інструменти для аналізу якості коду, зручна навігація, розширені рефакторинг і форматування для Java, Groovy, Scala, HTML, CSS, JavaScript, CoffeeScript, ActionScript, LESS, XML і багатьох інших мов.
- Підтримка всіх популярних фреймворків і платформ, включаючи Java EE, Spring Framework, Grails, Play Framework, GWT, Struts, Node.js, AngularJS, Android, Flex, AIR Mobile і багатьох інших.
- Інтеграція з серверами Додатків, включаючи Tomcat, TomEE, GlassFish, JBoss, WebLogic, WebSphere, Geronimo, Resin, Jetty и Virgo.
- Інструменти для роботи з базами даних і SQL файлами, включаючи зручний клієнт і редактор для схеми бази даних.
- Фреймворки Spring
- Підтримка Thymeleaf 3.0

2.2. Створення програмної структури з урахуванням спеціалізованого Фреймворку

4.3.1Spring MVC:

Фреймворк Spring MVC забезпечує архітектуру паттерна Model - View - Controller (Модель - Відображення - Контролер) за допомогою слабо пов'язаних

готових компонентів. Патерн MVC розділяє аспекти додатки (логіку введення, бізнес-логіку і логіку UI), забезпечуючи при цьому вільну зв'язок між ними.

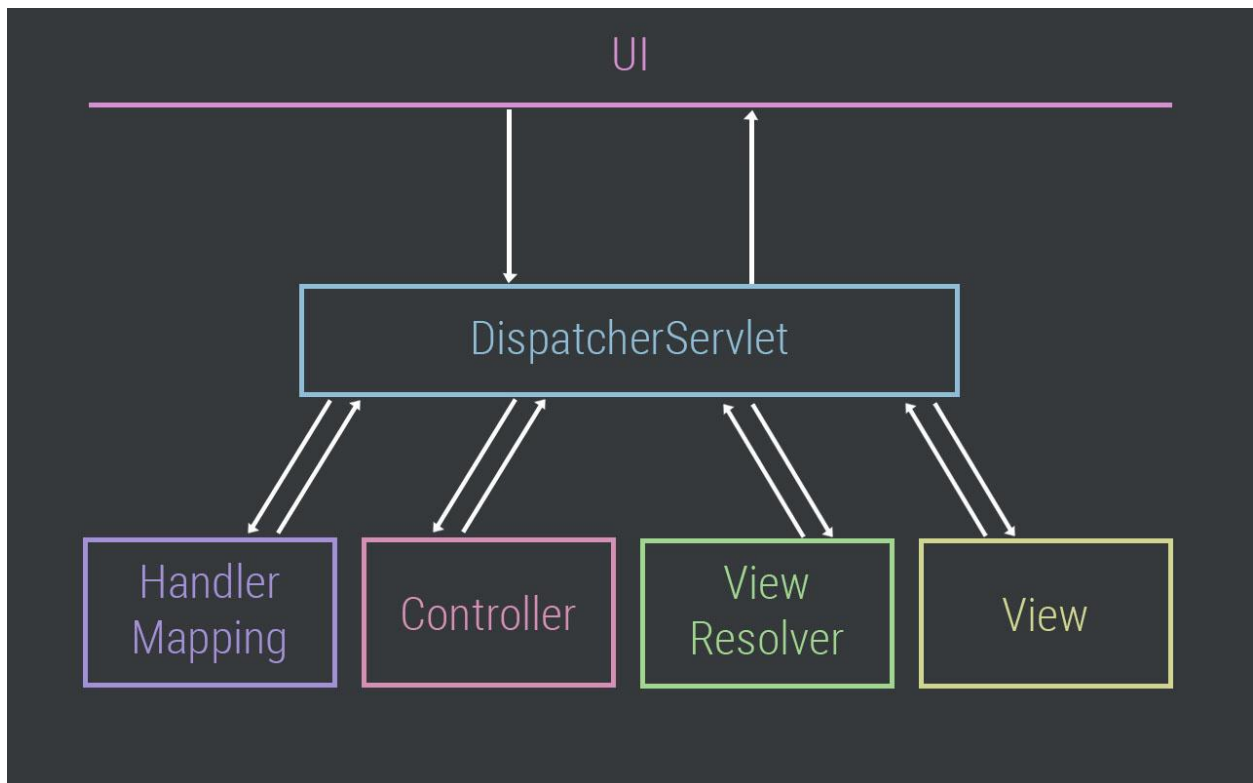


Рис.4.3.1.1 архітектуру паттерна MVC

1. **Model** (Модель) інкапсулює (об'єднує) дані програми, в цілому вони будуть складатися з POJO («Старих добрих Java-об'єктів», або бінів).
2. **View** (Отображення, Вид) відповідає за відображення даних Моделі, — зазвичай, генерує HTML, який ми бачимо в своєму браузері.
3. **Controller** (Контролер) обробляє запит користувача, створює відповідну Моделю і передає її для відображення в Вид.

Структура проекту

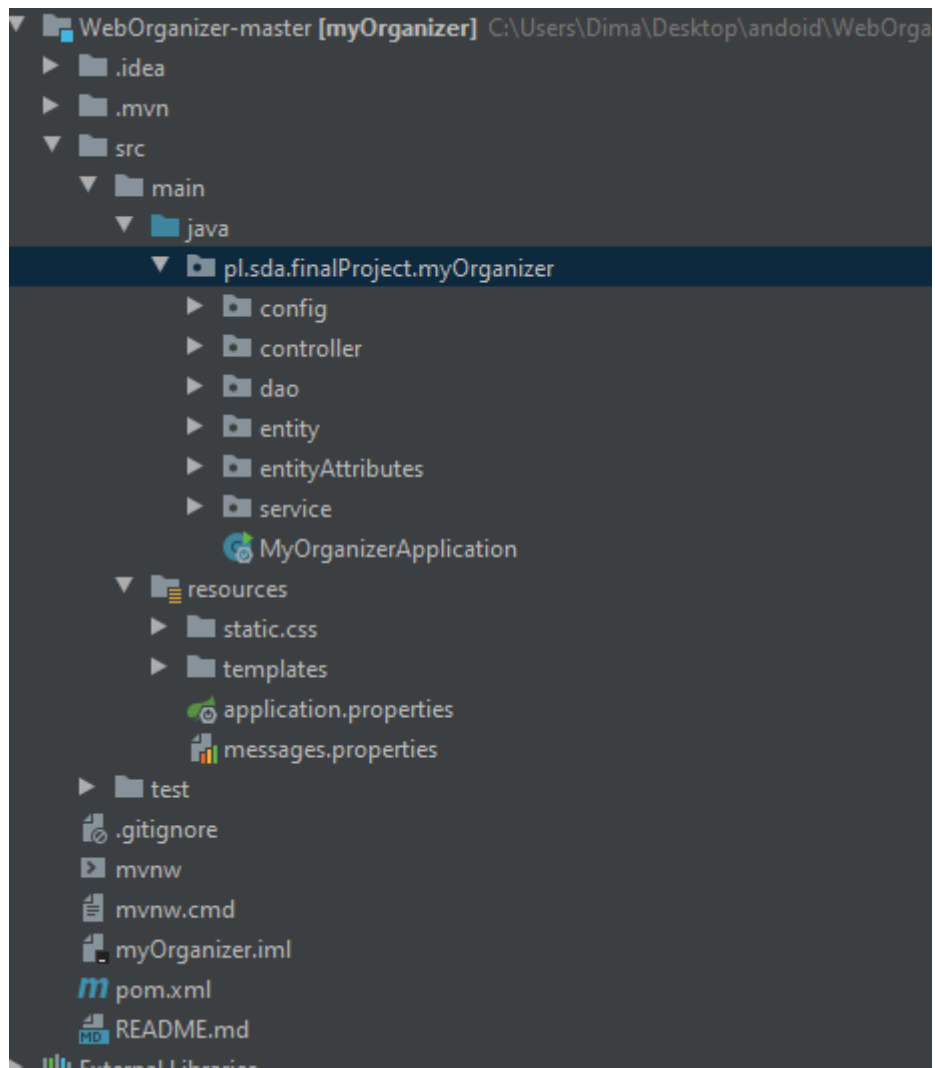


Рис.4.3.1.2 Структура проекту

Папка «dao», «entity» , «service» зберігає в собі класи , які потрібні для взаємодії з базою даних. Детальніше буде в наступних завданнях.

Entity – всі класи нашої моделі MVC.

Controller – всі класи контролерів MVC.

Templates – папка з html файлами (Вид MVC)

Файл application.properties - файл, де ми можемо вказати різні види властивостей (наприклад, параметри бази даних, чи параметри запуску серверу, чи параметри використання бібліотеки для авторизації та інше)

Maven автоматично завантажить бібліотеки залежностей Spring і помістить їх в локальний репозиторій Maven . Pom.xml – файл в якому, ми прописуємо потрібні бібліотеки.

Style.css – папка з файлами css для стилізації вашої веб-сторінки.

4.3.2. Створення програмних класів

Модель event:

```
package pl.sda.finalProject.myOrganizer.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.validator.constraints.NotEmpty;
import org.springframework.beans.factory.annotation.Autowired;
import pl.sda.finalProject.myOrganizer.entity.Event;
import pl.sda.finalProject.myOrganizer.entity.MyUser;
import pl.sda.finalProject.myOrganizer.service.EventService;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;

@Data
@NoArgsConstructor
@AllArgsConstructor

public class EventModel {
    private Long id;
    @NotEmpty
    private String name;
    private LocalDate creationDate;
    @NotEmpty
    private String stringEventDate;
    private LocalDate eventDate;
```

Продовження моделі event:

```
    private String stringEventTime;
    private LocalTime eventTime;
    @Min(0L)
    @Max(59L)
    private int minutesBefore;
    @Min(0L)
    @Max(23L)
```



```

private int hoursBefore;
@Min(0L)
@Max(366L)
private int daysBefore;
private MyUser user;

public EventModel(Event eventEntity){

    this.id = eventEntity.getId();

    this.name = eventEntity.getName();

    this.creationDate = eventEntity.getCreationDate();

    this.eventDate = eventEntity.getEventDate();

    this.eventTime = eventEntity.getEventTime();

    this.minutesBefore = eventEntity.getMinutesBefore();

    this.hoursBefore = eventEntity.getHoursBefore();

    this.daysBefore = eventEntity.getDaysBefore();

    this.user = eventEntity.getUser();

}
}

```

Таблиця 4.3.2.1. Опис анотації:

Анотація	Опис
@Controller	@Controller - (Слой представления) Аннотация для маркировки java класса, как класса контроллера. Данный класс представляет собой компонент, похожий на обычный сервлет (HttpServlet) (работающий с объектами HttpServletRequest и HttpServletResponse), но с расширенными возможностями от Spring Framework.

Продовження таблиці 4.3.2.1. Опис анотації:

@Autowired	Аннотация @Autowired обеспечивает контроль над тем, где и как автосвязывание должны быть
------------	------------------------------------------------------------------------------------------

	осуществлено. Мы можем использовать <code>@Autowired</code> как для методов, так и для конструкторов
<code>@GetMapping</code>	Аннотация <code>@GetMapping</code> говорит — метод <code>list()</code> должен быть вызван, когда кто-то вызывает метод GET на пути <code>/notes</code> . Имя пути, очевидно, берётся из параметра <code>@RequestMapping</code> на классе.
<code>@PostMapping</code>	Аннотация <code>@PostMapping</code> без спецификации пути, говорит нам, что метод <code>create()</code> обслуживает <i>POST</i> запросы по пути <code>/notes</code> .

Зразок коду контроллер event:

```
@Controller

public class EventController {

    @Autowired
    private IEventRepository eventRepository;
    @Autowired
    private IUserRepository userRepository;
    @Autowired
    private EventService eventService;
    @GetMapping("/organizer/events")
    public String showEventsPage(Model model, Principal principal) {
        EventModel newEvent = new EventModel();
        MyUser activeUser = userRepository.findOne(principal.getName());
        model.addAttribute("events", eventService.getCurrentEvents(activeUser));
        model.addAttribute("newEvent", newEvent);
        model.addAttribute("today", LocalDate.now());
        model.addAttribute("eventsToRemind",
eventService.getEventsToRemind(activeUser));

        return "events";
    }
}
```

Продовження зразка коду контролеру event:

```
@PostMapping(path = "organizer/events")

    public String addEvent(@Valid @ModelAttribute("newEvent") EventModel
newEvent, BindingResult bindingResult,
```

```

        Principal principal, Model model) {

    if (bindingResult.hasErrors()) {
        model.addAttribute("validErrors", true);
        model.addAttribute("errorMsg",
            "Adding the event failed, parameters name and date
required!");
        return "events";
    }

    MyUser activeUser = userRepository.findOne(principal.getName());
    eventService.parseEventDateAndTime(newEvent);

    Event entity = Event.builder()
        .creationDate(LocalDate.now())
        .user(activeUser)
        .name(newEvent.getName())
        .eventDate(newEvent.getEventDate())
        .eventTime(newEvent.getEventTime())
        .minutesBefore(newEvent.getMinutesBefore())
        .hoursBefore(newEvent.getHoursBefore())
        .daysBefore(newEvent.getDaysBefore())
        .build();
    eventRepository.save(entity);
    return "redirect:/organizer/events";
}

@GetMapping(path = "/organizer/events/edit/{id}")
public String showEditForm(@PathVariable("id") Long id, Model model) {
    Event editEvent = eventRepository.findOne(id);
    if (editEvent == null) {
        return "eventNotFound";
    }
}

```

4.4. Особливості розробки алгоритмів методів програмних класів

DAO (Data Access Object) - це шар об'єктів які забезпечують доступ до даних. Зазвичай для реалізації DAO використовується EntityManager і з його допомогою ми працюємо з нашою БД, але в нашому випадку це система не підійде, так як ми вивчаємо Spring Data нам потрібно використовувати її засоби інакше нема чого він нам.

Spring Data надає набір готових реалізацій для створення DAO але Spring вважали за краще цей шар називати не DAO, а Repository.

Приклад створення Repository:

```

package pl.sda.finalProject.myOrganizer.dao;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import pl.sda.finalProject.myOrganizer.entity.Event;

```

```

import pl.sda.finalProject.myOrganizer.entity.MyUser;

import java.util.List;

@Repository

public interface IEventRepository extends JpaRepository<Event, Long> {

    List<Event> findByUserOrderByEventDateAsc(MyUser user);

    List<Event> findAllByUser(MyUser user);

    void deleteAllByUser(MyUser user);
}

```

JpaRepository - це інтерфейс фреймворка Spring Data надає набір стандартних методів JPA для роботи з БД.

Service - це Java клас, який надає з себе основну (Бізнес-Логіку). В основному сервіс використовує готові DAO / Repositories або ж інші сервіси, для того щоб надати кінцеві дані для призначеного для користувача інтерфейсу.

Приклад створення Service:

```

package pl.sda.finalProject.myOrganizer.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import pl.sda.finalProject.myOrganizer.dao.IEventRepository;
import pl.sda.finalProject.myOrganizer.entity.Event;
import pl.sda.finalProject.myOrganizer.entity.MyUser;
import pl.sda.finalProject.myOrganizer.model.EventModel;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
import java.util.Iterator;
import java.util.List;
import java.util.stream.Collectors;
@Service

public class EventService {
    @Autowired
    private IEventRepository eventRepository;
    @Autowired
    private DateTimeFormatter dateFormatter;
    @Autowired
    private DateTimeFormatter timeFormatter;
    public void parseEventDateAndTime(EventModel eventModel) {
        // eventDate required

```

Продовження прикладу створення Service:

```

        eventModel.setEventDate(LocalDate.parse(eventModel.getStringEventDate(),
dateFormatter));

        // optional

```

```

        if (!eventModel.getStringEventTime().isEmpty()) {
            eventModel.setEventTime(LocalTime.parse(eventModel.getStringEventTime(),
timeFormatter));
        }
    }
    public String parseEventDate(LocalDate localDate) {
        return localDate.format(dateFormatter);
    }
    public String parseEventTime(LocalTime localTime) {
        return localTime.format(timeFormatter);
    }
    public List<Event> getCurrentEvents(MyUser activeUser) {
        List<Event> events =
eventRepository.findByUserOrderByEventDateAsc(activeUser);
        for (Iterator<Event> iterator = events.iterator(); iterator.hasNext(); )
        {
            Event event = iterator.next();
            if (event.getEventDate().isBefore(LocalDate.now())) {
                eventRepository.delete(event);
                iterator.remove();
            }
        }
        return events;
    }
    public List<Event> getEventsToRemind(MyUser activeUser) {
        return eventRepository.findAllByUser(activeUser).stream().filter(
            event -> isEventToRemind(event)
        ).collect(Collectors.toList());
    }
    private boolean isEventToRemind(Event event) {
        if (event.getEventTime() == null) {
            event.setEventTime(LocalTime.MIDNIGHT);
        }
        if (event.getDaysBefore() > 0 && event.getEventDate().
            minusDays(event.getDaysBefore()).isEqual(LocalDate.now())) {
            return true;
        }
        if (event.getHoursBefore() > 0 &&
LocalDateTime.now().isAfter(LocalDateTime.of(event.getEventDate(),
            event.getEventTime()).minusHours(event.getHoursBefore())) {
            return true;
        }
        if (event.getMinutesBefore() > 0 &&
LocalDateTime.now().isAfter(LocalDateTime.of(event.getEventDate(),
            event.getEventTime()).minusMinutes(event.getMinutesBefore())) {
            return true;
        } else return false;
    }
}

```

4.5. Використання спеціалізованих програмних бібліотек

За допомогою конфігураційного файлу `pom.xml` встановлюються бібліотеки. Все що потрібно це вказати пакети, які безпосередньо потрібні. Можна легко знайти пакети, використовуючи Google, і пошук "maven репозиторій".

Приклад файлу pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>pl.sda.finalProject</groupId>
    <artifactId>myOrganizer</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>myOrganizer</name>
    <description>Demo project for Spring Boot</description>
    // Все приложения Spring Boot конфигурируются от spring-boot-starter-parent
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.10.RELEASE</version>
        <relativePath/>
    </parent>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
    /* Оскільки ми створюємо REST API, то необхідно в якості залежно використовувати spring-boot-
starter-web, яка неявно визначає всі інші залежності, такі як spring-core, spring-web, spring-webmvc,
servlet api, і бібліотеку jackson-databind */
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```

Продовження прикладу файлу pom.xml:

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
    //Підключення бібліотеки для роботи с БД postgresql
    <dependency>
```

```

        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
//Підключення бібліотеки Lombok
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
//Підключення бібліотеки Security
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>
//Підключення бібліотеки Hibernate
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-java8</artifactId>
        <version>5.1.0.Final</version>
    </dependency>
//Підключення бібліотеки Security
    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity4</artifactId>
        <version>3.0.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.196</version>
        <scope>test</scope>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>

```

4.6. Модульне тестування програмних класів

Опишіть процес реального модульного тестування, використовуючи проект тестових наборів, описаних в лабораторній роботі No 8.

Якщо при тестуванні використовуються спеціалізовані програми, тоді опишіть процес їх використання.

1. Теоретичні відомості

Специфікація функції SetEvent():

- 1) Date
- 2) Time
- 3) Event name
- 4) minutesBefore
- 5) hoursBefore
- 6) daysBefore

Тестові набори:

Таблица 2

Название функции	Номер теста	Описание вводных значений	Описание ожидаемого результата
SetEvent()	10	Time = 10:00 Name = «Собрание в школе» Date = 12/01/2021	Событие успешно создано
SetEvent()	11	Name = Собрание Text = «Собрание в школе» Date = 1/12/2020	Ошибка: «Минимальное значение должно быть *сегодняшняя дата*»


```

@PostMapping(path = "organizer/events")
public String addEvent(@Valid @ModelAttribute("newEvent") EventModel newEvent, BindingResult bindingResult,
                      Principal principal, Model model) {

    if (bindingResult.hasErrors()) {
        model.addAttribute("validErrors", true);
        model.addAttribute("errorMsg",
            "Adding the event failed, parameters name and date required!");
        return "events";
    }

    MyUser activeUser = userRepository.findOne(principal.getName());
    eventService.parseEventDateAndTime(newEvent);

    Event entity = Event.builder()
        .creationDate(LocalDate.now())
        .user(activeUser)
        .name(newEvent.getName())
        .eventDate(newEvent.getEventDate())
        .eventTime(newEvent.getEventTime())
        .minutesBefore(newEvent.getMinutesBefore())
        .hoursBefore(newEvent.getHoursBefore())
        .daysBefore(newEvent.getDaysBefore())
        .build();

    eventRepository.save(entity);

    return "redirect:/organizer/events";
}

```


Рис. – обработка Post запроса событий

Після конструювання програмного модуля було проведено модульне мануальное тестування відповідно до тестами №10-11.


Add Event

×

Pick date



Event time



Event name

Here you can add a reminder to your event

Minutes before...

Hours before...

Days before...

Save

Close

Рис. заповнення полів

Процес виконання тесту №10, завершився отриманням наступних результатів:

#	Date	Time	Name
1	2021-01-12	10:00	Собрание в школе

Рис. виконання тесту №10

Процес виконання тесту №11, завершився отриманням наступних результатів:

The screenshot shows a web form titled "Add Event" with a close button (X) in the top right corner. The form contains the following elements:

- Pick date:** A date input field showing "01.12.2020" with a calendar icon on the right. Below the field is a yellow error message box with an exclamation mark icon: "Минимальное значение должно быть 07.12.2020."
- Event name:** A text input field containing the text "Собрание в школе".
- Reminder section:** A blue button labeled "Here you can add a reminder to your event". Below it are three input fields for reminders: "Minutes before..." with value "0", "Hours before..." with value "0", and "Days before..." with value "0".
- Buttons:** At the bottom right, there are two buttons: "Save" (blue) and "Close" (grey).

Рис. виконання тесту №11

2. Теоретичні відомості

Спецификация функции SetNote():

1) Name

2) Description

Тестові набори:

Таблица 3

SetNote()	12	Name = «Книги» Text = «Необходимые книги для прочтения...»	Заметка успешно сохранена
-----------	----	------------------------------------------------------------------	------------------------------

```
@PostMapping("/organizer/notes")
public String addNote(@Valid @ModelAttribute("newNote") Note newNote, BindingResult bindingResult,
    Principal principal) {
    MyUser activeUser = userRepository.findOne(principal.getName());
    newNote.setUser(activeUser);
    newNote.setCreationDate(LocalDate.now());
    if (bindingResult.hasErrors()) {
        return "notes";
    }
    noteService.addNote(newNote);
    return "redirect:/organizer/notes";
}
```

Рис. обработка Post запроса заметок

Після конструювання програмного модуля було проведено модульне мануальное тестування відповідно до тесту №12.

Заповнення полів реєстрації с угодою тест-кейсу №10:

The screenshot shows a web application interface for creating a note. On the left, under the heading "Note name:", there is a text input field containing the word "Книги" (Books) and a larger text area below it containing the text "Необходимые книги для прочтения" (Necessary books for reading). At the bottom of this section is a blue button labeled "Add Your Note". On the right, under the heading "Your notes", there is a table with the following structure:

#	Note name	Creation date	Open/Edit	Delete
---	-----------	---------------	-----------	--------

Рис. процесс заполнения полів заметки

Процес виконання тесту №12, завершився отриманням наступних результатів:

The screenshot shows the same application interface as before, but now with a note created. The "Note name" input field is empty, and the text area contains the placeholder "Enter some text...". The "Add Your Note" button is still present. The "Your notes" table now contains one row:

#	Note name	Creation date	Open/Edit	Delete
1	Книги	2020-12-07	<button>Open/Edit</button>	<button>Delete</button>

Рис. виконання тесту №12

5 Розгортання та валідація програмного продукту

5.1 Інструкція з встановлення програмного продукту

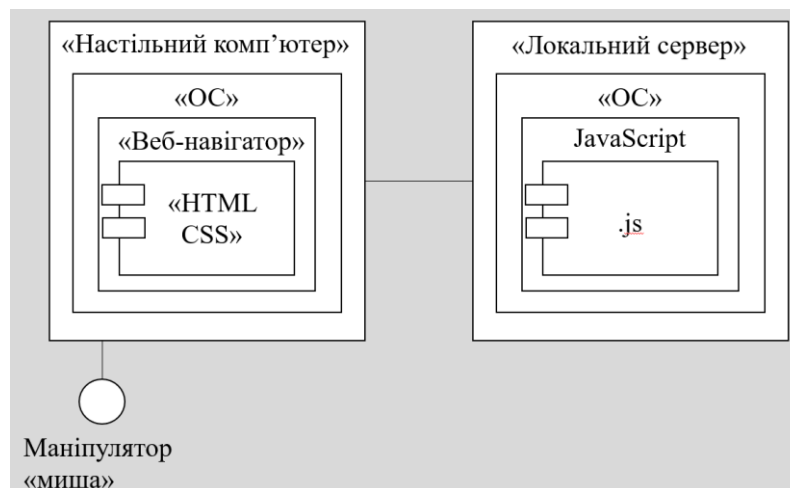


Рис.5.1.1. Приклад UML-діаграми розгортання ПП

5.2 Інструкція з використання програмного продукту

1) Реєстрація користувача

ПП надає можливість користувачу ролі «гість» вести параметри реєстрації (імя користувача, email та його пароль), як показано на (Рис.5.2.1).

Рис.5.2.1. Екранна форма реєстрації користувача

Для реєстрації користувач повин ввести значення, як показано на (Рис.5.2.2).

При цьому необхідно пам'ятати про обмеження значення паролю: не менше 5 символів, не більше 20 символів у логін та собака при заповненні електронної пошти.

A screenshot of a registration form on a dark background. It contains three input fields: 'Name' with the text 'DSD', 'Email' with the text 'cool1@gmail.com', and 'Password' with ten dots. Below the fields is a blue 'Register' button.

Name

DSD

Email

cool1@gmail.com


Password

.....

Register

Рис. 5.2.2. Екранна форма заповнення даних реєстрації користувача

Після етапу реєстрації користувачу необхідно буде увести свої дані у форму логіну (Рис.5.2.3).

A screenshot of a login form on a dark background. It contains two input fields: 'Email' with the text 'cool1@gmail.com' and 'Password' with ten dots. Below the fields is a blue 'Login' button.

Email

cool1@gmail.com

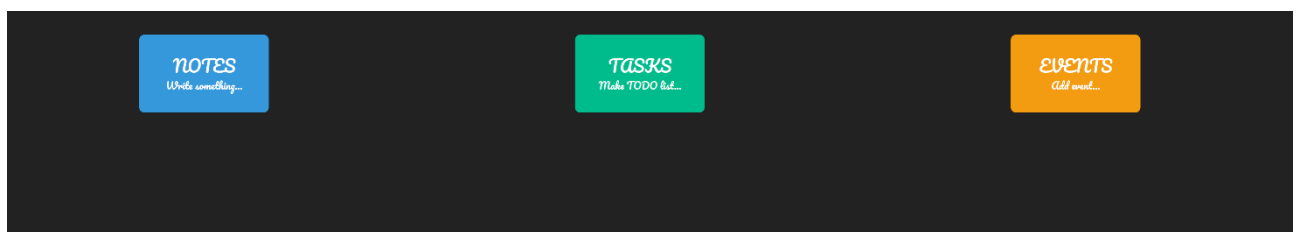
Password

.....

Login

Рис. Рис.5.2.3. Екранна форма заповнення даних логіну користувача

Далі користувач зможе вільно користуватися програмним продуктом(Рис.5.2.4).

A screenshot of a main menu on a dark background. It features three colored buttons: a blue 'NOTES' button with the subtitle 'Write something...', a green 'TASKS' button with the subtitle 'Make TODO list...', and an orange 'EVENTS' button with the subtitle 'Add event...'.

NOTES
Write something...

TASKS
Make TODO list...

EVENTS
Add event...

Рис.5.2.4. Екранної форма заповнення даних логіну користувача

Або додати та подивитись задачі, вікно інтерфейсу яких показано на (рис. 5.2.5)

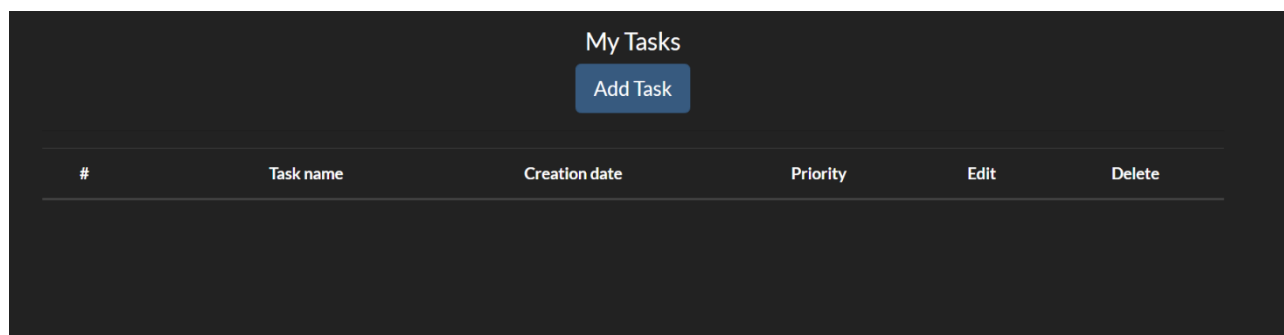


Рис.5.2.5. Інтерфейс задач

5.3 Результати валідації програмного продукту

Нашим завданням було допомогти користувачам структурувати свій день, а також не залишати поза голови плани та ідеї. Я вважаю, що з цією метою наша команда впоралася на усі сто.

1) Завдяки простому і зрозумілому інтерфейсу користувач зможе з легкістю додавати замітки / завдання / події.

2) Весь функціонал програмного продукту був протестований, тому з ним не виникне проблем.

Висновки

В результаті створення програмного продукту була досягнута наступна мета його споживача: «визначення мети з підпункту 1.2.2.2».

Доказом цього є наступні факти:

- 1) Завдяки простому і зрозумілому інтерфейсу користувач зможе з легкістю додавати замітки / завдання / події;
- 2) Весь функціонал програмного продукту був протестований, тому з ним не виникне проблем;

В процесі створення програмного продукту виникли такі труднощі (організаційні, проблеми відсутності досвіду, знань, потрібних в різних етапах):

- 1) Брак знань мови програмування;
- 2) Мала кількість досвіду в створенні програмних продуктів;
- 3) Незвичний досвід роботи в команді.

Через вищеописані непередбачені труднощі, а також через обмежений час на створення програмного продукту, залишилися нереалізованими такі прецеденти або їх окремі кроки роботи:

- 1) Плани зі створення розділу збереження паролів;

Зазначені недоробки планується реалізувати в майбутніх курсових роботах з урахуванням тем дисциплін наступних семестрів.