

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПЛАНУВАННЯ СПРАВ У СІМ'Ї
FAMILY ACTIVITIES PLANNING SOFTWARE

Виконав: студент(ка) 4 курсу, групи КНТ-138
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Павліченко О.А.

(прізвище та ініціали)

Керівник Голуб Т.В.

(прізвище та ініціали)

Рецензент Терещенко Е.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Павліченко Олени Андріївни

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення планування справ у сім'ї
Family Activities Planning Software

керівник проєкту (роботи) Голуб Тетяна Василівна, к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) червень 2022 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми.
3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Голуб Т.В., доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання «4» квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)


(підпис)

Павліченко О.А.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Голуб Т.В.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 80 с., 21 рис., 2 табл., 3 дод., 9 джерел.

СІМ'Я, ЗУСТРІЧ, СПИСОК СПРАВ, ГОЛОСУВАННЯ, ВЕБСИСТЕМА, БАЗА ДАНИХ, PYTHON.

Об'єктом роботи є процес планування справ у сім'ї. Предметом роботи є програмне забезпечення планування справ у сім'ї. Мета роботи – розробити програмне забезпечення планування справ у сім'ї для впорядкування спільного доступу до інформації в сім'ї.

Матеріали, методи та технічні засоби: мова програмування Python, система керування базами даних MySQL.

Результати. Слідуючи ідеї спільноти з доступом до інформації, доступ до якої ззовні закритий, було проведено аналіз проблеми, сформульовано вимоги до програми для забезпечення такої роботи, визначено структуру бази даних, визначено порядок роботи з програмою на основі схеми функціонування програми, реалізовано і описано реалізовану програму на рівні коду і на рівні користувацького інтерфейсу.

Висновки. Розроблена в результаті виконання роботи програма призначена для планування і обговорення справ у сім'ї, розглядаючи сім'ю як деяку спільноту, яка потребує впорядкованого, але обмеженого доступу до інформації про справи. Для цього програма забезпечує роботу з даними стосовно зустрічей, в яких бере участь вся сім'я або її члени, стосовно справ, які потрібно виконати, стосовно розповсюдження інформаційних повідомлень, обговорення різних питань, голосування за невирішеними питаннями.

Галузь використання. Ведення справ сім'ї, включаючи зустрічі, списки справ, обговорення тощо.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 80 pages, 21 figures, 2 tables, 3 appendixes, 9 sources.

FAMILY, MEETING, TO-DO LIST, VOTING, WEB SYSTEM, DATABASE, PYTHON.

The object of the work is the process of family planning. The subject of the work is family business planning software. The purpose of the work is to develop family planning software to organize the sharing of information in the family.

Materials, methods and technical means: Python programming language, MySQL database management system.

Results. Following the ideas of the community which works with information with forbidden external access, the problem was analyzed, the requirements for the software were formulated to ensure such a work, the database structure was determined, the procedure for working with the program was determined based on the program operation scheme. The software was presented and described at the code and user interface levels.

Conclusions. The resulting program is designed to plan and discuss family activities, viewing the family as a community which needs orderly but limited access to information. To do this, the program provides work with data on meetings attended by the whole family or its members, on matters to be performed, on the dissemination of information messages, discussion of various issues, voting on unresolved issues.

Field of use. Family management, including meetings, to-do lists, discussions and more.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	7
Вступ	8
1 Аналіз предметної області	9
1.1 Огляд програмних аналогів	9
1.2 Аналіз аналогів і розробки та визначення особливостей розробки	11
1.3 Висновки за розділом 1	12
2 Розробка архітектури програми	13
2.1 Вибір мови розробки програми	13
2.2 Функціональні вимоги до програмного забезпечення	13
2.3 Проектування бази даних	15
2.4 Висновки за розділом 2	28
3 Розробка програми	29
3.1 Послідовність роботи	29
3.2 Опис реалізованих класів програми	30
3.3 Висновки за розділом 3	38
4 Експлуатація та тестування програми	39
4.1 Призначення програми	39
4.2 Умови виконання програми	39
4.3 Тестування і опис принципів експлуатації програми	39
4.4 Висновки за розділом 4	43
Висновки	44
Перелік джерел посилання	45
Додаток А Технічне завдання	46
Додаток Б Текст програми	50
Додаток В Слайди презентації	74

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ПЗ – програмне забезпечення.

ВСТУП

Людина – істота соціальна, це загальновідомий факт. Першою ланкою взаємодії людини з соціумом є сім'я. Багато людей віддають велику частину своєї енергії своїй сім'ї. Чим більшою є сім'я, тим складніше керувати її справами. Виникають як окремі справи, які потрібно виконати, так і купа обговорень, а до того ж спільні плани. Потрібно організувати все таким чином, щоб всі члени сім'ї прибули в потрібне місце в потрібний час. З часом таких питань накопичується все більше, а кількість програм, які використовуються для їх вирішення, все збільшується. Це може бути і програмне забезпечення (ПЗ), що створює списки справ, при чому це можуть бути специфічні окремі списки справ, як то купівля продуктів у магазині, так і месенджери, календарі, і інші більш спеціалізовані інструменти.

Загубитися в усьому різноманітті програмних інструментів достатньо просто. Окрім того у випадку сім'ї це вже не персональне використання, потрібно, щоб кожен член сім'ї користувався кожним засобом. Звичайно, що це ускладнює вирішення задачі, і є незручним. Тому важливим є надати те ПЗ, яке зможе об'єднувати базові інструменти, що використовуються для роботи з інформацією, яка має бути доступна всім членам сім'ї.

У такому випадку сім'я фактично може розглядатися як спільнота, кожен член якої отримує доступ до інформації, до якої немає доступу іншим користувачам програми. Члени сім'ї потребують накопичення даних про зустрічі, списки справ, обговорення деяких питань, перегляду інформаційних повідомлень, щоб бути впевненими в тому, що кожен з членів сім'ї має доступ до такої інформації і за допомогою програми може не пропустити спільну подію, виконати спільні дії тощо. У результаті мета роботи була визначена наступним чином: розробити ПЗ планування справ у сім'ї для впорядкування спільного доступу до інформації в сім'ї.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд програмних аналогів

Cozi Family Organizer (рис. 1.1) [1] – це програма, яка може використовуватися для планування справ у сім'ї, для чого вона має достатньо широкий набір функцій, програма доступна в Google Play і App Store, окрім того останнім часом з'явилась версія для веббраузера, програма дозволяє користувачам додавати, редагувати та видаляти списки справ і подій [2]-[3].

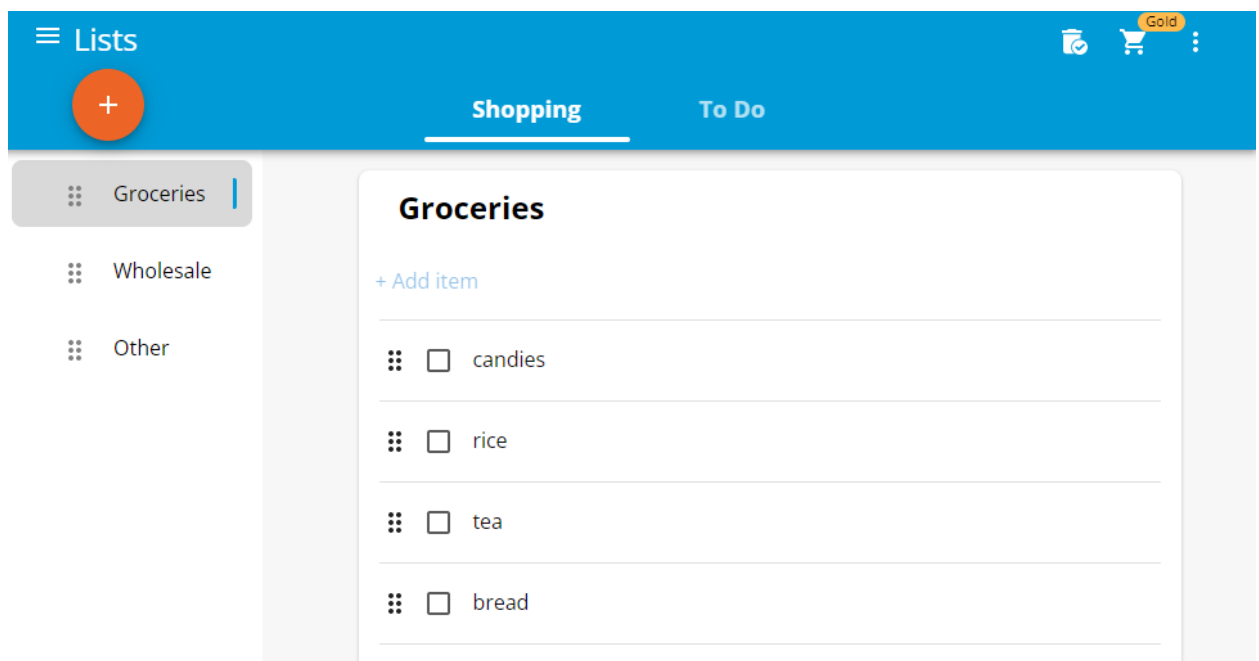


Рисунок 1.1 – Програма Cozi Family Organizer

Програма складається з кількох основних вкладок, що стосуються органайзера: розділ «Покупки» дає змогу створити кілька списків покупок, вкладка «Справи» відповідає за управління списками завдань (як загальними, актуальними для всіх, так і особистими), окрім того в розділі «Календар» можна побачити загальні розклади та особисті події кожного члена сім'ї на тій самій сторінці (рис. 1.2) [1], у календарі також є можливість надсилати нагадування про важливі події, вкладка «Рецепти» допомагає додати

інгредієнти улюблених рецептів до списку покупок, також можна додати страви з рецептів до календаря подій [2]-[4].

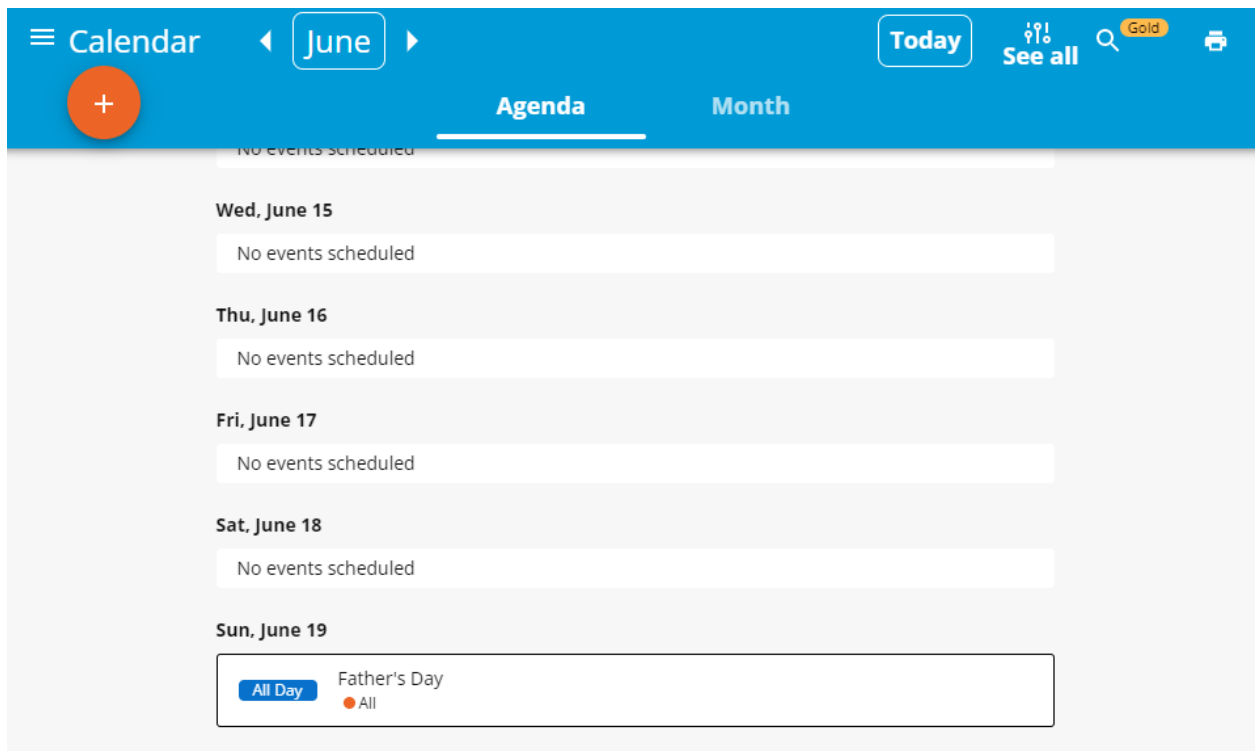


Рисунок 1.2 – Календар в програмі Cozi Family Organizer

Hub Family Organizer (рис. 1.3) [5] можна завантажити з популярних платформ застосунків обох мобільних операційних систем, представляє собою мінімалістичний застосунок, основною направленістю якого є події календаря та списки (завдань і покупок), усі події, записи та списки відображаються в одному місці, Hub Family Organizer також дозволяє писати повідомлення та публікувати фотографії, що є її особливістю порівняно з попередньою програмою, фактично ПЗ можна використовувати для будь-якої невеликої команди, при чому можна додати декілька таких груп, безкоштовна версія не має реклами, а платна підписка дає змогу прикріплювати необмежену кількість фотографій у чаті, коментувати, додавати нотатки, призначати теги та синхронізувати інформацію з необмеженою кількістю календарів [2], [6].



Рисунок 1.3 – Програма Hub Family Organizer [5]

1.2 Аналіз аналогів і розробки та визначення особливостей розробки

У процесі порівняння виділених аналогів з програмою, що розробляється, було виділено набір основних функцій, які надає кожна з них.

Виявлено, що Cozi має певну кулінарну направленість у своїх основних функціях. Дещо з цього набору функцій випадає список подій, проте наявність додатково до базових функцій, кожна з яких має застосування в приготуванні, підготовці до приготування, ще додатково рецептів вказує як раз на таку логіку.

У свою чергу Hub Family Organizer має особливості, які стосуються можливості комунікації між собою членів сім'ї, тобто надається можливість надсилення повідомлення.

Порівнюючи з цими аналогами, до особливості програми, яка розробляється, відноситься можливість створення і проведення голосувань,

коли члени сім'ї можуть швидко виявити свою позицію щодо того чи іншого питання, яке тут же стає основою зустрічей, справ зі списків тощо. Окрім того програма не має платних можливостей та об'єднує в собі фактично позитивні риси обох аналогів, які стосуються саме можливостей роботи всієї сім'ї.

Результати порівняння представлені в табл. 1.1.

Таблиця 1.1 – Порівняння розробки з програмними аналогами

Критерій	Cozi Family Organizer	Hub Family Organizer	Розробка
Планування зустрічей	Так	Так	Так
Створення списків справ	Так	Так	Так
Відправлення повідомлень	Ні	Так	Так
Голосування	Ні	Ні	Так
Наявність платних можливостей	Так	Так	Ні

1.3 Висновки за розділом 1

Виконано огляд програмних аналогів, включаючи Cozi Family Organizer, Hub Family Organizer, визначено їх основні можливості і особливості. Порівняно ці аналоги з розробкою, що виконується в роботі, виділено її особливості.

Відповідні завдання роботи, які було визначено для досягнення мети роботи:

- дослідження проблеми розроблення ПЗ планування справ у сім'ї;
- визначення вимог до розробки;
- проєктування ПЗ планування справ у сім'ї;
- програмна реалізація ПЗ планування справ у сім'ї.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір мови розробки програми

Для реалізації програми було обрано мову програмування Python [7]-[9] за результатами її порівняння з JavaScript (табл. 2.1), що також є мовою, що активно застосовується для вебпрограмування. Python порівняно з JavaScript (розглядалась мова одразу в зв'язці з Node.js) має більш стандартну модель наслідування, дозволяє створювати меншу кількість рядків коду, реалізуючи ту саму функціональність, та має більше готових для підключення модулів, що розширює можливості розробки.

Таблиця 2.1 – Вибір мови розробки програми

Критерій	JavaScript	Python
Розробка серверної частини вебзастосунку	Node.js	Так, також доступні Django, Flask
Модель наслідування	На основі прототипів	На основі класів
Необхідність розробки меншої кількості коду	Ні	Так
Стабільність	Нижче	Вище
Кількість модулів для підключення в програмах	Менша	Більша

2.2 Функціональні вимоги до програмного забезпечення

Функції, які були визначені для реалізації в програмі, охоплюють:

– додавання нової спільноти сім'ї;

- керування членами спільноти сім'ї (додавання і вилучення);
- авторизація і реєстрація користувачів;
- перегляд доступних спільнот сім'ї;
- додавання нового списку справ для спільноти сім'ї;
- змінювання списку справ для спільноти сім'ї;
- внесення справ до списку справ;
- перегляд списку справ;
- перегляд всіх списків справ сім'ї;
- збереження списку справ;
- перегляд збережених списків справ;
- додавання нової зустрічі в спільноту сім'ї;
- редагування існуючої зустрічі;
- внесення членів сім'ї для участі в зустрічі;
- перегляд всіх запланованих зустрічей сім'ї;
- перегляд інформації про зустріч;
- оцінка зустрічі;
- додавання нового інформаційного повідомлення;
- змінювання існуючого інформаційного повідомлення;
- перегляд всіх інформаційних повідомлень для членів сім'ї;
- уподобання інформаційного повідомлення;
- додавання нового повідомлення до загального обговорення;
- уподобання повідомлення з загального обговорення;
- перегляд всіх повідомлень у обговоренні;
- коментування зустрічі;
- коментування списку справ;
- коментування інформаційного повідомлення;
- додавання нового голосування;
- змінювання голосування;
- внесення варіантів вибору в голосуванні;
- голосування;

- перегляд членів сім'ї, які переглянули список справ;
- перегляд членів сім'ї, які переглянули інформацію про зустріч;
- перегляд членів сім'ї, які переглянули інформаційне повідомлення повністю;
- перегляд членів сім'ї, які переглянули голосування повністю;
- перегляд майбутніх зустрічей сім'ї;
- перегляд списків незроблених справ;
- перегляд зустрічей, в яких бере участь користувач.

Робота з системою у вигляді прецедентів користувача представлена на діаграмі прецедентів на рис. 2.1.

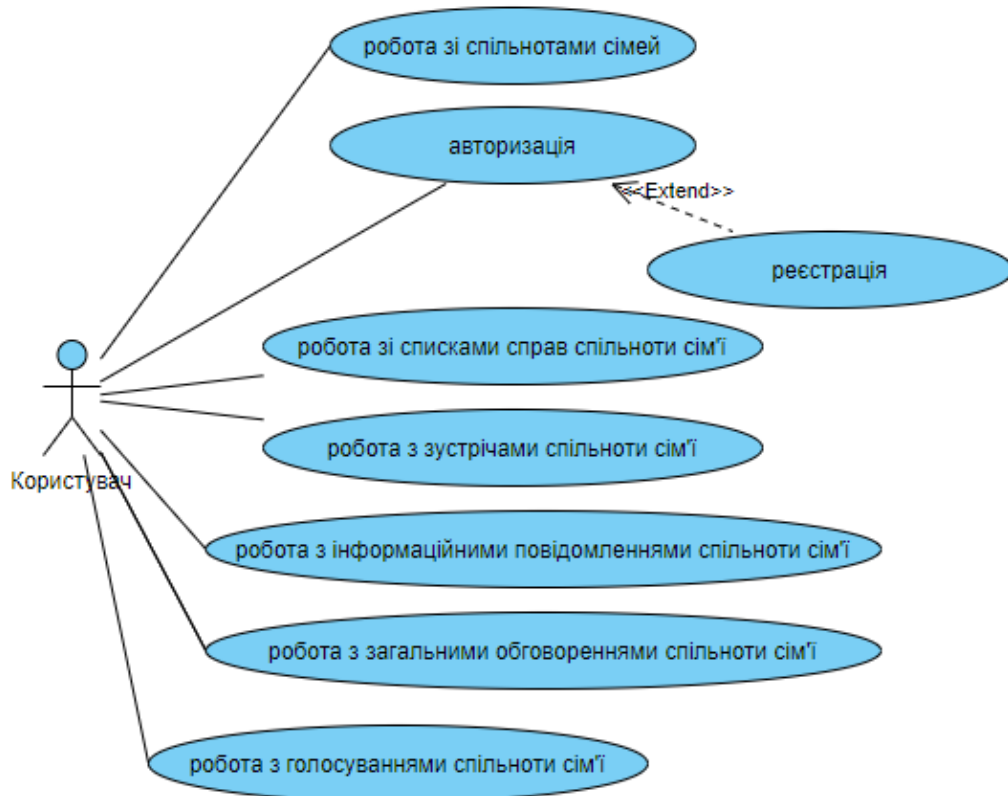


Рисунок 2.1 – Діаграма прецедентів

2.3 Проектування бази даних

Для реалізації бази даних (БД) і підтримки її роботи було використано систему керування БД MySQL. Цей вибір обґрунтовується тим, що ця

система є безкоштовною і достатньо надійною для даного рішення.

Оскільки з програмою можуть працювати різні сім'ї, а кожна сім'я – це певна спільнота, то однією з сутностей предметної області є сім'я. В БД вона представлена таблицею Family, яка включає наступні поля:

- id – ідентифікатор сім'ї;
- name – назва сім'ї;
- member – член сім'ї, який створив цю спільноту.

Поле family є зовнішнім ключем з таблиці Family, з якою зв'язок належить до типу один до багатьох, оскільки один член сім'ї може створити декілька спільнот, але кожна спільнота має одного власника.

Самі користувачі програми є членами сім'ї. Кожен член сім'ї зберігається в таблиці Member, яка включає наступні поля:

- id – ідентифікатор члена сім'ї;
- username – логін, що використовується членом сім'ї;
- name – ім'я члена сім'ї в програмі;
- password – пароль члена сім'ї.

У програмі передбачено, що кожен користувач може брати участь у різних сім'ях. Це продиктовано тим, що людина може отримувати інформацію від сім'ї, з якою проживає, і від сім'ї батьків, родичів. Необхідність цього продиктована тим, щоб не вимагати додавати до всіх учасників сім'ї всіх родичів, адже тоді кожен родич буде бачити абсолютно всю інформацію, потрібну більш вузькому колу. Тому сім'я розглядається фактично як певна спільнота, яка використовує інформацію, до якої доступ іншим користувачам не просто обмежений, а заборонений.

Відповідно через цю логіку зв'язок між таблицями Family і Member має тип багато до багатьох. Тому таблиця FamilyMember включає наступні поля:

- id – ідентифікатор запису про члена сім'ї;
- family – сім'я, членом якої є цей користувач;

– member – запис члена сім'ї, який використовується для роботи в програмі.

Поле family є зовнішнім ключем з таблиці Family, з якою зв'язок належить до типу один до багатьох. Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох.

Одним з важливих елементів роботи в програмі для користувача є список справ. Цей список справ може представляти список продуктів, які потрібно придбати, або список страв, які необхідно приготувати. Узагальнено він названий список справ. Таблиця Todolist включає наступні поля:

- id – ідентифікатор списку;
- title – назва списку справ;
- todotext – текст для пояснення списку справ;
- created – час створення списку справ;
- seencount – кількість переглядів списку справ членами сім'ї;
- member – автор цього списку справ;
- family – сім'я, в якій цей список справ створено.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: один член сім'ї може створювати різні списку справ, а один список справ може мати тільки одного автора, який є членом сім'ї.

Поле family є зовнішнім ключем з таблиці Family, з якою зв'язок належить до типу один до багатьох: одна сім'я може мати багато списків справ, але кожен список справ стосується тільки однієї сім'ї.

Однак список справ в БД не існує сам по собі, адже окрім узагальненої інформації, фактично його опису, до його складу входять ще справи. Відповідні дані зберігаються в таблиці TodolistItem, яка включає наступні поля:

- id – ідентифікатор складової списку справ;
- itemtext – сама справа (її текстовий опис);

- done – позначає виконання цієї справи (придбання продукту, приготування страви);
- done_date – час, коли було позначено, що справу виконано;
- member – член сім'ї, який позначив, що справу було виконано;
- todolist – список справ, складовою якого є цей запис.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: один член сім'ї може відзначати виконання багатьох різних справ, а одна конкретна справа з конкретного списку справ може бути виконана тільки одним членом сім'ї, адже саме для цього таке виконання і позначається.

Поле todolist є зовнішнім ключем з таблиці Todolist, з якою зв'язок належить до типу один до багатьох: один список справ може мати багато справ, а одна справа відноситься до конкретного списку.

Для того щоб розуміти, чи всі члени сім'ї переглянули створений список справ, потрібно зберігати інформацію про перегляди різних членів сім'ї цього списку. Відповідно для цього таблиця TodolistSeen включає наступні поля:

- id – ідентифікатор перегляду списку справ членом сім'ї;
- todolist – список справ, який переглянуто;
- member – член сім'ї, який цей список переглянув.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: один член сім'ї може переглядати різні списки справ, а кожен перегляд конкретного списку справ має належати тільки одному члену сім'ї.

Поле todolist є зовнішнім ключем з таблиці Todolist, з якою зв'язок належить до типу один до багатьох: один перегляд стосується тільки одного списку справ, а один список справ може переглядатися різними членами сім'ї.

Списки справ можна зберігати для того, щоб мати їх для швидкого доступу зокрема в ситуаціях, коли сім'я велика і справ багато. Відповідно для цього таблиця TodolistActive включає наступні поля:

- id – ідентифікатор запису про збереження списку справ;
- todolist – список справ, який було збережено;
- member – член сім'ї, який зберіг цей список справ;
- state – поле, яке визначає, чи зберігається ще цей список справ, чи ні.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: один член сім'ї може зберегти багато різних списків справ, а кожне збереження може бути виконано тільки одним конкретним членом сім'ї.

Поле todolist є зовнішнім ключем з таблиці Todolist, з якою зв'язок належить до типу один до багатьох: один список справ може бути збережений різними членами сім'ї, а одне збереження стосується тільки одного списку справ.

Зустрічі є однією з ключових сутностей, виділених в предметній області. Вони фактично необхідні для того, щоб кожен з членів сім'ї мав через програму доступ до всіх спільних зустрічей, в яких він бере участь. Відповідно створена для цього була таблиця Meetup, яка включає наступні поля:

- id – ідентифікатор зустрічі;
- title – назва зустрічі;
- address – адреса за, якою має відбутися зустріч;
- start_time – час початку зустрічі;
- end_time – час завершення зустрічі;
- phone – телефонний номер для отримання інформації про зустріч;
- created – час створення зустрічі в програмі;
- seen_count – кількість переглядів інформації про зустріч членами сім'ї;

- score – середня оцінка, виставлена зустрічі членами сім'ї;
- member – член сім'ї, котрим було внесено інформацію про зустріч в програму;
- family – сім'я, для якої цю зустріч заплановано.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: один член сім'ї може внести в програму багато зустрічей, але кожна зустріч може бути визначена тільки один тільки одним членом сім'ї.

Поле family є зовнішнім ключем з таблиці Family, з якою зв'язок належить до типу один до багатьох: одна сім'я може мати багато запланованих зустрічей, але кожна зустріч стосується тільки однієї сім'ї.

Однак сам по собі запис про зустріч є лише інформуванням кожного члена сім'ї, до того ж не кожен член сім'ї обов'язково має брати участь в зустрічі. Наприклад, в зустрічі можуть брати участь тільки батьки, а діти – ні. До того ж бажано, щоб відповідний учасник зустрічі підтвердив своє бажання взяти участь в зустрічі, щоб краще розуміти це. Відповідно для цього створено таблицю MeetupMember, яка включає наступні поля:

- id – ідентифікатор запису;
- meetup – зустріч, якої стосується запис;
- member – член сім'ї, що повинен взяти участь в зустрічі;
- done – підтвердження членом сім'ї своєї участі в зустрічі.

Таблиця MeetupMember реалізує зв'язок багато до багатьох між таблицями Meetup і Member, адже кожен член сім'ї може брати участь в різних зустрічах, а у кожній зустрічі можуть брати різні члени сім'ї.

Перегляди інформації про зустріч фіксуються окремо для розуміння того, чи всі члени сім'ї переглянули в певний момент інформацію про зустріч. Відповідно була створена таблиця MeetupSeen, яка включає наступні поля:

- id – ідентифікатор перегляду інформації про зустріч;
- meetup – зустріч, яку було переглянуто;

– member – член сім'ї, який переглянув інформацію про цю зустріч.

Поле meetup є зовнішнім ключем з таблиці Meetup, з якою зв'язок належить до типу один до багатьох: кожен перегляд стосується тільки однієї зустрічі, але зустріч може мати багато переглядів.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: кожен член сім'ї може переглянути інформацію про різні зустрічі, але кожен перегляд стосується тільки одного члена сім'ї.

Окрім того зустрічі можуть бути оцінені членами сім'ї. Для збереження оцінювань створена таблиця MeetupEstimation, яка включає наступні поля:

- id – ідентифікатор оцінювання зустрічі;
- member – член сім'ї, який виконав це оцінювання;
- meetup – зустріч, яку було оцінено;
- score – виставлена оцінка.

Поле meetup є зовнішнім ключем з таблиці Meetup, з якою зв'язок належить до типу один до багатьох: кожна оцінка виставляється одній конкретній зустрічі, а кожна зустріч може мати різні оцінки від різних членів сім'ї.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: кожен член сім'ї може виставляти різні оцінки, а кожна оцінка може бути виставлена тільки одним конкретним членом сім'ї.

Ще однією стороною взаємодії між членами сім'ї є так звана інформація, яка представляє загальні дані, які мають бути розповсюджені між членами сім'ї. Це фактично може бути новина. Відповідно ці дані представлені в таблиці Information, яка включає наступні поля:

- id – ідентифікатор інформативного повідомлення;
- title – назва інформативного повідомлення;
- info_text – текст інформативного повідомлення;

- created – час створення інформативного повідомлення;
- seen_count – кількість переглядів інформативного повідомлення;
- like_count – кількість уподобань інформативного повідомлення;
- member – член сім'ї, який створив інформативне повідомлення;
- family – сім'я, якої стосується це інформаційне повідомлення.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: кожен член сім'ї створює багато інформативних повідомлень, а кожне інформативне повідомлення може бути створено тільки одним членом сім'ї.

Поле family є зовнішнім ключем з таблиці Family, з якою зв'язок належить до типу один до багатьох: одна сім'я може мати багато інформаційних повідомлень, але кожне інформаційне повідомлення стосується тільки однієї сім'ї.

Для того щоб зафіксувати, чи всі члени сім'ї переглянули інформативне повідомлення, створено таблицю InformationSeen, яка включає наступні поля:

- id – ідентифікатор перегляду інформативного повідомлення;
- information – інформативне повідомлення, яке було переглянуто;
- member – член сім'ї, який переглянув інформативне повідомлення.

Поле information є зовнішнім ключем з таблиці Information, з якою зв'язок належить до типу один до багатьох: кожне інформативне повідомлення може бути багато разів переглянуто, але кожен перегляд стосується тільки одного інформативного повідомлення.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: кожен член сім'ї може переглянути багато інформативних повідомлень, але кожен перегляд стосується тільки одного інформативного повідомлення.

Для того щоб розуміти відношення членів сім'ї до інформації, що розповсюджується через програму, потрібно фіксувати уподобання членів сім'ї.

Уподобання інформативного повідомлення зберігаються за допомогою таблиці InformationLike, яка включає наступні поля:

- id – ідентифікатор уподобання інформативного повідомлення;
- information – інформативне повідомлення, уподобання якого виконано;
- member – член сім'ї, який уподобав інформативне повідомлення;
- created – час, коли було створене уподобання.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: кожне уподобання інформативного повідомлення може бути виконано тільки одним членом сім'ї, а кожен член сім'ї може уподобати різні інформативні повідомлення.

Для обговорення різних ситуацій членам сім'ї надається можливість відправляти повідомлення, які доступні всім членам сім'ї, це фактично сімейний форум. Для збереження цих даних створено таблицю MemberMessage, яка включає наступні поля:

- id – ідентифікатор повідомлення;
- member – член сім'ї, який відправив це повідомлення;
- message – текст повідомлення;
- created – час створення повідомлення;
- like_count – кількість уподобань даного повідомлення;
- family – сім'я, якої стосується це повідомлення.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: кожне повідомлення може бути створено тільки одним членом сім'ї, а кожен член сім'ї може створити різні повідомлення.

Поле family є зовнішнім ключем з таблиці Family, з якою зв'язок належить до типу один до багатьох: одна сім'я може мати багато повідомлень, але кожне повідомлення стосується тільки однієї сім'ї.

Уподобання повідомлень фіксується в БД за допомогою таблиці MemberMessageLike, яка включає наступні поля:

- id – ідентифікатор уподобання повідомлення;
- member – член сім'ї, який уподобав повідомлення;
- membermessage – повідомлення, уподобання якого виконано.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: кожне уподобання виконується одним членом сім'ї, але кожен член сім'ї може мати багато уподобань різних повідомлень.

Поле membermessage є зовнішнім ключем з таблиці MemberMessage, з якою зв'язок належить до типу один до багатьох: кожне уподобання стосується конкретного повідомлення, але кожне повідомлення може бути уподобано різними членами сім'ї.

Обговорення щодо зустрічі фіксується в БД за допомогою таблиці MeetupMessage, яка включає наступні поля:

- id – ідентифікатор повідомлення щодо зустрічі;
- member – член сім'ї, який створив це повідомлення;
- meetup – зустріч, якої стосується повідомлення;
- message – текст повідомлення;
- created – час, коли було створено повідомлення.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: кожне повідомлення про зустріч створюється одним членом сім'ї, але кожен член сім'ї може мати багато повідомлень стосовно зустрічей.

Поле meetup є зовнішнім ключем з таблиці Meetup, з якою зв'язок належить до типу один до багатьох: кожне повідомлення про зустріч стосується однієї конкретної зустрічі, а кожна зустріч може мати багато повідомлень про неї.

Повідомлення, які стосуються конкретного списку справ, тобто фактично обговорення списку справ, зберігаються за допомогою таблиці TodolistMessage, яка включає наступні поля:

- id – ідентифікатор повідомлення про список справ;

- todolist – список справ, відносно якого відправлено повідомлення;
- member – член сім'ї, який відправив це повідомлення;
- message – текст повідомлення;
- created – час, коли було створено повідомлення про список справ.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: кожен член сім'ї може створити багато повідомлень про списки справ, але кожне повідомлення створюється тільки одним членом сім'ї.

Поле todolist є зовнішнім ключем з таблиці Todolist, з якою зв'язок належить до типу один до багатьох: кожен список справ може мати багато повідомлень стосовно нього, але кожне повідомлення стосується тільки одного списку справ.

Обговорення загальних сімейних інформаційних повідомлень фіксується в БД за допомогою таблиці InformationMessage, яка включає наступні поля:

- id – ідентифікатор повідомлення;
- information – інформаційне повідомлення, про яке відправлено дане повідомлення;
- member – член сім'ї, який відправив це повідомлення;
- message – текст повідомлення;
- created – час, коли було створено повідомлення.

Поле information є зовнішнім ключем з таблиці Information, з якою зв'язок належить до типу один до багатьох: кожне повідомлення стосується конкретного інформаційного повідомлення, але кожне інформаційне повідомлення може мати багато повідомлень про нього.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: кожне повідомлення про інформаційне повідомлення створюється одним членом сім'ї, але кожен член сім'ї може мати багато повідомлень про інформаційні повідомлення.

Ще одним засобом комунікації між членами сім'ї є голосування. Збереження самих голосувань виконується таблицею Voting, яка включає наступні поля:

- id – ідентифікатор голосування;
- title – назва голосування;
- content – вміст голосування;
- created – дата створення голосування;
- end_time – час завершення голосування;
- member – член сім'ї, який створив це голосування;
- seen_count – кількість переглядів голосування;
- family – сім'я, якої стосується голосування.

Поле member є зовнішнім ключем з таблиці Member, з якою зв'язок належить до типу один до багатьох: кожне голосування має одного автора, який є членом сім'ї, але кожен член сім'ї може створити багато голосувань.

Поле family є зовнішнім ключем з таблиці Family, з якою зв'язок належить до типу один до багатьох: одна сім'я може мати багато голосувань, але кожне голосування стосується тільки однієї сім'ї.

Варіанти вибору, які надаються в голосуванні представлені таблицею VotingItem, яка включає наступні поля:

- id – ідентифікатор варіанту вибору в голосуванні;
- voting – голосування, до якого відноситься цей варіант вибору;
- item_text – текст варіанту вибору;
- vote_count – кількість голосів, яку було віддано за цей варіант у голосуванні.

Поле voting є зовнішнім ключем з таблиці Voting, з якою зв'язок належить до типу один до багатьох: кожен варіант стосується одного голосування, але кожне голосування може мати багато варіантів вибору.

Перегляд голосувань фіксується в таблиці БД VotingSeen, яка включає наступні поля:

- id – ідентифікатор перегляду голосування;

- voting – голосування, перегляд якого виконано;
- member – член сім’ї, який переглянув голосування.

Структура БД представлена на рис. 2.2.

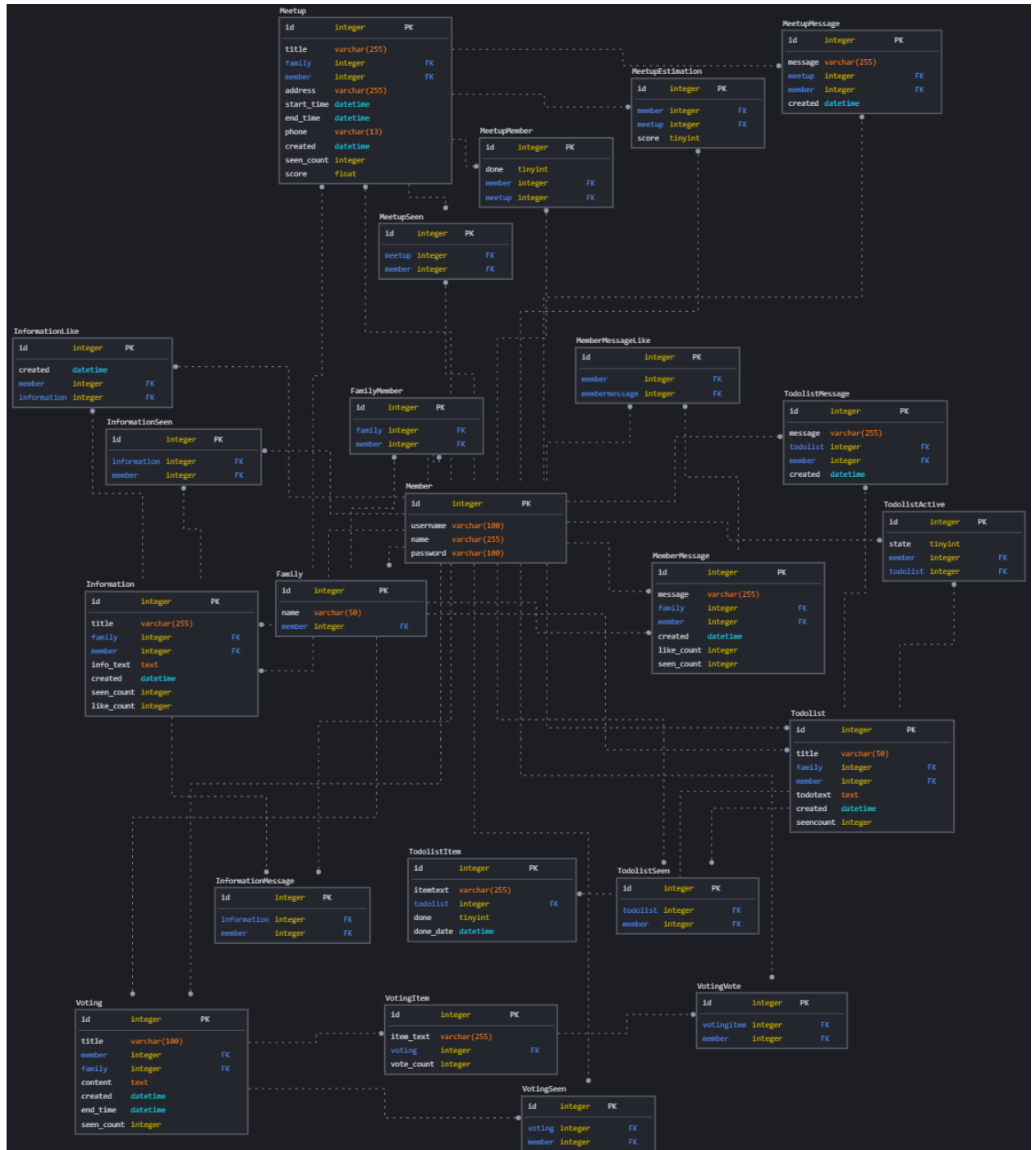


Рисунок 2.2 – Схема БД

Поле voting таблиці БД VotingSeen є зовнішнім ключем з таблиці Voting, з якою зв'язок належить до типу один до багатьох: кожне

голосування переглядається різними членами сім'ї, але кожен перегляд стосується одного голосування.

Поле `member` таблиці БД `VotingSeen` є зовнішнім ключем з таблиці `Member`, з якою зв'язок належить до типу один до багатьох: кожен член сім'ї може мати багато переглядів різних голосувань, але кожен конкретний перегляд стосується одного члена сім'ї.

Результати голосування членами сім'ї в різних голосуваннях фіксуються в таблиці БД `VotingVote`, яка включає наступні поля:

- `id` – ідентифікатор вибору, виконаного членом сім'ї;
- `votingitem` – варіант вибору в голосуванні, який зробив член сім'ї;
- `member` – член сім'ї, який виконав голосування.

Поле `votingitem` є зовнішнім ключем з таблиці `VotingItem`, з якою зв'язок належить до типу один до багатьох: кожен результат голосування виражається в одному обраному варіанті голосування, але за кожен варіант голосування може бути віддано багато голосів.

Поле `member` є зовнішнім ключем з таблиці `Member`, з якою зв'язок належить до типу один до багатьох: кожен член сім'ї може мати багато обраних варіантів у різних голосуваннях, але кожен результат голосування стосується тільки одного конкретного члена сім'ї.

2.4 Висновки за розділом 2

Під час проєктування ПЗ виконано вибір мови програмування для реалізації програми на користь мови Python, що було розширено для реалізації програми за допомогою фреймворку веброзробки Django, виконано вибір функцій, які має забезпечувати програма, створено діаграму прецедентів, розроблено структуру БД, обрано систему керування БД MySQL для підтримки її роботи, описано створені таблиці та зв'язки між ними.

3 РОЗРОБКА ПРОГРАМИ

3.1 Послідовність роботи

Порядок функціонування програми представлено у вигляді схеми функціонування (рис. 3.1).

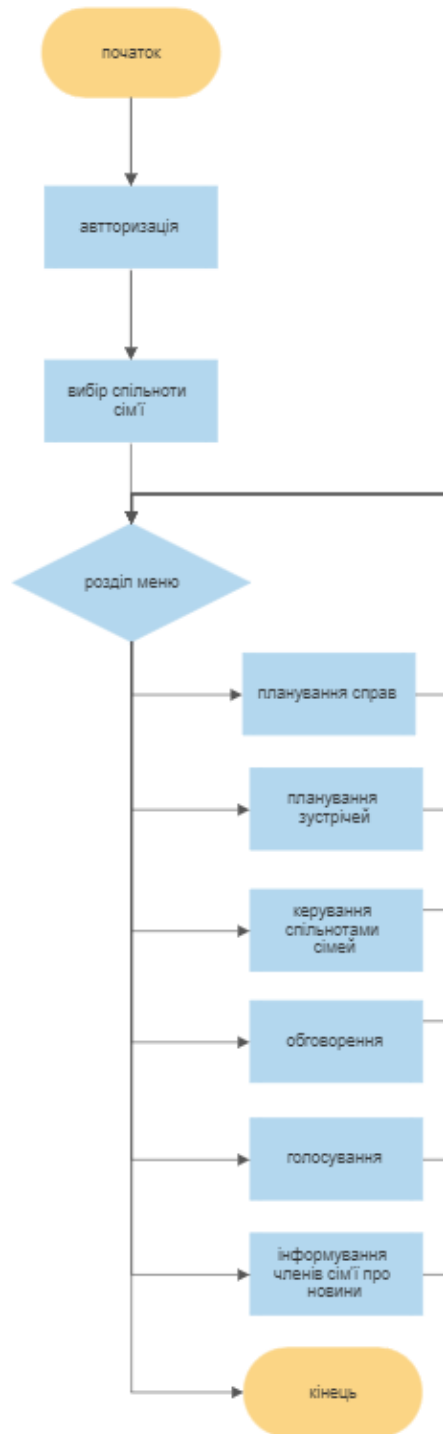


Рисунок 3.1 – Схема функціонування програми

Робота програми розпочинається з авторизації користувача. Саме після цього його можна ідентифікувати як учасника однієї зі спільнот. Тому після цього він повинен обрати одну з доступних йому спільнот як базову. Після цього йому стане доступним меню, з якого він зможе обрати будь-який потрібний йому розділ взаємодії.

3.2 Опис реалізованих класів програми

Звичайно, що базові функціональні можливості програми належать до процесів, пов'язаних зі створенням певної спільноти, яка розглядається як сім'я і в яку її власник може додавати учасників. Для підтримки цієї діяльності першочергово створено клас `FamilyAddView`, який ґрунтуючись на класі `CreateView` вебфреймворку Django, виконує підтримку роботи над додаванням нової спільноти сім'ї до програми, назва якої встановлюється через поле форми, а власник задається на основі даних авторизованого користувача.

Для забезпечення редагування відповідних даних спільноти сім'ї створено клас `FamilyEditView` на основі класу `UpdateView`. Окрім редагування значення назви спільноти цей клас також реалізує на основі підтримки роботи з формою отримання з неї через POST-запит списку членів сім'ї в параметрі `members`, виділення окремо всіх цих членів сім'ї і внесення їх до списку учасників цієї спільноти через поле `members` моделі `Family`, яке визначає зв'язок багато-до-багатьох з моделлю `Member`.

Підтримка виведення списку спільнот сімей, до яких має доступ авторизований член сім'ї, щоб він міг обрати ту спільноту, за якою в подальшому будуть відображатися дані на різних вебсторінках застосунку, реалізується через клас `FamiliesListView`. Сам список передається в шаблон вебсторінки `families.html`.

Для підтримки додавання нового списку справ створено клас `TodolistAddView`, який ґрунтуючись на класі `CreateView`, виконує заповнення

назви списку, опису цього списку через поля форми напряму та програмно задає дату створення списку як поточну, визначає автором списку авторизованого користувача, розбирає перелік складових цього списку, переданий через параметр `items` POST-запиту та визначаючи доступ до цього списку тільки для тієї спільноти сім'ї, номер якої було передано через параметр методу, тобто номер, який було обрано самим користувачем на початку роботи з застосунком або змінено після і який постійно зберігається на клієнтській стороні.

Особливою функцією в даному випадку є перегляд списку справ. Адже перегляд повинен не тільки відобразити вміст списку, але і зафіксувати перший перегляд кожного користувача, щоб потім можна було переглянути, хто переглянув потрібну йому інформацію, а хто – ні. Реалізацію цього процесу виконано на основі класу `TodolistView`, який реалізовано на основі класу `DetailView`. У методі `get_context_data` цього класу під час звернення до нього перевіряється, чи вже було зроблено запис про те, що цей авторизований користувач переглядав цей список, відповідно якщо ні, то виконується запис, лічильник переглядів цього списку справ оновлюється, а якщо так, то ніяких змін не реалізовується:

```
try:
    td = TodolistSeen(todolist__id = self.object.pk, member =
memberparam)
except TodolistSeen.DoesNotExist:
    td = TodolistSeen.objects.add(todolist__id = self.object.pk,
member = memberparam)
    td.save()
    self.object.seencount = self.object.seencount + 1
```

Для забезпечення редагування списку справ створено клас `TodolistEditView` на основі класу `UpdateView`. Окрім редагування значення назви цього списку справ та опису клас також реалізує на основі підтримки роботи з формою отримання з неї через POST-запит списку складових списку справ в параметрі `items`, виділення окремо всіх цих складових і внесення їх

до БД з паралельним розбиранням інформації про те, чи було виконано вже цю справу, та про назву цієї справи і відповідно внесенням цієї інформації разом. Якщо запис про таку справу в списку справ вже є в БД, то тільки перевіряється, чи необхідно оновити стан виконання справи, і якщо це потрібно, то у випадку виконання справи заноситься також час цієї фіксації.

Підтримка виведення всіх списків справ, які належать спільноті певної сім'ї, реалізується через клас `FamilyTodolistsListView`. Сам список передається в шаблон вебсторінки `familytodos.html`. Номер сім'ї, спільноту якої обрав користувач, передається через параметр `family`. Усі списки справ, які відповідають цій сім'ї, розташовуються за датою створення, починаючи з останнього створеного.

Підтримка виведення всіх списків справ, які належать спільноті певної сім'ї і які містять хоч одну справу, яку ще не було виконано, реалізується через клас `FamilyUndoneTodolistsListView`. Сам список передається в шаблон вебсторінки `familyundonetodos.html`. Номер сім'ї, спільноту якої обрав користувач, передається через параметр `family`. Усі списки справ, які містять незавершені справи і які відповідають цій сім'ї, розташовуються за датою створення, починаючи з останнього створеного:

```

todolists = Todolist.objects.filter(family = familyparam)
restodolists = []
for todo in todolists:
    res = True
    for item in todo.items:
        if item.done == 0:
            res = False
    if res:
        restodolists.append(todo)
restodolists = restodolists.order_by('-created')
```

Підтримка збереження списку справ до власного списку справ, реалізується через клас `TodolistActiveAddView`. Сам список передається в шаблон вебсторінки `todolistactiveadd.html`. Номер списку отримується через

параметр методу, дані користувача визначаються за даними авторизації. Стан збереження списку фіксується через відсутність або наявність значення, яке було передано через параметр `end`.

Для забезпечення редагування збережень списків справ створено клас `TodolistChangeView` на основі класу `UpdateView`. Цей клас реалізує автоматичне переключення стану збереження на протилежний за кожного звернення до нього. Тобто якщо цей список мав відображатися активно, то стан перемикається на неактивний, а якщо навпаки був неактивний, то перемикається на активний.

Для підтримки додавання нової зустрічі створено клас `MeetupAddView`, який ґрунтуючись на класі `CreateView`, виконує заповнення назви зустрічі, адреси, за якою відбудеться зустріч, часу початку і завершення зустрічі, номеру телефону через поля форми напряму та програмно задає дату створення зустрічі як поточну, визначає автором списку авторизованого користувача та визначає доступ до інформації про цю зустріч тільки для тієї спільноти сім'ї, номер якої було передано через параметр методу, тобто номер, який було обрано самим користувачем на початку роботи з застосунком або змінено після і який постійно зберігається на клієнтській стороні.

Для забезпечення редагування інформації про зустріч створено клас `MeetupEditView` на основі класу `UpdateView`. Редагування значення назви зустрічі, адреси, за якою відбудеться зустріч, часу початку і завершення зустрічі, номеру телефону реалізується на основі підтримки роботи з формою через відповідні її поля.

Підтримка виведення всіх зустрічей, які належать спільноті певної сім'ї, реалізується через клас `FamilyMeetupsListView`. Сам список передається в шаблон вебсторінки `familymeetups.html`. Номер сім'ї, спільноту якої обрав користувач, передається через параметр `family`. Усі зустрічі, які відповідають цій сім'ї, розташовуються за датою створення, починаючи з останньої за часом виконання.

Підтримка виведення всіх зустрічей, які належать спільноті певної сім'ї і які ще не були реалізовані, реалізується через клас `FamilyNexMeetupsListView`. Сам список передається в шаблон вебсторінки `familynextmeetups.html`. Номер сім'ї, спільноту якої обрав користувач, передається через параметр `family`. Усі зустрічі, які мають час реалізації пізніший за поточний і відповідають цій сім'ї, розташовуються за датою створення, починаючи з найближчої.

Реалізацію перегляду інформації про зустріч виконано на основі класу `MeetupView`, який реалізовано на основі класу `DetailView`. У методі `get_context_data` цього класу під час звернення до нього перевіряється, чи вже було зроблено запис про те, що цей авторизований користувач переглядав інформацію цієї зустрічі, відповідно якщо ні, то виконується запис, лічильник переглядів цієї зустрічі оновлюється, а якщо так, то ніяких змін не реалізовується.

Додавання членів сім'ї до учасників зустрічі реалізовано через клас `MeetupMemberAddView`, який реалізовано на основі класу `CreateView`. Номер зустрічі передається через параметр методу, так само ідентифікатор члена сім'ї.

Для забезпечення зміни учасників зустрічі створено клас `MeetupMembersChangeView` на основі класу `UpdateView`. Цей клас реалізує оновлення списку учасників зустрічі на основі значення параметра `members`. Цей список аналізується і виділяються ті члени сім'ї, які не були внесені до списку учасників (вони вносяться), і ті, які були вилучені зі списку учасників (вони вилучаються).

Підтримка виведення всіх зустрічей, учасником яких є авторизований користувач, реалізується через клас `MemberMeetupsListView`. Сам список передається в шаблон вебсторінки `memberfamilymeetups.html`. Спочатку обираються всі записи про участь авторизованого користувача в зустрічах, а потім виділяються повні об'єкти зустрічей під ці записи, а створений список

впорядковується за часом початку зустрічей, починаючи з останньої за часом виконання.

Для підтримки оцінювання зустрічі створено клас MeetupEstimationAddView, який ґрунтуючись на класі CreateView, виконує заповнення оцінки зустрічі через поле форми напряму та програмно задає дані про члена сім'ї, який цю оцінку виставив за даними авторизації, а також оновлює загальну оцінку зустрічі на основі виставленої оцінки одразу:

```
mes = MeetupEstimation.objects.filter(meetup = meetup)
score = 0
scoreq = mes.count()
for me in mes:
    score = score + me/scoreq
meetup.score = score
```

Підтримка виведення всіх оцінок зустрічі реалізується через клас MeetupEstimationsListView. Сам список передається в шаблон вебсторінки meetupestimations.html. Номер зустрічі передається через параметр family. Усі оцінки розташовуються за номером оцінки, починаючи з останнього.

Для підтримки додавання інформаційного повідомлення створено клас InformationAddView, який ґрунтуючись на класі CreateView, виконує заповнення назви інформаційного повідомлення, його тексту через поля форми напряму та програмно задає дату створення інформаційного повідомлення як поточну, визначає автором інформаційного повідомлення авторизованого користувача та визначає доступ до нього тільки для тієї спільноти сім'ї, номер якої було передано через параметр методу, тобто використовуючи номер, який було обрано самим користувачем на початку роботи з застосунком або змінено після і який постійно зберігається на клієнтській стороні.

Для забезпечення редагування інформаційного повідомлення створено клас InformationEditView на основі класу UpdateView. Редагування значення

назви повідомлення, його тексту реалізується на основі підтримки роботи з формою через відповідні її поля.

Реалізацію перегляду інформаційного повідомлення виконано на основі класу `InformationView`, який реалізовано на основі класу `DetailView`. У методі `get_context_data` цього класу під час звернення до нього перевіряється, чи вже було зроблено запис про те, що цей авторизований користувач переглядав це інформаційне повідомлення, відповідно якщо ні, то виконується запис, лічильник переглядів цього інформаційного повідомлення оновлюється, а якщо так, то ніяких змін не реалізовується.

Підтримка виведення всіх інформаційних повідомлень спільноти сім'ї реалізується через клас `InformationsListView`. Сам список передається в шаблон вебсторінки `familyinformations.html`. Номер зустрічі передається через параметр `family`. Усі оцінки розташовуються за датою створення, починаючи з останнього.

Для підтримки уподобання інформаційного повідомлення створено клас `InformationLikeAddView`, який ґрунтуючись на класі `CreateView`, виконує заповнення через передані значення параметрів інформаційного повідомлення, яке уподобано, члена сім'ї, який уподобав повідомлення, окрім того оновлюється кількість уподобань у відповідного інформаційного повідомлення.

Підтримка виведення всіх уподобань інформаційного повідомлення реалізується через клас `InformationLikesListView`. Сам список передається в шаблон вебсторінки `informationlikess.html`. Номер інформаційного повідомлення передається через параметр `information`. Усі уподобання розташовуються за датою створення, починаючи з останнього.

Для підтримки відправлення повідомлення членом сім'ї в обговорення створено клас `MemberMessageAddView`, який ґрунтуючись на класі `CreateView`, виконує заповнення через передане значення параметра тексту повідомлення, спільноти сім'ї, до якої відправлено повідомлення, також на

основі даних авторизації вноситься автор повідомлення, а після встановлюється час публікації повідомлення як поточний.

Підтримка виведення всіх повідомлень обговорення сім'ї реалізується через клас `MemberMessagesListView`. Сам список передається в шаблон вебсторінки `familymembermessages.html`. Номер сім'ї передається через параметр `family`. Усі повідомлення розташовуються за датою створення, починаючи з останнього.

Для підтримки уподобання повідомлення з обговорення створено клас `MemberMessageLikeAddView`, який ґрунтуючись на класі `CreateView`, виконує заповнення через передані значення параметрів повідомлення, яке уподобано, члена сім'ї, який уподобав повідомлення, окрім того оновлюється кількість уподобань у відповідного повідомлення.

Вилучення уподобання повідомлення в обговоренні реалізовано через клас `MemberMessageLikeDeleteView`, який ґрунтується на основі класу `DeleteView`.

Для підтримки відправлення повідомлення членом сім'ї щодо зустрічі створено клас `MeetupMessageAddView`, який ґрунтуючись на класі `CreateView`, виконує заповнення через передане значення з поля форми тексту повідомлення, а також через параметри зустрічі, до якої стосується повідомлення, також на основі даних авторизації вноситься автор повідомлення, а після встановлюється час публікації повідомлення як поточний.

Для підтримки відправлення повідомлення членом сім'ї щодо списку справ створено клас `TodolistMessageAddView`, який ґрунтуючись на класі `CreateView`, виконує заповнення через передане значення з поля форми тексту повідомлення, а також через параметри списку справ, до якого стосується повідомлення, також на основі даних авторизації вноситься автор повідомлення, а після встановлюється час публікації повідомлення як поточний.

Для підтримки відправлення повідомлення членом сім'ї щодо інформаційного повідомлення створено клас `InformationMessageAddView`, який ґрунтуючись на класі `CreateView`, виконує заповнення через передане значення з поля форми тексту повідомлення, а також через параметри інформаційного повідомлення, до якого стосується повідомлення, також на основі даних авторизації вноситься автор повідомлення, а після встановлюється час публікації повідомлення як поточний.

Для підтримки відправлення голосу в голосуванні створено клас `VotingVoteAddView`, який ґрунтуючись на класі `CreateView`, виконує заповнення через параметри варіанту голосування, за який віддано голос, також на основі даних авторизації вноситься автор повідомлення, а після цього оновлюється кількість голосів за кожний варіант.

Реалізацію перегляду голосування виконано на основі класу `VotingView`, який реалізовано на основі класу `DetailView`. У методі `get_context_data` цього класу під час звернення до нього перевіряється, чи вже було зроблено запис про те, що цей авторизований користувач переглядав голосування, відповідно якщо ні, то виконується запис, лічильник переглядів цього голосування оновлюється, а якщо так, то ніяких змін не реалізовується.

3.3 Висновки за розділом 3

Реалізовано програму планування справ у сім'ї за допомогою фреймворку веброзробки Django. У розділі описано створені в процесі класи програми.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для планування і обговорення справ у сім'ї, розглядаючи сім'ю як деяку спільноту, яка потребує впорядкованого, але обмеженого доступу до інформації про справи. Для цього програма забезпечує роботу з даними стосовно зустрічей, в яких бере участь вся сім'я або її члени, стосовно справ, які потрібно виконати, стосовно розповсюдження інформаційних повідомлень, обговорення різних питань, голосування за невирішеними питаннями.

4.2 Умови виконання програми

Технічні засоби в мінімальній конфігурації, достатній для роботи програми, охоплюють 1 Гб пам'яті жорсткого диску, 4 Гб оперативної пам'яті, 4 ядра процесору (частота 2,8 ГГц).

4.3 Тестування і опис принципів експлуатації програми

Для виконання тестування було розгорнуто ПЗ на власному вебсервері з наданням доступу до нього з мережі. Таким чином, використовуючи браузер на клієнтському пристрої, було перевірено результати реалізації функціональних вимог до програми. Кожна функція, що стосується додавання і редагування даних перевірялась в декількох режимах, включаючи як коректний сценарій, так і внесення некоректних даних і перевірку реакції програми на таку дію. Співставлення результатів спочатку дозволило виконати проблеми в реалізації деяких функцій, а під час остаточного тестування вказало на успішність реалізації функцій.

Сценарій роботи з програмою загалом є достатньо інтуїтивно зрозумілий. Після звернення до вебсервера програма обов'язково вимагає

авторизації або реєстрації. Працювати з програмою без цього неможливо, адже всі дані є даними з обмеженим доступом для певної спільноти. Користувач має свідвідноситися з якоюсь спільнотою.

Після авторизації користувач починає працювати з інструментами, які доступні для нього (рис. 4.1).

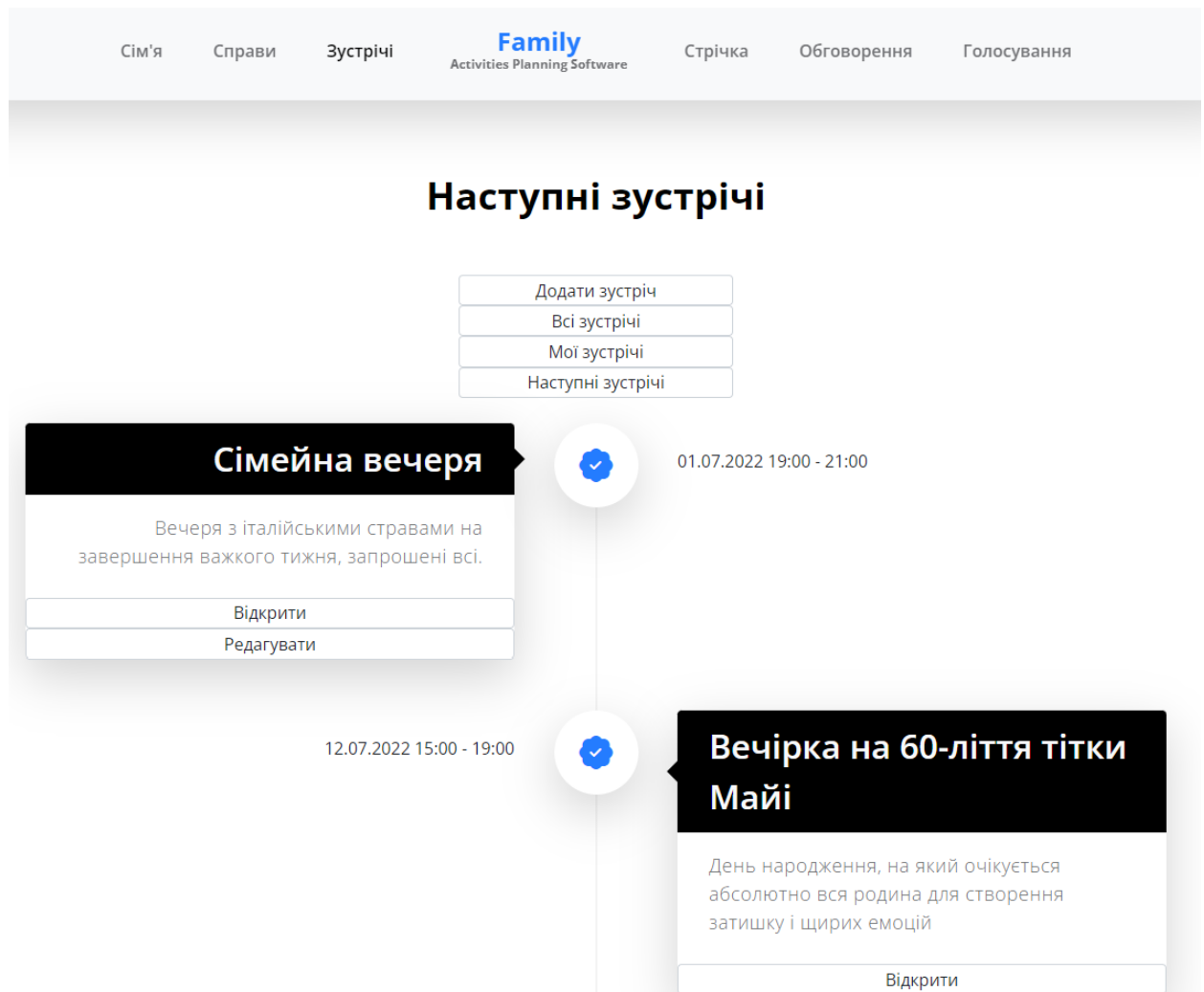


Рисунок 4.1 – Перегляд майбутніх зустрічей сім'ї

За замовчуванням одразу відкривається сторінка, яка в меню позначена як «Сім'я». Ця сторінка дозволяє переглянути перелік спільнот сімей, до яких відноситься користувач. За замовчуванням це може бути одна сім'я. Якщо в списку декілька, користувач має обрати одну з них для використання в усіх інших функціях застосунку. Окрім того звідси можна створити нову спільноту, можна визначити доступ інших користувачів до

кожної спільноти, якщо її створив користувач, який авторизувався в програмі, використовуючи кнопки редагування. Тим не менше першочерговим є вибір за допомогою кнопки «Зробити базовою» спільноти, за якою будуть відображатися дані на всіх інших сторінках.

Якщо користувач потребує змінити дані свого облікового запису, то для цього призначена центральна частина меню з назвою програми.

Розділ меню «Справи» дозволяє переглядати списки справ у режимах повного списку всіх справ сім'ї або в режимі невиконаних справ. Окрім того звідси можна додавати нові і редагувати існуючі списки, відкривати один зі списків, редагувати самі справи або змінювати тільки стан виконання справи. Списки справ необхідні для того, щоб наприклад, внести список продуктів, які необхідно придбати в магазині, створити список страв, які необхідно приготувати тощо. Кожен список справ – це список, у якому можна позначати, чи виконанню справу, чи ні за допомогою галочки.

Розділ «Зустрічі» дозволяє переглядати зустрічі в двох режимах, а також додавати нові зустрічі. Для останнього доступна відповідна кнопка зверху. Кнопка «Всі зустрічі» дозволяє відтворити всі зустрічі сім'ї, починаючи з тієї, яка відбудеться найпізніше. Кнопка «Наступні зустрічі» відображає на сторінці тільки ті зустрічі, які відбудуться не раніше поточного моменту часу. Починаючи з найближчої зустрічі, усі вони відтворюються в ланцюжку. На кожній позиції відображається відповідна зустріч, зокрема її назва і опис, надаються кнопки «Відкрити» для відкриття повної сторінки, де окрім повної інформації також надається інформація про те, хто переглянув зустріч, хто запрошений на зустріч, а також повідомлення (коментарі) до зустрічі. Якщо авторизований користувач створив цю зустріч в системі, то йому також надається кнопка «Редагувати» для змінювання даних.

Розділ меню «Стрічка» дозволяє переглянути за датою створення інформаційні повідомлення сім'ї. Це фактично новини.

Розділ меню «Обговорення» дозволяє спілкуватися між собою одразу всім членам сім'ї (рис. 4.2). При цьому відтворюється перелік 10 останніх повідомлень від членів сім'ї. Надається можливість відправити власне повідомлення. Кожні 10 повідомлень відтворюються на окремій картці, для того щоб відтворити попередні 10 повідомлень, необхідно переміститись на сусідню картку (стрілки посередині картки вліво і вправо або потягнути).

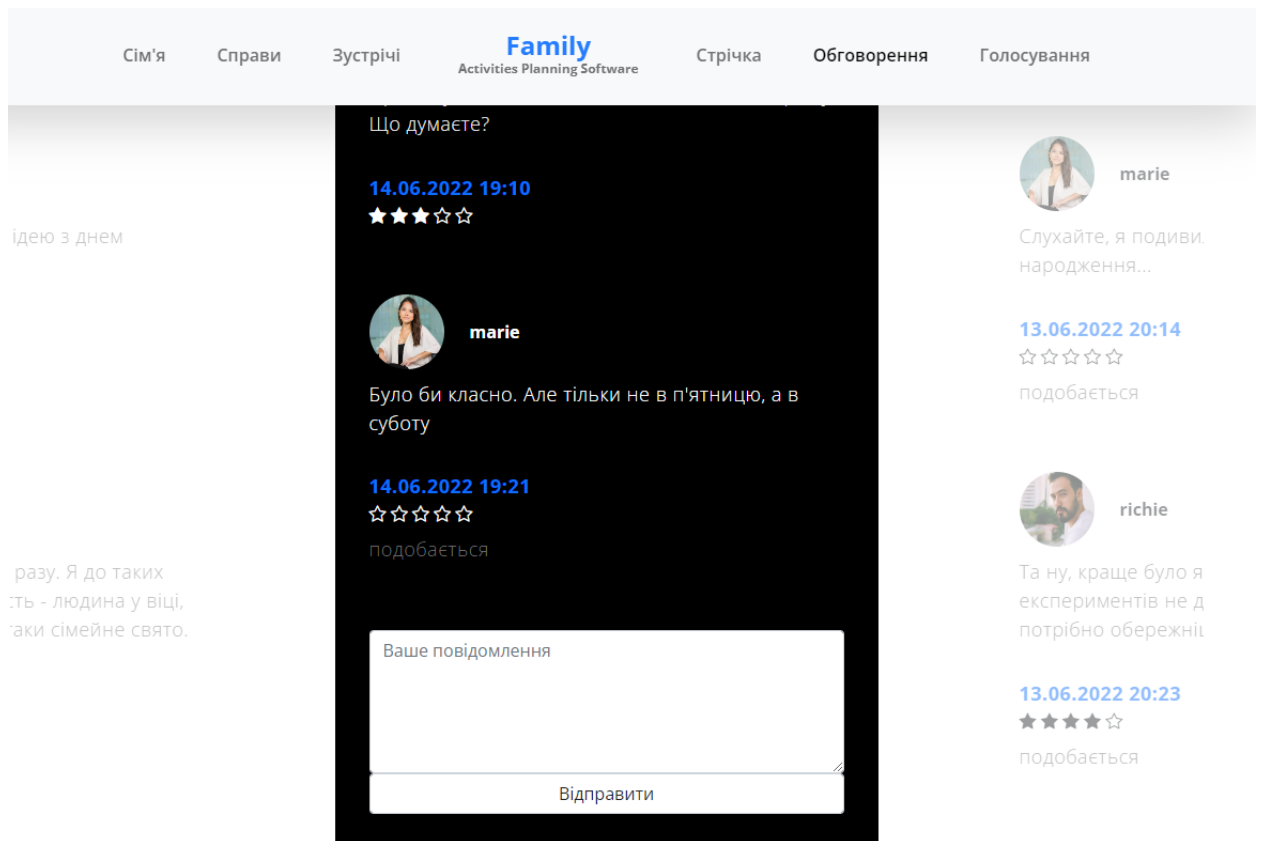


Рисунок 4.2 – Загальне обговорення в програмі

Для кожного повідомлення відображається його автор, текст повідомлення, дата і час публікації, кількість уподобань. Кількість уподобань відображається зірочками. Максимальна кількість зірочок – кількість членів сім'ї. Якщо сім'я містить більше 10 членів, то значення відображається числом. Підсвічені зірочки – це поточна кількість уподобань. Якщо авторизований користувач ще не уподобав повідомлення, то йому відображається посилання «подобається», щоб встановити уподобання повідомленню.

Розділ меню «Голосування» (рис. 4.3) дозволяє працювати над голосуваннями, створюючи нові, переглядаючи і беручи участь в існуючих. При відкритті голосування відображається його назва, зміст, час, до якого голосування триває, кількість переглядів голосування членами сім'ї (можна відкрити сторінку зі списком тих, хто переглянув голосування), члена сім'ї, який створив голосування, а також самі варіанти вибору. За кожним варіантом відображається кількість відданих голосів і поточний відсоток. Якщо користувач ще не голосував, то йому відображаються кнопки «Голосувати» біля кожного варіанту вибору.

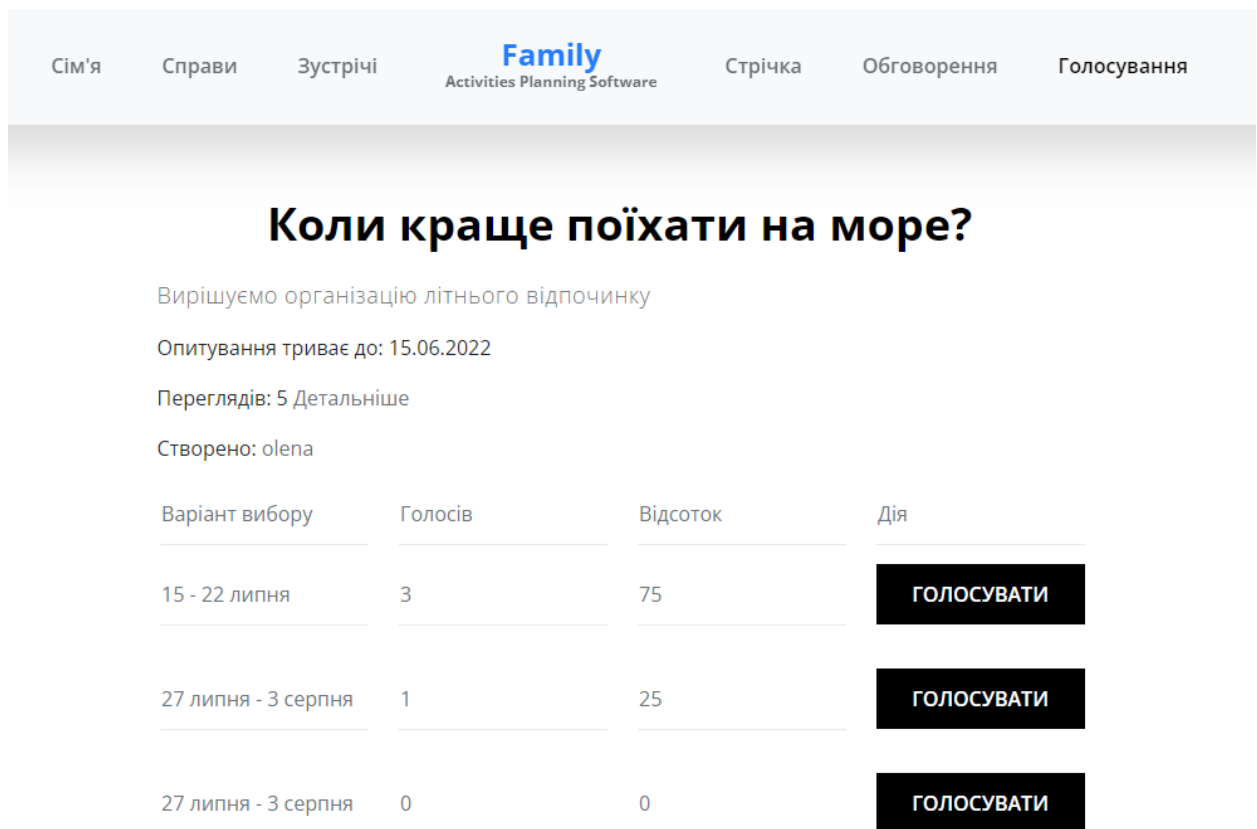


Рисунок 4.3 – Голосування

4.4 Висновки за розділом 4

Описано основні сценарії роботи користувача з програмою за допомогою відповідних інтерфейсів. Представлено можливості використання програми.

ВИСНОВКИ

У дипломній роботі проводилось розроблення ПЗ планування справ у сім'ї для впорядкування спільного доступу до інформації в сім'ї. Проведений аналіз предметної області дозволив виявити аналоги і визначити їх активне використання на практиці, що говорить про важливість проблеми, а також визначити шляхи покращення сценаріїв використання програми користувачами.

Під час проектування ПЗ виконано вибір мови програмування для реалізації програми на користь мови Python, виконано вибір функцій, які має забезпечувати програма, створено діаграму прецедентів, розроблено структуру БД, обрано систему керування БД MySQL для підтримки її роботи, описано створені таблиці та зв'язки між ними.

Реалізовано програму з забезпеченням всіх запланованих функцій на основі реалізації відповідних класів підтримки їх виконання на основі взаємодії з даними та підготовки цих даних для відображення на відповідних вебсторінках. Описано створені в процесі класи програми. Описано основні сценарії роботи користувача з програмою за допомогою відповідних інтерфейсів. Представлено можливості використання програми.

Розроблена в результаті виконання роботи програма призначена для планування і обговорення справ у сім'ї, розглядаючи сім'ю як деяку спільноту, яка потребує впорядкованого, але обмеженого доступу до інформації про справи. Для цього програма забезпечує роботу з даними стосовно зустрічей, в яких бере участь вся сім'я або її члени, стосовно справ, які потрібно виконати, стосовно розповсюдження інформаційних повідомлень, обговорення різних питань, голосування за невирішеними питаннями.

Поставлені спочатку завдання дипломної роботи в підсумку були успішно виконані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cozi Family Organizer. Must-Have App For Families [Electronic resource]. – Access mode : <https://www.cozi.com/>.
2. Jones, T. Top 10 Best Family Organizer Apps of 2022 [Electronic resource] / T. Jones. – Access mode : <https://famisafe.wondershare.com/family/best-family-organizer-apps.html>.
3. Feyoh, M. 11 Best Shared Family Calendar Apps & Organizers [Electronic resource] / M. Feyoh. – Access mode : <https://www.developgoodhabits.com/family-calendar-apps/>.
4. Ward, K. The 8 best calendar and organizer apps for families [Electronic resource] / K. Ward. – Access mode : <https://www.care.com/c/best-calendar-organizer-apps-families/>.
5. Hub Family Organizer by Ironark, Inc. [Electronic resource]. – Access mode : <https://appadvice.com/app/hub-family-organizer/614810693>.
6. The Best Family Organizer Apps of 2022 [Electronic resource]. – Access mode : <https://www.bestapp.com/best-family-calendar-apps/>.
7. Shaw, Z. Learn Python 3 the Hard Way [Text] : A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code / Z. Shaw. – Boston, Massachusetts : Addison-Wesley Professional, 2017. – 320 p.
8. Python Tutorial [Electronic resource]. – Access mode : <https://www.w3schools.com/python/>.
9. Python 3.10.5 documentation [Electronic resource]. – Access mode : <https://docs.python.org/3/>.

ДОДАТОК А
Технічне завдання

Вступ

Сім'я фактично може розглядатися як спільнота, кожен член якої отримує доступ до інформації, до якої немає доступу іншим користувачам програми. Члени сім'ї потребують накопичення даних про зустрічі, списки справ, обговорення деяких питань, перегляду інформаційних повідомлень, щоб бути впевненими в тому, що кожен з членів сім'ї має доступ до такої інформації і за допомогою програми може не пропустити спільну подію, виконати спільні дії тощо.

A.1 Підстави для розробки

Розробка в межах дипломної кваліфікаційної роботи реалізується на підставі наказу № 102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка» за темою «Програмне забезпечення планування справ у сім'ї».

A.2 Призначення розробки

Програма призначена для планування справ у сім'ї, обговорення цих справ.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функції, які виконуватиме програма за результатами розробки:

- додавання нової спільноти сім'ї;
- керування членами спільноти сім'ї (додавання і вилучення);
- авторизація і реєстрація користувачів;
- перегляд доступних спільнот сім'ї;

- додавання нового списку справ для спільноти сім'ї;
- змінювання списку справ для спільноти сім'ї;
- внесення справ до списку справ;
- перегляд списку справ;
- перегляд всіх списків справ сім'ї;
- збереження списку справ;
- перегляд збережених списків справ;
- додавання нової зустрічі в спільноту сім'ї;
- редагування існуючої зустрічі;
- внесення членів сім'ї для участі в зустрічі;
- перегляд всіх запланованих зустрічей сім'ї;
- перегляд інформації про зустріч;
- оцінка зустрічі;
- додавання нового інформаційного повідомлення;
- змінювання існуючого інформаційного повідомлення;
- перегляд всіх інформаційних повідомлень для членів сім'ї;
- уподобання інформаційного повідомлення;
- додавання нового повідомлення до загального обговорення;
- уподобання повідомлення з загального обговорення;
- перегляд всіх повідомлень у обговоренні;
- коментування зустрічі;
- коментування списку справ;
- коментування інформаційного повідомлення;
- додавання нового голосування;
- змінювання голосування;
- внесення варіантів вибору в голосуванні;
- голосування;
- перегляд членів сім'ї, які переглянули список справ;
- перегляд членів сім'ї, які переглянули інформацію про зустріч;

- перегляд членів сім'ї, які переглянули інформаційне повідомлення повністю;
- перегляд членів сім'ї, які переглянули голосування повністю;
- перегляд майбутніх зустрічей сім'ї;
- перегляд списків незроблених справ;
- перегляд зустрічей, в яких бере участь користувач.

А.3.2 Умови експлуатації

Рекомендовано для забезпечення роботи програми виділити системного адміністратора, до зобов'язань якого віднести підтримку роботи вебсервера, сервера БД та внесення за потреби базових даних.

А.3.3 Вимоги до складу та параметрів технічних засобів

Технічні засоби в мінімальній конфігурації, достатній для роботи програми, охоплюють 1 Гб пам'яті жорсткого диску, 4 Гб оперативної пам'яті, 4 ядра процесору (частота 2,8 ГГц).

А.4 Порядок контролю та приймання

Остаточна перевірка отриманих результатів виконання роботи здійснюється під час захисту дипломної роботи.

ДОДАТОК Б
Текст програми

```

from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import ListView
from django.views.generic import UpdateView
from django.views.generic.edit import CreateView
from django.views.generic.edit import DeleteView
from django.views.generic.detail import DetailView
from django.urls import reverse_lazy
from .models import Family
from .models import Member
from .models import Todolist
from .models import TodolistItem
from .models import TodolistSeen
from .models import TodolistActive
from .models import Meetup
from .models import MeetupMember
from .models import MeetupSeen
from .models import MeetupEstimation
from .models import Information
from .models import InformationSeen
from .models import InformationLike
from .models import MemberMessage
from .models import MeetupMessage
from .models import TodolistMessage
from .models import InformationMessage
from .models import Voting
from .models import VotingItem
from .models import VotingSeen
from .models import VotingVote
from datetime import date

class FamilyAddView(LoginRequiredMixin, CreateView):
    model = Family
    fields = ['name']
    template_name = 'familyadd.html'

    def get_success_url(self):
        return reverse_lazy('familyview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        return super().form_valid(form)

```

```

class FamilyEditView(LoginRequiredMixin, UpdateView):
    model = Family
    fields = ['name']
    template_name = 'familyedit.html'

    def get_success_url(self):
        return reverse_lazy('familyview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        members = str(self.request.POST.get('members'))
        members = members.split(',')
        members_new = []
        members_old = self.members
        for member in members:
            mem = Member.objects.get(id = member)
            members_new.append(mem)
            if mem not in members_old:
                self.members.add(mem)
        for member in members_old:
            if member not in members_new:
                self.members.remove(mem)
        return super().form_valid(form)

class FamiliesListView(ListView):
    model = Family
    template_name = 'families.html'

    def get_queryset(self):
        user = self.request.user
        member = Member(id = user.id)
        families = Family.objects.filter(members__contains =
mamber)

        families = families.order_by('name')
        return families

class TodolistAddView(LoginRequiredMixin, CreateView):
    model = Todolist
    fields = ['title', 'todotext']
    template_name = 'todolistadd.html'

    def get_success_url(self):
        return reverse_lazy('todolistview', kwargs={'pk':
self.object.pk})

```

```

    def get_context_data(self, **kwargs):
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        kwargs['family'] = family
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.created = date.today()
        self.seencount = 0
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        self.family = family
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        todoitems = str(self.request.POST.get('items'))
        todoitems = todoitems.split(',')
        todoitems_new = []
        todoitems_old = self.items
        itemtexts = str(self.request.POST.get('itemtexts'))
        itemtexts = itemtexts.split('++')
        count = 0
        for todoitem in todoitems:
            mem = None
            try:
                mem = TodolistItem.objects.get(id =
todoitem)

            except SomeModel.DoesNotExist:
                TodolistItem.objects.add(itemtext =
itemtexts[count], done = 0)
                count = count + 1
        return super().form_valid(form)

class TodolistView (DetailView):
    template_name = 'todolist.html'
    model = Todolist

    def get_context_data(self, **kwargs):
        data = super(TodolistView,
self).get_context_data(**kwargs)
        user = self.request.user
        memberparam = Member.objects.get(id = user.id)
        try:
            td = TodolistSeen(todolist__id = self.object.pk,
member = memberparam)

```

```

        except TodolistSeen.DoesNotExist:
            td = TodolistSeen.objects.add(todolist__id =
self.object.pk, member = memberparam)
            td.save()
            self.object.seencount = self.object.seencount + 1
        return data

class MemberAddView(LoginRequiredMixin, CreateView):
    model = Member
    fields = ['name', 'username', 'password']
    template_name = 'memberadd.html'

    def get_success_url(self):
        return reverse_lazy('memberview', kwargs={'pk':
self.object.pk})

class MemberEditView(LoginRequiredMixin, UpdateView):
    model = Member
    fields = ['name', 'username', 'password']
    template_name = 'memberupdate.html'

    def get_success_url(self):
        return reverse_lazy('memberview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        user = self.request.user
        member = Member(id = user.id)
        name = str(self.request.POST.get('name'))
        username = str(self.request.POST.get('username'))
        password = str(self.request.POST.get('password'))
        oldpassword =
str(self.request.POST.get('oldpassword'))
        newpassword =
str(self.request.POST.get('newpassword'))
        if member.username == username and member.password ==
oldpassword:
            if newpassword:
                self.password = newpassword
                self.name = name
            else:
                self.name = name
        return super().form_valid(form)

```

```

class TodolistEditView(LoginRequiredMixin, UpdateView):
    model = Todolist
    fields = ['title', 'todotext']
    template_name = 'todolistadd.html'

    def get_success_url(self):
        return reverse_lazy('todolistview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        todoitems = str(self.request.POST.get('items'))
        todoitems = todoitems.split(',')
        dones = str(self.request.POST.get('dones'))
        dones = dones.split(',')
        todoitems_new = []
        todoitems_old = self.items
        itemtexts = str(self.request.POST.get('itemtexts'))
        itemtexts = itemtexts.split('++')
        count = 0
        for todoitem in todoitems:
            mem = None
            try:
                mem = TodolistItem.objects.get(id =
todoitem)

                if mem.done == 0 and dones[count]:
                    mem.done = 1
                    mem.done_date = date.today()
                    mem.save()
                elif mem.done == 1 and not dones[count]:
                    mem.done = 0
                    mem.save()
            except TodolistItem.DoesNotExist:
                TodolistItem.objects.add(itemtext =
itemtexts[count], done = 0)
            count = count + 1
        for todoitem in todoitems_old:
            if todoitem not in todoitems_new:
                self.items.remove(todoitem)
        return super().form_valid(form)

class FamilyTodolistsListView(ListView):
    model = Todolist

```

```

        template_name = 'familytodos.html'

    def get_queryset(self):
        familyparam = get_object_or_404(Family,
id=self.kwargs['family'])
        todolists = Todolist.objects.filter(family =
familyparam)
        todolists = todolists.order_by('-created')
        return todolists

    def get_context_data(self, **kwargs):
        famdata = super(FamilyTodolistsListView,
self).get_context_data(**kwargs)
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        famdata ['family'] = family
        return famdata

class FamilyUndoneTodolistsListView(ListView):
    model = Todolist
    template_name = 'familyundonetodos.html'

    def get_queryset(self):
        familyparam = get_object_or_404(Family,
id=self.kwargs['family'])
        todolists = Todolist.objects.filter(family =
familyparam)

        restodolists = []
        for todo in todolists:
            res = True
            for item in todo.items:
                if item.done == 0:
                    res = False
            if res:
                restodolists.append(todo)
        restodolists = restodolists.order_by('-created')
        return restodolists

    def get_context_data(self, **kwargs):
        famdata = super(FamilyUndoneTodolistsListView,
self).get_context_data(**kwargs)
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        famdata ['family'] = family
        return famdata

```

```

class TodolistActiveAddView(LoginRequiredMixin, CreateView):
    model = TodolistActive
    template_name = 'todolistactiveadd.html'

    def get_success_url(self):
        return reverse_lazy('todolistactiveview',
kwargs={'pk': self.object.pk})

    def get_context_data(self, **kwargs):
        todolist = get_object_or_404(Todolist,
id=self.kwargs['todolist'])
        kwargs['todolist'] = todolist
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.todolist = get_object_or_404(Todolist,
id=self.kwargs['todolist'])
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        if self.kwargs['end']:
            self.state = 1
        else:
            self.state = 0
        return super().form_valid(form)

class TodolistChangeView(LoginRequiredMixin, UpdateView):
    model = TodolistActive
    template_name = 'todolistactivechange.html'

    def get_success_url(self):
        return reverse_lazy('todolistactiveview',
kwargs={'pk': self.object.pk})

    def form_valid(self, form):
        if self.state == 0:
            self.state = 1
        else:
            self.state = 0
        return super().form_valid(form)

class MeetupAddView(LoginRequiredMixin, CreateView):
    model = Meetup
    fields = ['title', 'address', 'start_time', 'end_time',
'phone']

```

```

        template_name = 'meetupadd.html'

        def get_success_url(self):
            return reverse_lazy('meetupview', kwargs={'pk':
self.object.pk})

        def get_context_data(self, **kwargs):
            family = get_object_or_404(Family,
id=self.kwargs['family'])
            kwargs['family'] = family
            return super().get_context_data(**kwargs)

        def form_valid(self, form):
            self.created = date.today()
            self.seen_count = 0
            self.score = 0
            family = get_object_or_404(Family,
id=self.kwargs['family'])
            self.family = family
            user = self.request.user
            member = Member(id = user.id)
            self.member = member
            return super().form_valid(form)

class MeetupEditView(LoginRequiredMixin, UpdateView):
    model = Meetup
    fields = ['title', 'address', 'start_time', 'end_time',
'phone']
    template_name = 'meetupedit.html'

    def get_success_url(self):
        return reverse_lazy('meetupview', kwargs={'pk':
self.object.pk})

class FamilyMeetupsListView(ListView):
    model = Meetup
    template_name = 'familymeetups.html'

    def get_queryset(self):
        familyparam = get_object_or_404(Family,
id=self.kwargs['family'])
        meetups = Meetup.objects.filter(family = familyparam)
        meetups = meetups.order_by('-start_time')
        return meetups

```

```

        def get_context_data(self, **kwargs):
            famdata = super(FamilyMeetupsListView,
self).get_context_data(**kwargs)
            family = get_object_or_404(Family,
id=self.kwargs['family'])
            famdata ['family'] = family
            return famdata

class FamilyNexMeetupsListView(ListView):
    model = Meetup
    template_name = 'familynextmeetups.html'

    def get_queryset(self):
        familyparam = get_object_or_404(Family,
id=self.kwargs['family'])
        meetups = Meetup.objects.filter(family = familyparam,
start_time__gte = date.today())
        meetups = meetups.order_by('start_time')
        return meetups

    def get_context_data(self, **kwargs):
        famdata = super(FamilyNexMeetupsListView,
self).get_context_data(**kwargs)
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        famdata ['family'] = family
        return famdata

class MeetupView (DetailView):
    template_name = 'meetup.html'
    model = Meetup

    def get_context_data(self, **kwargs):
        data = super(MeetupView,
self).get_context_data(**kwargs)
        user = self.request.user
        memberparam = Member.objects.get(id = user.id)
        try:
            td = MeetupSeen (meetup__id = self.object.pk,
member = memberparam)
        except MeetupSeen.DoesNotExist:
            td = MeetupSeen.objects.add(meetup__id =
self.object.pk, member = memberparam)
            td.save()

```

```

        self.object.seen_count = self.object.seen_count +
1
        return data

class MeetupMemberAddView(LoginRequiredMixin, CreateView):
    model = MeetupMember
    template_name = 'meetupmemberadd.html'

    def get_success_url(self):
        return reverse_lazy('meetupview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        meetup = get_object_or_404(Meetup,
id=self.kwargs['meetup'])
        kwargs['meetup'] = meetup
        member = get_object_or_404(Member,
id=self.kwargs['member'])
        kwargs['member'] = member
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        meetup = get_object_or_404(Meetup,
id=self.kwargs['meetup'])
        self.meetup = meetup
        member = get_object_or_404(Member,
id=self.kwargs['member'])
        self.member = member
        self.done = 0
        return super().form_valid(form)

class MeetupMembersChangeView(LoginRequiredMixin, UpdateView):
    model = MeetupMember
    template_name = 'meetupmemberschange.html'

    def get_success_url(self):
        return reverse_lazy('meetupview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        members = self.kwargs['members']
        members = members.split(',')
        dones = self.kwargs['dones']
        dones = dones.split(',')
        mu = self.meetup

```

```

        user = self.request.user
        memb = Member(id = user.id)
        mms = []
        count = 0
        for member in members:
            try:
                mm = MeetupMember.objects.filter(meetup =
mu, member = memb)

                mm.done = done[count]
                mm.save()
            except MeetupMember.DoesNotExist:
                MeetupMember.objects.create(meetup = mu,
member = memb, done = done[count])
                mms.append(member)
                count = count + 1
        for mm in mms:
            if mm not in members:
                MeetupMember.objects.filter(meetup = mu,
member = memb).delete()
        return super().form_valid(form)

```

```

class MemberMeetupsListView(ListView):
    model = Meetup
    template_name = 'memberfamilymeetups.html'

```

```

    def get_queryset(self):
        user = self.request.user
        memb = Member(id = user.id)
        mms = MeetupMember.objects.filter(member = memb)
        meetups = []
        for mm in mms:
            meetups.append(mm.meetup)
        meetups = meetups.order_by('-start_time')
        return meetups

```

```

class MeetupEstimationAddView(LoginRequiredMixin, CreateView):
    model = MeetupEstimation
    template_name = 'meetupestimationadd.html'
    fields = ['score']

```

```

    def get_success_url(self):
        return reverse_lazy('meetupview', kwargs={'pk':
self.object.pk})

```

```

    def get_context_data(self, **kwargs):

```

```

        meetup = get_object_or_404(Meetup,
id=self.kwargs['meetup'])
        kwargs['meetup'] = meetup
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        meetup = get_object_or_404(Meetup,
id=self.kwargs['meetup'])
        self.meetup = meetup
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        mes = MeetupEstimation.objects.filter(meetup = meetup)
        score = 0
        scoreq = mes.count()
        for me in mes:
            score = score + me/scoreq
        meetup.score = score
        meetup.save()
        return super().form_valid(form)

class MeetupEstimationsListView(ListView):
    model = MeetupEstimation
    template_name = 'meetupestimations.html'

    def get_queryset(self):
        meetup = get_object_or_404(Meetup,
id=self.kwargs['meetup'])
        meetupestimations =
MeetupEstimation.objects.filter(meetup = meetup)
        meetupestimations = meetupestimations.order_by('-id')
        return meetupestimations

    def get_context_data(self, **kwargs):
        famdata = super(MeetupEstimationsListView,
self).get_context_data(**kwargs)
        meetup = get_object_or_404(Meetup,
id=self.kwargs['meetup'])
        famdata ['meetup'] = meetup
        return famdata

class InformationAddView(LoginRequiredMixin, CreateView):
    model = Information
    fields = ['title', 'info_text']
    template_name = 'informationadd.html'

```

```

    def get_success_url(self):
        return reverse_lazy('informationview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        kwargs['family'] = family
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.created = date.today()
        self.seen_count = 0
        self.like_count = 0
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        self.family = family
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        return super().form_valid(form)

class InformationEditView(LoginRequiredMixin, UpdateView):
    model = Information
    fields = ['title', 'info_text']
    template_name = 'informationedit.html'

    def get_success_url(self):
        return reverse_lazy('informationview', kwargs={'pk':
self.object.pk})

class InformationView (DetailView):
    template_name = 'information.html'
    model = Information

    def get_context_data(self, **kwargs):
        data = super(InformationView,
self).get_context_data(**kwargs)
        user = self.request.user
        memberparam = Member.objects.get(id = user.id)
        try:
            td = InformationSeen (information__id =
self.object.pk, member = memberparam)
        except InformationSeen.DoesNotExist:

```

```

        td = InformationSeen.objects.add(information__id
= self.object.pk, member = memberparam)
        td.save()
        self.object.seen_count = self.object.seen_count +
1
        return data

```

```

class InformationsListView(ListView):
    model = Information
    template_name = 'familyinformations.html'

    def get_queryset(self):
        familyparam = get_object_or_404(Family,
id=self.kwargs['family'])
        informations = Information.objects.filter(family =
familyparam)

        informations = informations.order_by('-created')
        return informations

    def get_context_data(self, **kwargs):
        famdata = super(InformationsListView,
self).get_context_data(**kwargs)
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        famdata ['family'] = family
        return famdata

class InformationLikeAddView(LoginRequiredMixin, CreateView):
    model = InformationLike
    template_name = 'informationlikeadd.html'

    def get_success_url(self):
        return reverse_lazy('informationview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        information = get_object_or_404(Information,
id=self.kwargs['information'])
        kwargs['information'] = information
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        information = get_object_or_404(Information,
id=self.kwargs['information'])
        self.information = information

```

```

        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        mes = InformationLike.objects.filter(information =
information)

        score = 0
        information.like_count = mes.count()
        information.save()
        return super().form_valid(form)

```

```

class InformationLikesListView(ListView):
    model = InformationLike
    template_name = 'informationlikess.html'

    def get_queryset(self):
        information = get_object_or_404(Information,
id=self.kwargs['information'])
        informationlikes =
InformationLike.objects.filter(information = information)
        informationlikes = informationlikes.order_by('-
created')

        return informationlikes

    def get_context_data(self, **kwargs):
        information = get_object_or_404(Information,
id=self.kwargs['information'])
        kwargs['information'] = information
        return super().get_context_data(**kwargs)

```

```

class MemberMessageAddView(LoginRequiredMixin, CreateView):
    model = MemberMessage
    template_name = 'membermessageadd.html'
    fields = ['message']

    def get_success_url(self):
        return reverse_lazy('membermessagesview',
kwargs={'pk': self.object.family.pk})

    def get_context_data(self, **kwargs):
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        kwargs['family'] = family
        return super().get_context_data(**kwargs)

    def form_valid(self, form):

```

```

        family = get_object_or_404(Family,
id=self.kwargs['family'])
        self.family = family
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        self.created = date.today()
        self.like_count = 0
        self.seen_count = 0
        return super().form_valid(form)

class MemberMessagesListView(ListView):
    model = MemberMessage
    template_name = 'familymembermessages.html'

    def get_queryset(self):
        familyparam = get_object_or_404(Family,
id=self.kwargs['family'])
        messages = MemberMessage.objects.filter(family =
familyparam)

        messages = messages.order_by('-created')
        return messages

    def get_context_data(self, **kwargs):
        famdata = super(MemberMessagesListView,
self).get_context_data(**kwargs)
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        famdata ['family'] = family
        return famdata

class MemberMessageLikeAddView(LoginRequiredMixin, CreateView):
    model = MemberMessageLike
    template_name = 'membermessagelikeadd.html'

    def get_success_url(self):
        return reverse_lazy('membermessagesview',
kwargs={'pk': self.object.membermessage.family.pk})

    def get_context_data(self, **kwargs):
        membermessage = get_object_or_404(MemberMessage,
id=self.kwargs['membermessage'])
        kwargs['membermessage'] = membermessage
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        membermessage = get_object_or_404(MemberMessage,
id=self.kwargs['membermessage'])
        self.membermessage = membermessage
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        mes = MemberMessageLike.objects.filter(membermessage =
membermessage)
        membermessage.like_count = mes.count()
        return super().form_valid(form)

class MemberMessageLikeDeleteView(DeleteView):
    model = MemberMessageLike
    template_name = 'membermessagelikeddelete.html'

class MeetupMessageAddView(LoginRequiredMixin, CreateView):
    model = MeetupMessage
    template_name = 'meetupmessageadd.html'
    fields = ['message']

    def get_success_url(self):
        return reverse_lazy('meetupview', kwargs={'pk':
self.object.meetup.pk})

    def get_context_data(self, **kwargs):
        meetup = get_object_or_404(Meetup,
id=self.kwargs['meetup'])
        kwargs ['meetup'] = meetup
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        meetup = get_object_or_404(Meetup,
id=self.kwargs['meetup'])
        self.meetup = meetup
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        self.created = date.today()
        return super().form_valid(form)

class TodolistMessageAddView(LoginRequiredMixin, CreateView):
    model = TodolistMessage
    template_name = 'todolistmessageadd.html'
    fields = ['message']

```

```

    def get_success_url(self):
        return reverse_lazy('todolistview', kwargs={'pk':
self.object.todolist.pk})

    def get_context_data(self, **kwargs):
        todolist = get_object_or_404(Todolist,
id=self.kwargs['todolist'])
        kwargs ['todolist'] = todolist
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        todolist = get_object_or_404(Todolist,
id=self.kwargs['todolist'])
        self.todolist = todolist
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        self.created = date.today()
        return super().form_valid(form)

class InformationMessageAddView(LoginRequiredMixin, CreateView):
    model = InformationMessage
    template_name = 'informationmessageadd.html'
    fields = ['message']

    def get_success_url(self):
        return reverse_lazy('informationview', kwargs={'pk':
self.object.information.pk})

    def get_context_data(self, **kwargs):
        information = get_object_or_404(Information,
id=self.kwargs['information'])
        kwargs ['information'] = information
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        information = get_object_or_404(Information,
id=self.kwargs['information'])
        self.information = information
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        self.created = date.today()
        return super().form_valid(form)

```

```

class VotingVoteAddView(LoginRequiredMixin, CreateView):
    model = VotingVote
    template_name = 'votingvoteadd.html'

    def get_success_url(self):
        return reverse_lazy('votingview', kwargs={'pk':
self.object.votingitem.voting.pk})

    def get_context_data(self, **kwargs):
        vitem = get_object_or_404(VotingItem,
id=self.kwargs['vitem'])
        kwargs['vitem'] = vitem
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        vitem = get_object_or_404(VotingItem,
id=self.kwargs['vitem'])
        self.votingitem = vitem
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        votes = VotingVote.objects.filter(votingitem__id =
vitem, member = member)
        vitem.vote_count = votes.count()
        return super().form_valid(form)

class VotingView (DetailView):
    template_name = 'voting.html'
    model = Voting

    def get_context_data(self, **kwargs):
        data = super(VotingView,
self).get_context_data(**kwargs)
        user = self.request.user
        memberparam = Member.objects.get(id = user.id)
        try:
            td = VotingSeen (voting__id = self.object.pk,
member = memberparam)
        except VotingSeen.DoesNotExist:
            td = VotingSeen.objects.add(voting__id =
self.object.pk, member = memberparam)
            td.save()
            self.object.seen_count = self.object.vote_count +
1

        return data

```

```

class VotingAddView(LoginRequiredMixin, CreateView):
    model = Voting
    fields = ['title', 'content', 'end_time']
    template_name = 'votingadd.html'

    def get_success_url(self):
        return reverse_lazy('votingview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        kwargs['family'] = family
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        self.family = family
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        self.created = date.today()
        return super().form_valid(form)

class VotingEditView(LoginRequiredMixin, UpdateView):
    model = Voting
    template_name = 'votingedit.html'

    def get_success_url(self):
        return reverse_lazy('votingview', kwargs={'pk':
self.object.pk})

class VotingItemEditView(LoginRequiredMixin, UpdateView):
    model = VotingItem
    template_name = 'votingitemedit.html'

    def get_success_url(self):
        return reverse_lazy('votingview', kwargs={'pk':
self.object.voting.pk})

class VotingItemAddView(LoginRequiredMixin, CreateView):
    model = VotingItem
    fields = ['item_text']
    template_name = 'votingitemadd.html'

```

```

    def get_success_url(self):
        return reverse_lazy('votingview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        voting = get_object_or_404(Voting,
id=self.kwargs['voting'])
        kwargs['voting'] = voting
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        voting = get_object_or_404(Voting,
id=self.kwargs['voting'])
        self.voting = voting
        self.vote_count = 0
        return super().form_valid(form)

class VotingsListView(ListView):
    model = Voting
    template_name = 'familyvotings.html'

    def get_queryset(self):
        familyparam = get_object_or_404(Family,
id=self.kwargs['family'])
        votings = Voting.objects.filter(family = familyparam)
        votings = votings.order_by('-created')
        return votings

    def get_context_data(self, **kwargs):
        famdata = super(VotingsListView,
self).get_context_data(**kwargs)
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        famdata ['family'] = family
        return famdata

class CurrentVotingsListView(ListView):
    model = Voting
    template_name = 'currfamilyvotings.html'

    def get_queryset(self):
        familyparam = get_object_or_404(Family,
id=self.kwargs['family'])
        votings = Voting.objects.filter(family = familyparam)
        dt = date.today()

```

```

    votings = votings.filter(end_time__gte = dt)
    votings = votings.order_by('-created')
    return votings

```

```

    def get_context_data(self, **kwargs):
        famdata = super(VotingsListView,
self).get_context_data(**kwargs)
        family = get_object_or_404(Family,
id=self.kwargs['family'])
        famdata ['family'] = family
        return famdata

```

```

class MyTodolistsListView(ListView):
    model = Todolist
    template_name = 'mytodos.html'

    def get_queryset(self):
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        todolists = Todolist.objects.filter(member = member)
        todolists = todolists.order_by('-created')
        return todolists

```

```

class MyMeetupsListView(ListView):
    model = Meetup
    template_name = 'mymeetups.html'

    def get_queryset(self):
        user = self.request.user
        member = Member(id = user.id)
        self.member = member
        meetups = Meetup.objects.filter(member = member)
        meetups = meetups.order_by('-created')
        return meetups

```

```

class MyInformationsListView(ListView):
    model = Information
    template_name = 'myinformations.html'

    def get_queryset(self):
        user = self.request.user
        member = Member(id = user.id)
        self.member = member

```

```
        informations = Information.objects.filter(member =  
member)  
        informations = informations.order_by('-created')  
        return informations  
  
class MyVotingsListView(ListView):  
    model = Voting  
    template_name = 'myvotings.html'  
  
    def get_queryset(self):  
        user = self.request.user  
        member = Member(id = user.id)  
        self.member = member  
        vorings = Voting.objects.filter(member = member)  
        vorings = vorings.order_by('-created')  
        return vorings
```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
кафедра програмних засобів

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПЛАНУВАННЯ СПРАВ У СІМ'Ї

Виконав
ст. гр. КНТ-138

О. А. Павліченко

Керівник
к.т.н., доц.

Т. В. Голуб

2022

Рисунок В.1 – Слайд № 1

Об'єктом роботи є процес планування справ у сім'ї.

Предметом роботи є програмне забезпечення планування справ у сім'ї.

Мета роботи – розробити програмне забезпечення планування справ у сім'ї для впорядкування спільного доступу до інформації в сім'ї.

Рисунок В.2 – Слайд № 2

Завдання

- дослідження проблеми розроблення програмного забезпечення планування справ у сім'ї;
- визначення вимог до розробки;
- проєктування програмного забезпечення планування справ у сім'ї;
- програмна реалізація програмного забезпечення планування справ у сім'ї.

3

Рисунок В.3 – Слайд № 3

Порівняння програмних аналогів			
Критерій	Cozi Family Organizer	Hub Family Organizer	Розробка
Планування зустрічей	Так	Так	Так
Створення списків справ	Так	Так	Так
Відправлення повідомлень	Ні	Так	Так
Голосування	Ні	Ні	Так
Наявність платних можливостей	Так	Так	Ні

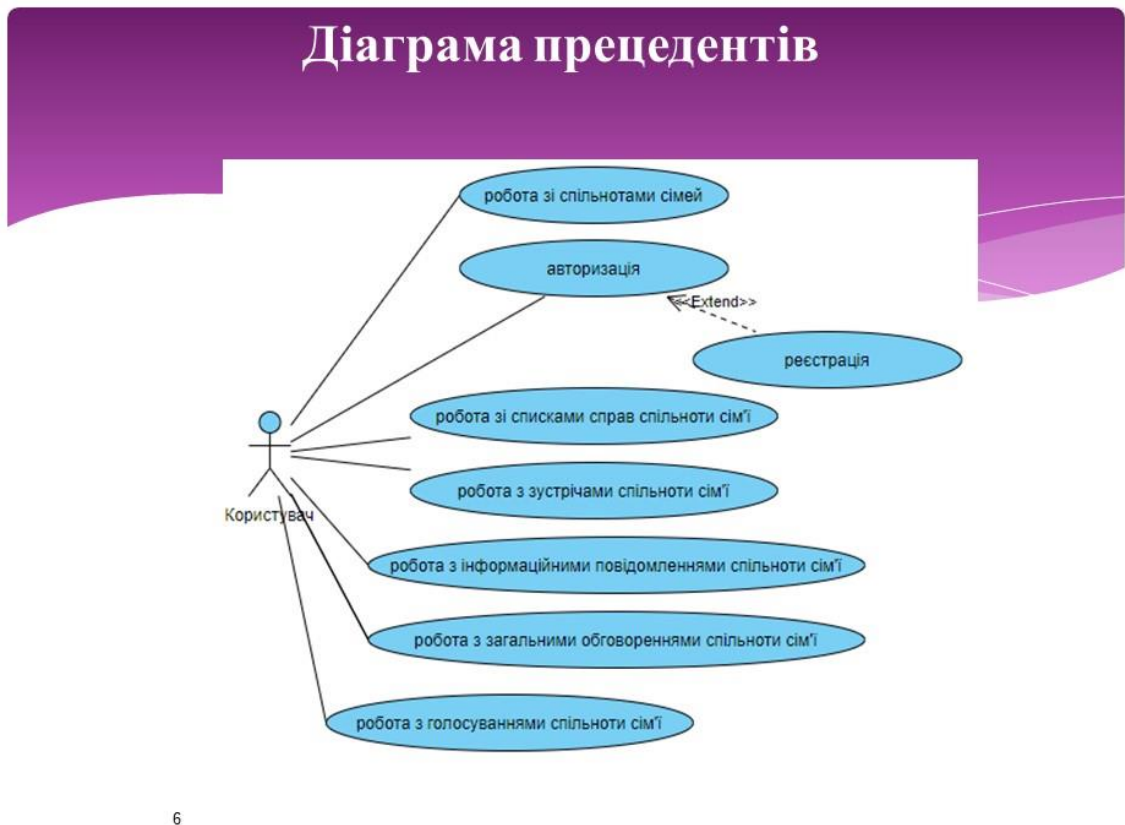
4

Рисунок В.4 – Слайд № 4

Вибір мови розробки програми		
Критерій	JavaScript	Python
Розробка серверної частини вебзастосунку	Node.js	Так, також доступні Django, Flask
Модель наслідування	На основі прототипів	На основі класів
Необхідність розробки меншої кількості коду	Ні	Так
Стабільність	Нижче	Вище
Кількість модулів для підключення в програмах	Менша	Більша

5

Рисунок В.5 – Слайд № 5



6

Рисунок В.6 – Слайд № 6

Схема бази даних

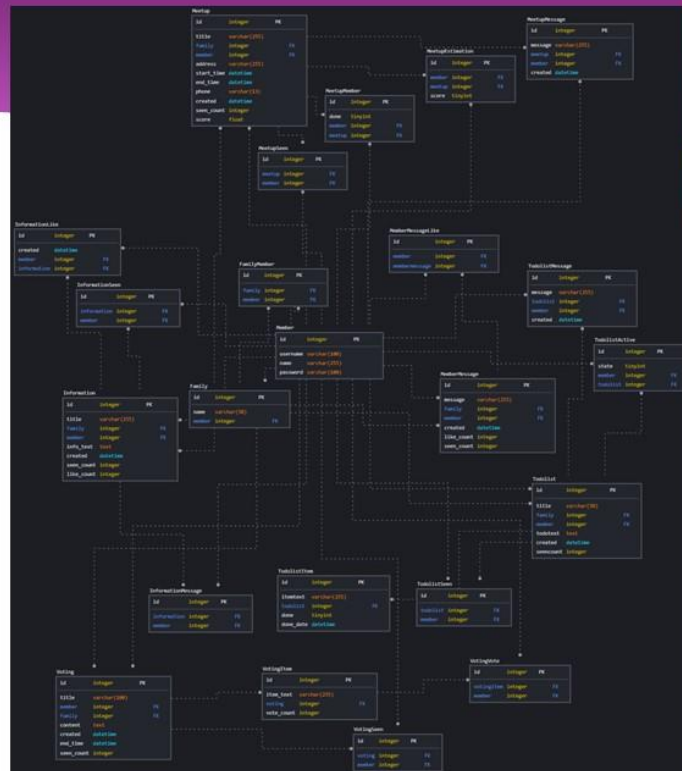


Рисунок В.7 – Слайд № 7

Схема функціонування програми

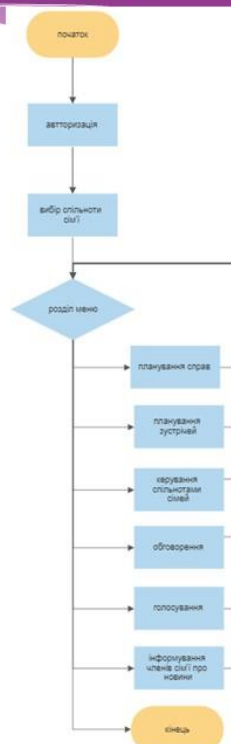


Рисунок В.8 – Слайд № 8

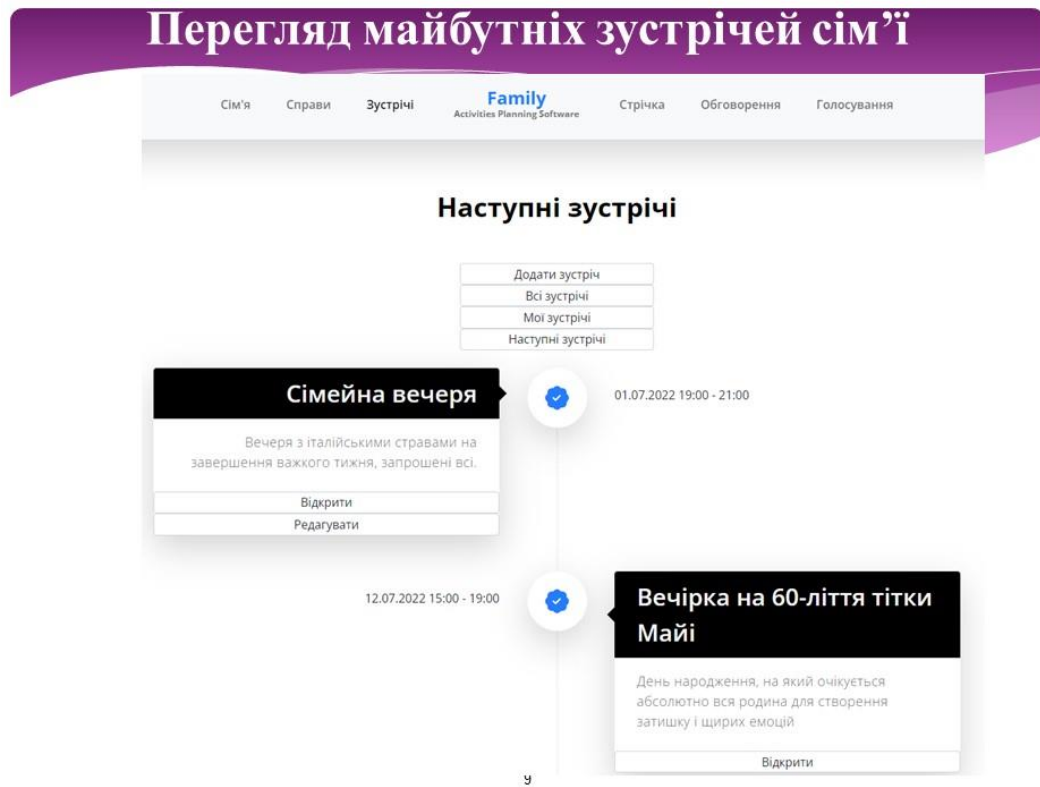


Рисунок В.9 – Слайд № 9

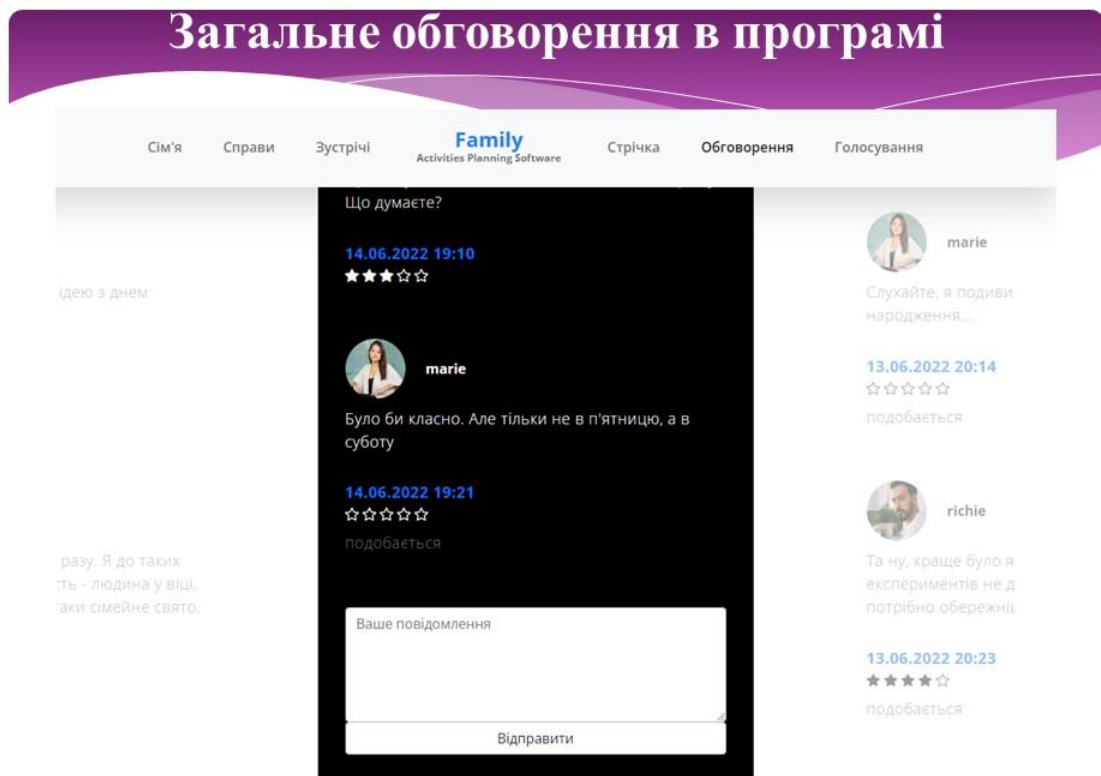


Рисунок В.10 – Слайд № 10

Голосування

Сім'я Справи Зустрічі **Family** Стрічка Обговорення Голосування
Activities Planning Software

Коли краще поїхати на море?

Вирішуємо організацію літнього відпочинку

Опитування триває до: 15.06.2022

Переглядів: 5 [Детальніше](#)

Створено: olena

Варіант вибору	Голосів	Відсоток	Дія
15 - 22 липня	3	75	ГОЛОСУВАТИ
27 липня - 3 серпня	1	25	ГОЛОСУВАТИ
27 липня - 3 серпня	0	0	ГОЛОСУВАТИ

11

Рисунок В.11 – Слайд № 11

Висновки

Під час проєктування програмного забезпечення виконано вибір мови програмування для реалізації програми на користь мови Python, виконано вибір функцій, які має забезпечувати програма, створено діаграму прецедентів, розроблено структуру бази даних, обрано систему керування базами даних MySQL для підтримки її роботи, описано створені таблиці та зв'язки між ними.

Реалізовано програму з забезпеченням всіх запланованих функцій на основі реалізації відповідних класів підтримки їх виконання на основі взаємодії з даними та підготовки цих даних для відображення на відповідних вебсторінках. Описано створені в процесі класи програми. Описано основні сценарії роботи користувача з програмою за допомогою відповідних інтерфейсів. Представлено можливості використання програми.

Розроблена в результаті виконання роботи програма призначена для планування і обговорення справ у сім'ї, розглядаючи сім'ю як деяку спільноту, яка потребує впорядкованого, але обмеженого доступу до інформації про справи. Поставлені спочатку завдання дипломної роботи в підсумку були успішно виконані.

12

Рисунок В.12 – Слайд № 12