МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

ПРОГРАМНИЙ ДОДАТОК "ПЕРСОНАЛЬНИЙ ОРГАНАЙЗЕР"

Здобувач освіти гр. ІН – 82 Денис БУБЛИК

Науковий керівник,
кандидат фізико-математичних наук,
асистент кафедри комп'ютерних наук Олександр ВЛАСЕНКО

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи

здобувача вищої освіти четвертого курсу, групи ІН-82 спеціальності «122 – Комп'ютерні науки» денної форми навчання Бублика Дениса Руслановича.

Тема: «ПРОГРАМНИЙ ДОДАТОК "ПЕРСОНАЛЬНИЙ ОРГАНАЙЗЕР"»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) Літературний огляд за обраною тематикою роботи. 2) Методика вирішення поставлених задач. 3) Практична реалізація.

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____Олександр ВЛАСЕНКО
Завдання прийняв до виконання _____ Денис БУБЛИК

РЕФЕРАТ

Записка: 79 стор., 24 рис., 2 табл., 2 додатки, 13 джерел.

Об’єкт дослідження – процес проектування та реалізації програмного органайзеру для власних потреб.

Мета роботи – розробка програмного додатку-органайзеру для зберігання, додавання, видалення, перегляду задач та створення на їх основі календаря з власним графічним інтерфейсом.

Методи дослідження – вивчення літератури, розгляд тематичних джерел, порівняльний аналіз сучасних аналогів.

Результати – розроблено програмний додаток-органайзер, котрий має методи зберігання, обробки, зміни, оптимізації задач, функціональний графічний інтерфейс та створення на основі задач календарю. Продукт реалізований за допомогою об’єктно-орієнтовної мови програмування Java та платформи JavaFX, яка відповідає за графічні компоненти.

ОГРАНАЙЗЕР, ДОДАТОК, UI/UX-ДИЗАЙН, JAVA, JAVAFX

ЗМІСТ

ВСТУП

Актуальність роботи. Кожен з нас щодня виконує будь-яку діяльність, яка потребує вільного часу, який необхідно планувати заздалегідь. На день всі без виключення мають безліч різноманітних справ, які повинні зробити, і як в цьому всьому не заплутатися, та ще гірше, не забути про одну або декілька справ? Тому люди вирішили «керувати» своїм часом: записувати справи, нотувати їх дату та час виконання, прогнозувати скільки займе часу та чи інша справа, щоб не тримати це в голові та сконцентруватися на справах. Цим «керування» займаються органайзери. Вони зберігають дату та час справ, дають можливість продивиться поточний список всіх справ, попередять завчасно. Також мають можливість зручно та ефективно, в рамках часу, розподілити справи.

Розподіл свого часу на зараз є нелегким та важливою справою для сучасної людини, тому що як-ніяк, але все це веде до секрету успіху багатьох видатних людей. З плином часу все більше і більше розвиваються технології, тому потрібно більше і більше чого про них знати, щоб знати, для чого воно, як воно працює, так як його зробити. А це в свою чергу більше забирає часу на опанування цієї купи знать, тому важливо сьогодні вміти правильно користуватися своїм часом, щоб встигнути все зробити та ще відпочити.

Актуальність ефективного розподілу часу на зараз велика, всі намагаються якомога більше працювати, вчити, тренуватись аби бути успішним, при достатку, розумним, здоровим, тому люди користуються органайзерами. Вони дозволяють людини жити зручніше та ефективніше: звільняють мозок людини від запам’ятовування, які так коли в неї справи завтра \ через тиждень \ через місяць; будувати плани наперед, не побоюючись пропустити важливу зустріч або не завершити роботу над проектом вчасно.

Тому **метою кваліфікаційної роботи є** розробка та створення програмного додаток «Персональний органайзер» для «організації» щоденних справ користувача.

1. ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

1.1. Аналіз предметної області

Створення органайзеру у вигляді програмного додатку є зручним та сучасним рішенням.

Додаток – це комп’ютерна програма, котра виконую певну функцію безпосередньо для користувача. Для його роботи потрібна ОС (операційна система) і системне програмне забезпечення, де додаток використовує їх можливості, функції та концепції. Він має власний інтерфейс взаємодії та власну логіку виконання певних операцій, які розробник створив. Може бути як автономним так і комплексним, тобто складатися з декількох додатків.

У більшості користувачів при собі завжди смартфони, багато людей працюють або зв’язані з персональними комп’ютерами, де хто працює на більш мобільних пристроях, такі як ноутбуки або планшети. Це дає можливість завжди мати під рукою органайзер.

Програма органайзер – це комп'ютерна програма, призначена для накопичення інформації, а потім оперативного пошуку по її, організації справ і контролю за їх виконанням, відстеження певних користувачем подій. [1].

Основними функціями органайзеру є календар, список справ, котрі мають свої дату та час та менеджер контактів, але остання функція в додатку буде неактуальною на сьогодні через те, що в кожному смартфоні є своя телефона книга, там зручніше зберігати контакти, адже все буде в одному місці та одразу можна і зателефонувати. Типові функції органайзеру складаються, окрім вище згаданих: можливість нотувати велику за обсягом інформацію, планувальник контролю завдань за їх виконанням та завчасне оповіщення про початок виконання певних справ користувача. Деякі органайзери можуть як не мати деяких із перерахованих функцій, так і надавати свої, додаткові функції.

1.2. Аналіз аналогів

Так як актуальність в тайм-менеджменті дуже висока, і з плином часу вона тільки и буде підвищуватись, то існую багато різних органайзерів, як за наповненням функціональності, так і привабливістю та зручністю інтерфейсу взаємодії з додатком. Окрім цього додатки можуть бути на різних платформах та різних операційних системах. Тому вибір органайзеру досить таки важка справа, що аж очі розбігаються.

Перед тим, як розпочати розробку програмного органайзера, потрібно було визначитися з платформою під яку буде створено додаток – це ПК (персональний комп’ютер), а далі шукати та досліджувати аналоги під цю платформу.

Було обрано такі аналоги як: «Todoist» та «MyLifeOrganized».

«Todoist» – менеджер завдань і програма для списку справ з привабливим та лаконічним інтерфейсом (рис. 1.1).

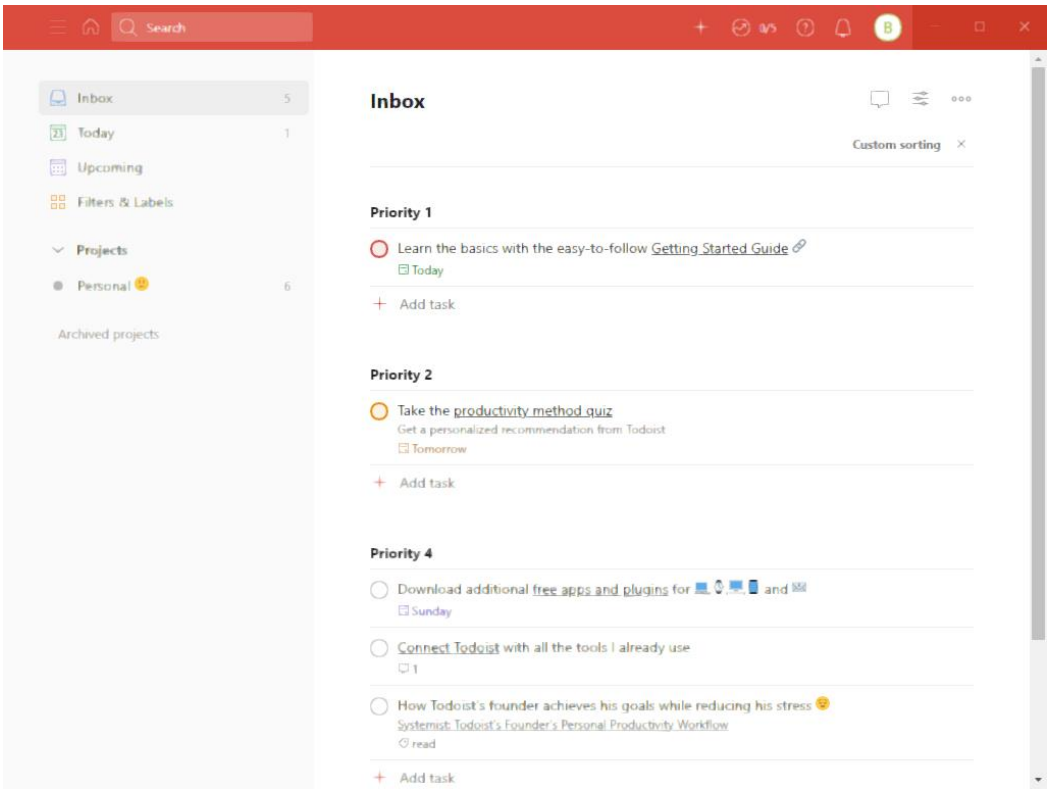

Рисунок 1.1 – Інтерфейс додатку «Todoist»

Даним аналогом можна користуватися майже будь-де, хоч на ПК для macOS, Windows 10/11, Linux; на смартфоні під IOS/Android; в браузері з використанням потрібного розширення та навіть на смарт-годинниках. Додаток дає змогу інтегруватися з іншими програмами, які користувач використовував раніше, аби перенести нотатки, справі тощо, і зробити його організованим центром для виконання всіх завдань. Даний органайзер із типових функцій має: календар, список справ за датою та часом та нагадування о початку справ (рис. 1.2).

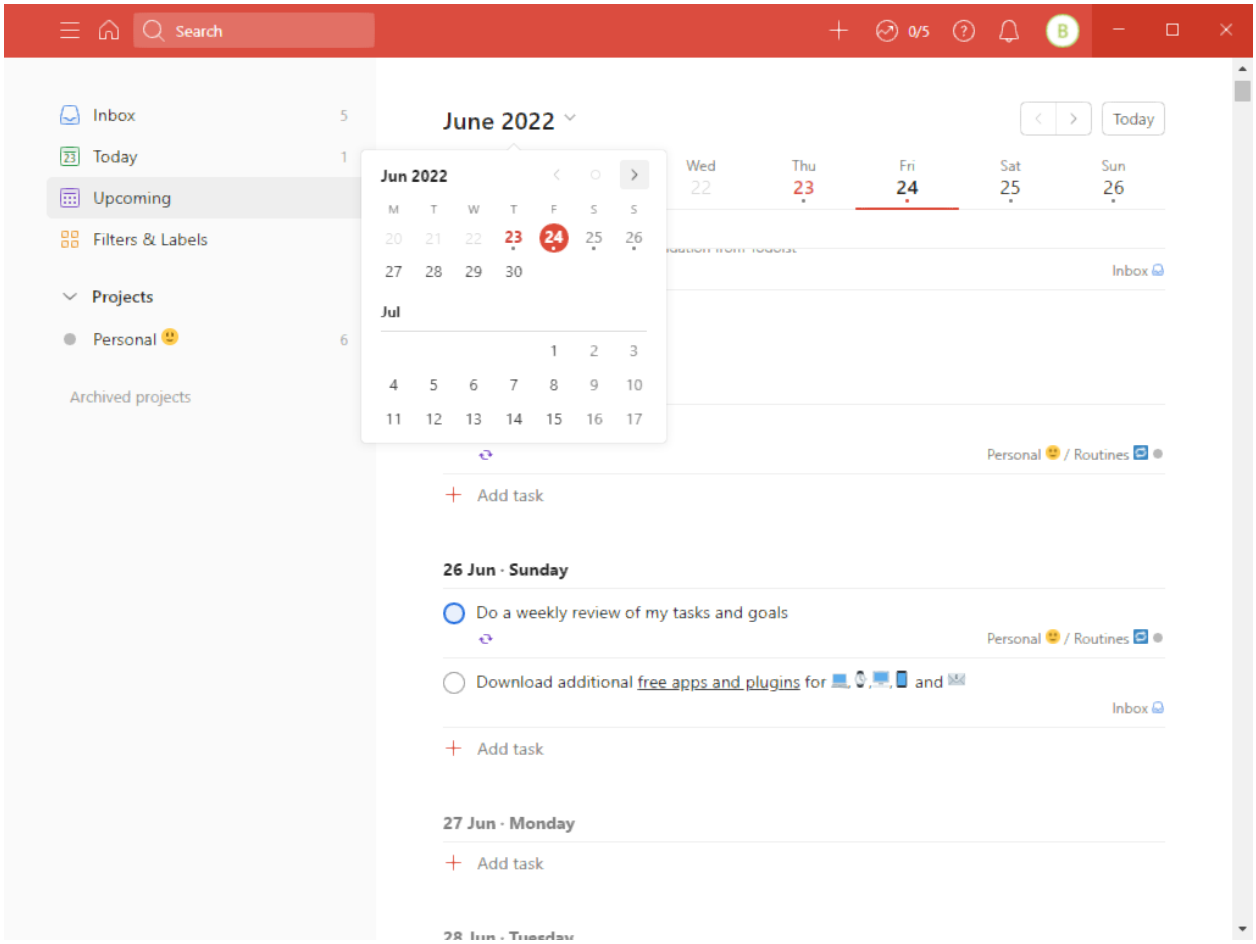

Рисунок 1.2 – Календар додатку «Todoist»

Із особливостей програми можна відмітити [2]:

- додавання задачі в один клік;
- повторювані терміни задач аби не забувати про них та виробляти звички;

- розділи на підзадачі, щоб зручно всі завдання були організовані;
- рівні пріоритету для виділення найважливіших справ;
- делегувати завдання іншим людям аби звільнити для себе час;
- сповіщати вас, коли люди залишають коментарі, виконують завдання тощо;
- лейбли, які допоможуть знайти потрібну групу справ;
- фільтри, які надають вам переглядати справи на основі термінів виконання, особи, яка відповідає за завдання, пріоритетності тощо;
- коментарі та завантаження файлів (включаючи голосові нотатки);
- візуалізація продуктивності, що дає уявлення про виконану роботу.

«Todoist» має шаблони справ (рис. 1.3), які необхідні для швидкого початку користуванням органайзером або для ефективного виконання тієї чи іншої справи, як наприклад: написання книги, вивчення певної мови тощо.

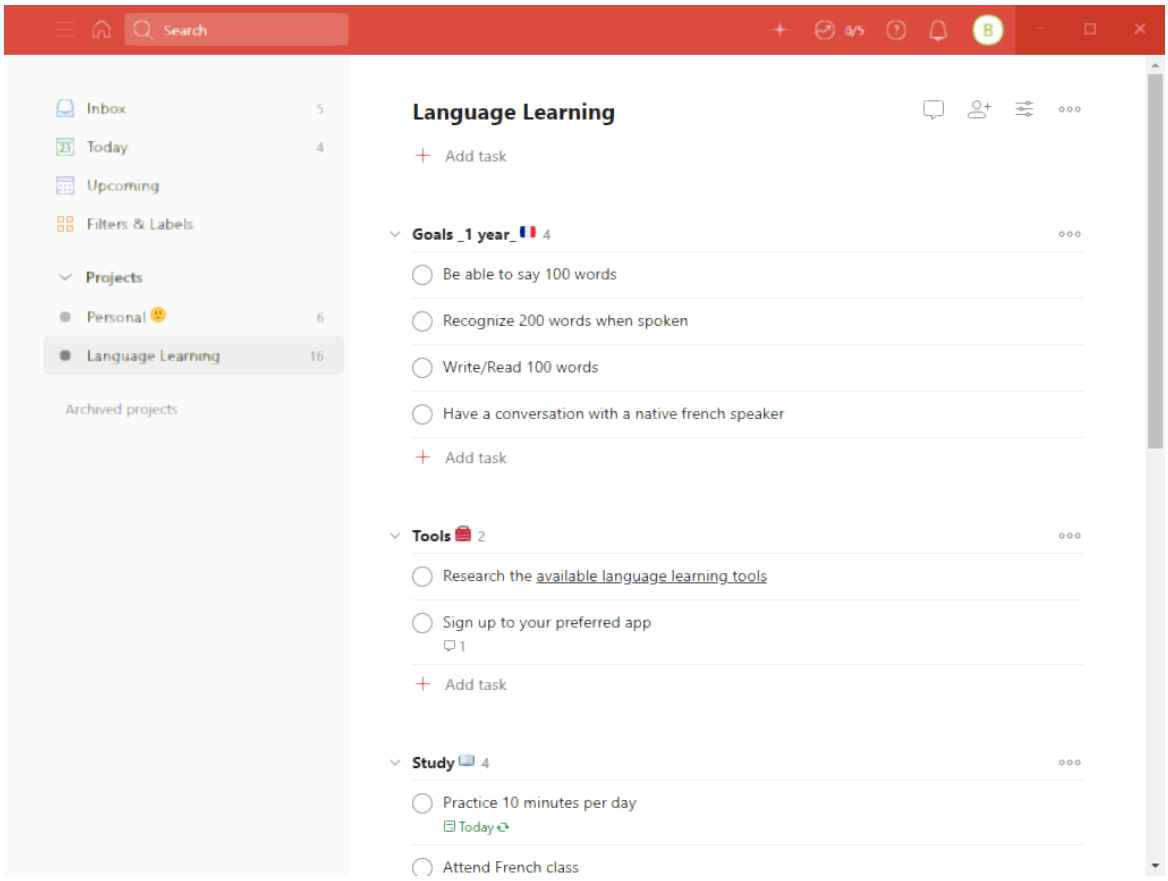

Рисунок 1.3 – Шаблон «Language Learning» додатку «Todoist»

Вони мають заздалегідь підготовленні пункти справ та специфічну для цього налаштування додатку. Кількість таких шаблонів обмежена, не під всі задачі можна знайти, але додаток дає можливість створювати власні шаблони та ділитись ним з іншими.

Додаток «MyLifeOrganized» має функції списку справ за датою та часом та сповіщення о задачі. Інших типових функцію, на жаль, не має. Він також є кросплатформним як і попередній додаток, але не така велика різноманітність, можна користуватися на ПК під Windows, macOS та на смартфоні під IOS/Android. Має нагромаджений і дещо «тяжкий» для розуміння інтерфейс (рис 1.4).

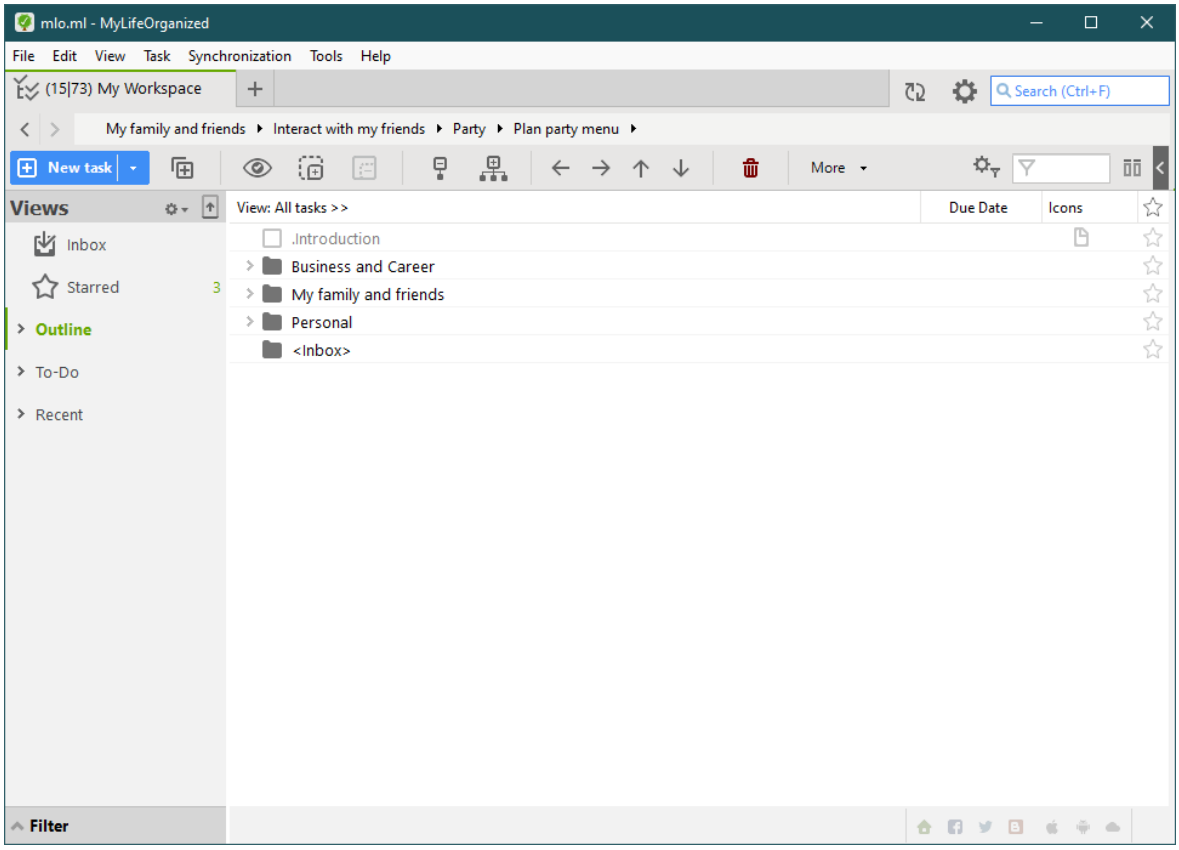

Рисунок 1.4 – Інтерфейс додатку «MyLifeOrganized»

Можна відмити декілька особливостей. Створенні ієрархій, що дозволяє розбити задачу на підзадачі, і ці підзадачі, в свою чергу, на інші підзадачі і додавати між задачами залежність, щоб краще контролювати хід виконання (рис. 1.5).

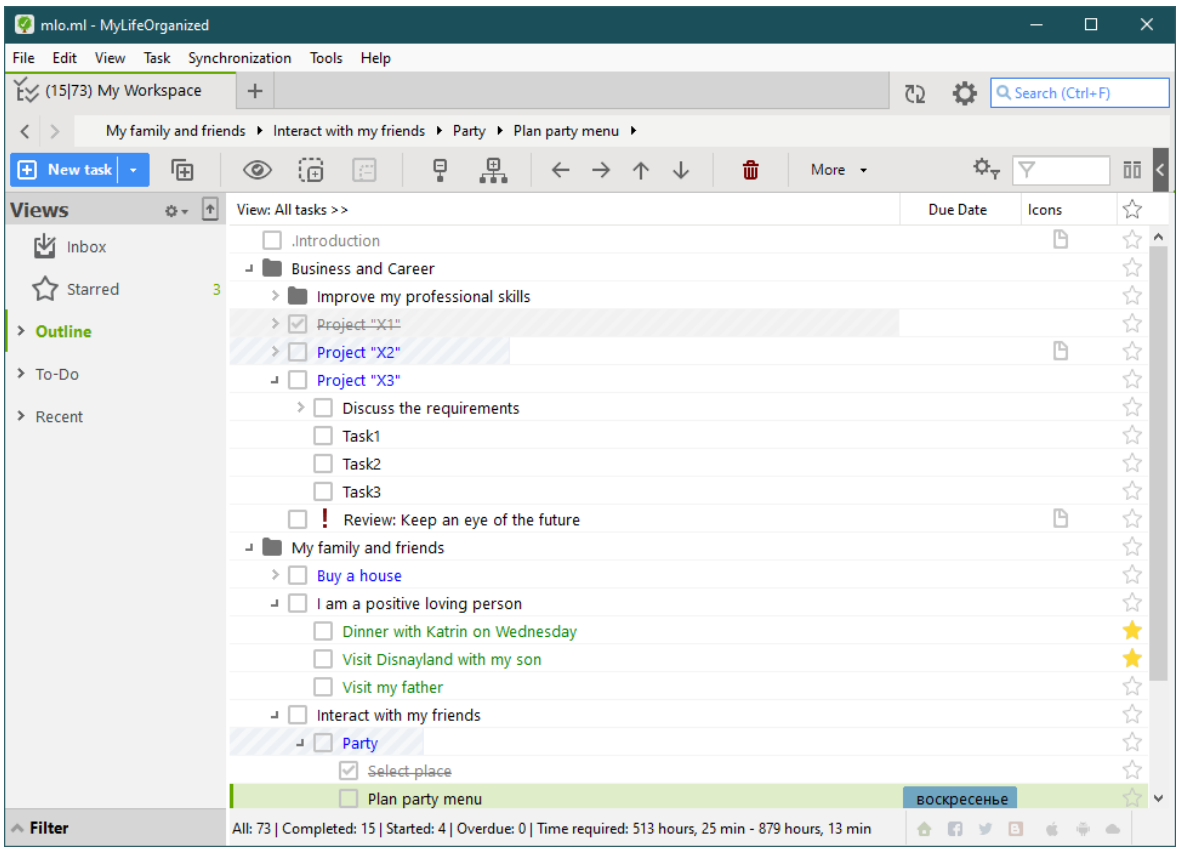

Рисунок 1.5 – Ієрархія задач в додатку «MyLifeOrganized»

Здатність перетворити ієрархію задач на звичайний простий список.

Функція «нагадування за місцем» дозволяє отримати інформацію в потрібному місці. Додаток розроблений для надсилання розумних нагадувань, коли ви прибули в необхідне місце, наприклад, ви біля супермаркету, органайзер може надіслати сповіщення зі списком покупок, як тільки ви дісталися додому, нагадування зі списком невідкладних справ уже чекатиме вас на екрані смартфона.

Синхронізація через хмару дозволяє синхронізувати всі пристрої на яких встановлений цей додаток. Таким чином, всі списки та справи завжди знаходяться в актуальному стані.

Створення завдання відправивши лист на e-mail адрес, який потрібно зазначити в програмі. У додатку можна буде перетворити список задач на конкретну дію, ієрархію та інше [3].

Таблиця 1.1 – Порівняльний аналіз додатків

Критерії	Todoist	MyLifeOrganized
Наявність календаря	+	-
Списки справ з датою та часом	+	+
Мультиплатформеність	+	+
Зручний та зрозумілий інтерфейс	+	-
Синхронізація з іншими пристроями	-	+
Інтеграція з іншими програмами	+	-

Проаналізувавши аналоги органайзерів можна відмітити, що мультиплатформеність та задачі з датою та часом є головними вимогами до програми, на них потрібно звернути особливу увагу.

1.3. Визначення середовища розробки

На сьогодні існує багато різних способів створення десктопного додатку, починаючи від великої кількості мов програмування для розробки логіки програми, закінчуючи різними інструментами для розробки графічного інтерфейсу. І для розробки потрібно визначитись з мовою програмування та інструментом для створення графічного інтерфейсу.

При виконанні бакалаврської роботи було використано мову програмування Java та інструмент для графічного інтерфейсу JavaFX.

Java – це об’єктно-орієнтована мова програмування яка працює на більшості відомих операційних систем, включаючи Windows, macOS та Linux. Мова є адаптацією мов C/C++, тому що має дуже схожий синтаксис та об’єктну модель. Java пропонує простий синтаксис кодування разом із вдосконаленим управлінням пам’яттю та збиранням сміття [4]. В реалізації

Java-програма компілюються у байт-код, який при виконанні інтерпретується JVM (віртуальною машиною Java) для конкретної платформи, що забезпечує мультиплатформеність додатку. Тобто мається на увазі те, що програма, написана на мові Java, працюватиме на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції [5]. Аби цього досягти, початковий код компілюється у байт-код, який є машинним та незалежним кодом низького рівня, що виконується інтерпретатором. Потім байт-код за допомогою віртуальної машини Java інтерпретується у код, який пристосований до певної операційної системи та процесора. Java є об’єктно-орієнтовною мовою через те, що всі дані та методи групуються в класи об’єктів, тобто для отримання даних чи виконання будь-якої дії, яка знаходиться в цьому об’єкті, потрібно створювати об’єкт. Однією із гарною особливістю мови Java, яка допоможе оптимізувати роботу додатка, є автоматичний збирач сміття (GC – Garbage Collector) – «він керує пам'яттю під час життєвого циклу об'єкта. Програміст вирішує, коли створювати об'єкти, а віртуальна машина відповідальна за звільнення пам'яті після того, як об'єкт стає непотрібним. Коли до певного об'єкта вже не залишається посилань, збирач сміття може автоматично прибирати його із пам'яті.».

JavaFX – це інструментарій для створення мультиплатформених графічних додатків на основі Java. За допомогою JavaFX можна створювати програми для різних операційних систем: Windows, macOS, Linux, Android, IOS і для різних пристроїв: десктопи, смартфони, планшети, вбудовані пристрої. Додаток на JavaFX буде працювати скрізь, де встановлено середовище Java (JRE). Інструмент надає великі можливості порівняно з низкою інших подібних платформ, зокрема, порівняно зі Swing. Це і великий набір елементів управління, і можливості роботи з мультимедіа, двовимірною та тривимірною графікою, декларативний спосіб опису інтерфейсу за допомогою мови розмітки FXML, можливість стилізації інтерфейсу за допомогою CSS, інтеграція зі Swing та багато іншого [6].

Вибір був зроблений на користь Java та JavaFX за рахунок того, що вони зроблені на одній і тій же платформі, що дає змогу їх використовувати, не хвилюючись, що щось може конфліктувати або бути не сумісним; мають мультиплатформеність за рахунок своєрідної структури обробки коду, що є важливим фактором в розробці органайзеру; простота та майже ідентичність синтаксису, тому що спільна основа; великий набір елементів управління та можливість стилізації інтерфейсу за допомогою CSS, що дає можливість створити зручний, привабливий та унікальний графічний інтерфейс.

1.4. Постановка задачі

Метою виконання кваліфікаційної роботи є проектування і розробка програмного додатку-органайзеру для зберігання, маніпулювання задачами з датою та часом, який має певну логіку роботи та власний графічний інтерфейс.

1. Проектування програмного додатку

Створення «Communication Diagram», яка буде демонструвати, які дії може виконувати користувач із програмою і до чого це призведе. Створення «Class Diagram», яка буде відображати архітектуру додатку, системи, підсистеми та модулі, способи взаємодії між ними.

2. Створення внутрішньої логіки роботи додатку

Маючи приблизну архітектуру додатку розробити потрібну логіку, яка буде мати такий функціонал:

- зберігати задачу з датою та часом;
- обирати тип задачі, тобто одноразова або повторювальна;
- модифікувати вже готову задачу;
- видаляти задачі;
- переглядати задачі;
- переглядати календар задач з певним інтервалом у часі;
- сповіщати заздалегідь о початку задачі;
- зберігати весь список задач у файлі;

- імпортувати в програму та експортувати в json-файл список задач;
- виведення зрозумілих та змістовних попередження та помилок у разі неправильного користування програмою;
- оптимізація швидкості та ресурсоємності програми за рахунок створення власних об’єктів по керуванню та зберіганню задач.

3. Створення графічного інтерфейсу додатку

Потрібно розробити приблизний макет, як буде виглядати додаток, обрати певний стиль та кольорову палітру і далі починати створення UI\UX дизайну додатку:

- зручно та доступно розмістити всі елементи інтерфейсу;
- зв’язати всі функціональні елементи інтерфейсу з логікою програми;
- зробити валідацію всіх полів аби користувач не зміг ввести некоректні дані, або зламати програму;
- наповнення всіх вікон попередження та помилок зрозумілим текстом, щоб користувач міг зрозуміти, де він помилився;
- налаштування виведення інформації у зрозумілому вигляді.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

Перед початком створення продукту, в нашому випадку десктопного додатку, потрібно визначитись зі структурою взаємодії з програмою, структурою моделі програми та шаблоном проектування. Ці основні компоненти при розробці додатку задають йому вектор подальшого розвитку та допоможуть легко розібратися що за що відповідає, а також спрощує подальшу його підтримку.

2.1. Проектування інформаційної системи

Communication Diagram. Проектуючи будь-який додаток, в першу чергу слід пам’ятати що він є орієнтованим на користувача, він буде мати доступ до основного функціоналу програми та подальшого прийняття рішень, що буде виконувати система. Опираючись на все це була розроблена діаграма зв’язку (Communication Diagram), яка знаходиться на рисунку 2.1, для демонстрації того, яким функціоналом напряму може керувати користувач. Суцільними лініями на діаграмі позначені дії, котрі виконує сам користувач, а пунктирними лініями – дії, що додаток виконує у фоні без прямого впливу користувача, котрі йдуть після дій користувача аби тримати дані в системі в актуальному стані.

Діаграма зв'язку – це свого роду діаграма взаємодії, котра показує, як взаємодіють об'єкти між собою. Це розширення діаграми об'єктів, яка показує об'єкти разом з повідомленнями (діями), які передаються від одного до іншого об’єкту. Призначення комунікаційної діаграми:

- модель передачі повідомлень між об’єктами або ролями, які забезпечують функціональні можливості операцій;
- модельні механізми в рамках архітектурного проектування системи;
- демонстрація взаємодії, які показують передані повідомлення між об’єктами та ролями в рамках сценарію співпраці;

- моделювання альтернативних сценаріїв в рамках операцій, які передбачають співпрацю різних об’єктів і взаємодій;
- підтримка ідентифікації об'єктів та їх та операцій (повідомлень) [7].

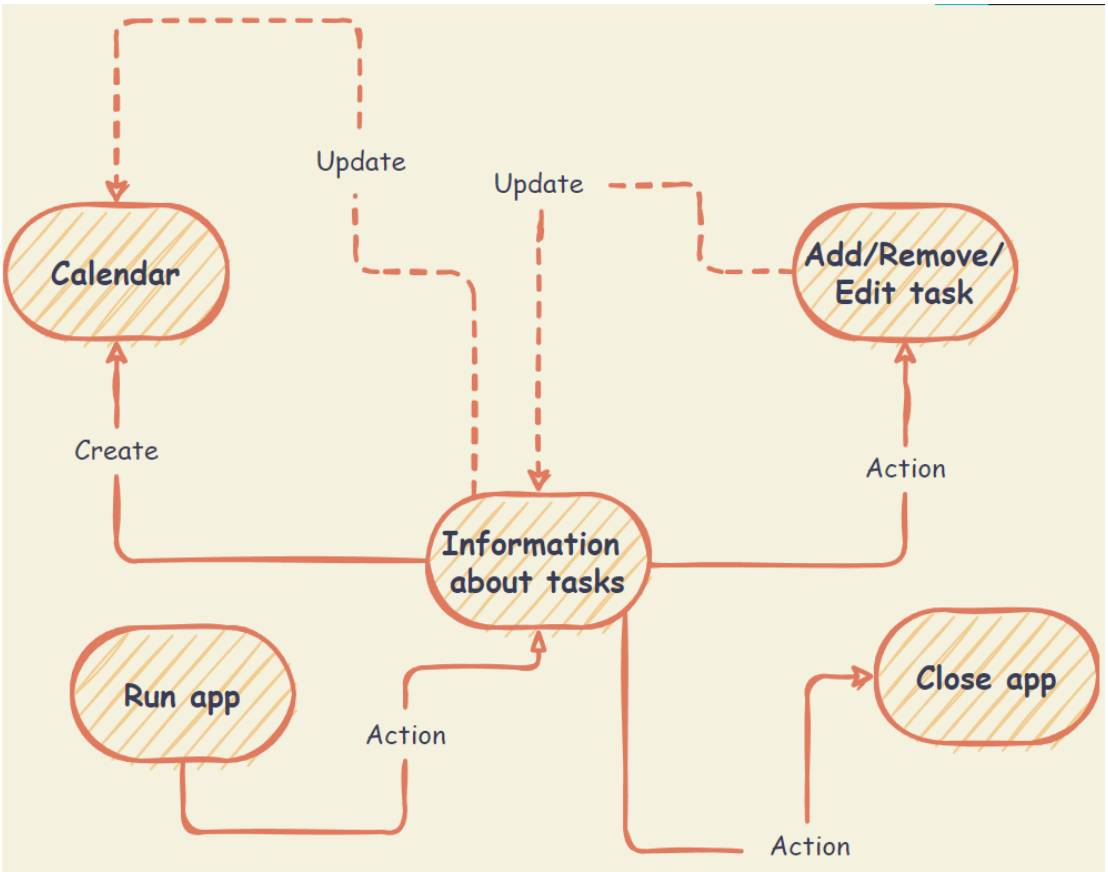

Рисунок 2.1 – Діаграма зв’язку користувача та додатку

Умовні позначення в діаграмах зв’язку. Символи та позначення, що використовуються в діаграмах зв'язку:

- прямокутники представляють об’єкти, які складають програму;
- лінії між екземплярами класів представляють відносини між різними частинами програми;
- стрілки представляють повідомлення, які передаються між об'єктами;
- нумерація дає змогу знати, в якому порядку надсилаються повідомлення та скільки повідомлень потрібно для завершення процесу.

Class Diagram. Для того щоб мати представлення о структурі моделі додатку, було розроблено приблизну діаграму класів (Class Diagram), яка

представлена на рисунку 2.2. Вона містить в собі класи, їх взаємодії між ними та їхні методи. В процесі розробки додатку структура та вміст діаграми може змінитися.

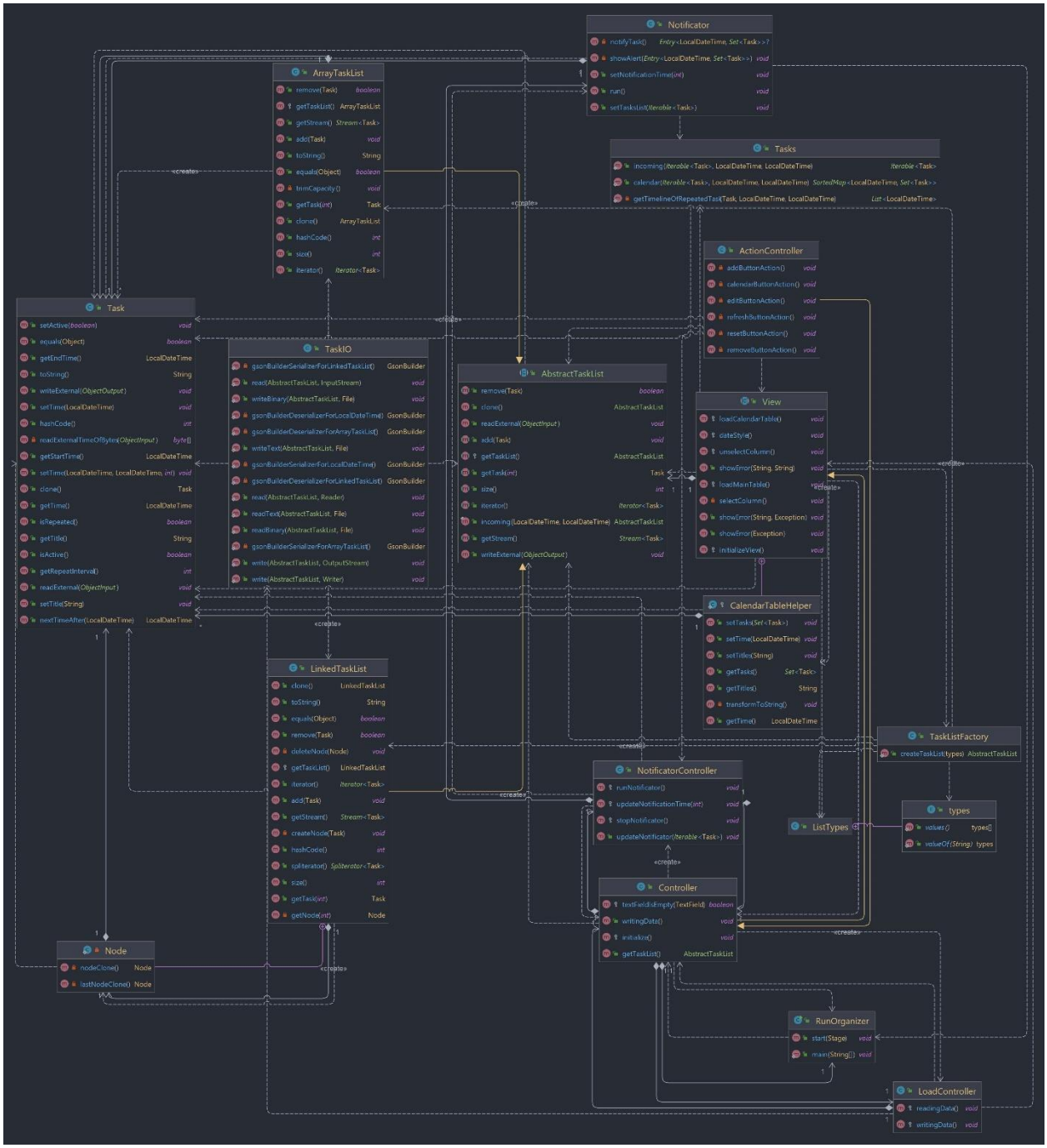

Рисунок 2.2 – Діаграма класів

Діаграма класів – це діаграма, яка представляє статичне відображення структури програми. Діаграма класів використовується не тільки для

візуалізації, опису та документування різних аспектів системи, але й для побудови виконуваного коду програмного додатка. Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування [8]. Діаграма класів описує атрибути та операції класу, а також обмеження, накладені на систему. Вона широко використовуються при моделюванні об'єктно-орієнтованих систем, оскільки вони є єдиними діаграмами UML, які можна відобразити безпосередньо з об'єктно-орієнтованими мовами. Метою діаграми є моделювання статичного вигляду програми. Це єдині діаграми, які можна безпосередньо відобразити за допомогою об’єктно-орієнтованих мов і, таким чином, вони широко використовуються під час проектування програми.

На діаграмі класи представлені в прямокутниках, що можуть містити три компоненти:

- у верхній частині написано ім'я класу. Ім'я класу вирівнюється по центру і пишеться напівжирним шрифтом, починаються з великої літери. Якщо клас абстрактний – його ім'я пишеться напівжирним курсивом;
- посередині розташовуються поля (атрибути) класу. Вони вирівняні по лівому краю і починаються з маленької літери;
- нижня частина містить методи класу. Вони також вирівняні по лівому краю і пишуться з маленької літери [9].

Взаємозв'язок - це особливий тип логічних відносин між сутностями, показаних на діаграмах класів та об'єктів. Діаграма класів представлена трьома типами зв’язків як:

- асоціація (Association);
- наслідування (Generalization);
- залежність (Dependency).

Асоціація: описує статичний або фізичний зв'язок між двома або більше об'єктами. Він показує, скільки об’єктів є у відносинах. Приклад представлений на рисунку 2.3, де відділ асоціюється з коледжем.

Рисунок 2.3 – Зв’язок асоціації

Узагальнення (наслідування) – це зв’язок між батьківським класом (суперкласом) і дочірнім класом (підкласом). Дочірній клас успадковується від батьківського класу. Приклад представлений на рисунку 2.4, де поточний рахунок, ощадний рахунок і кредитний рахунок є узагальненою формою банківського рахунку.

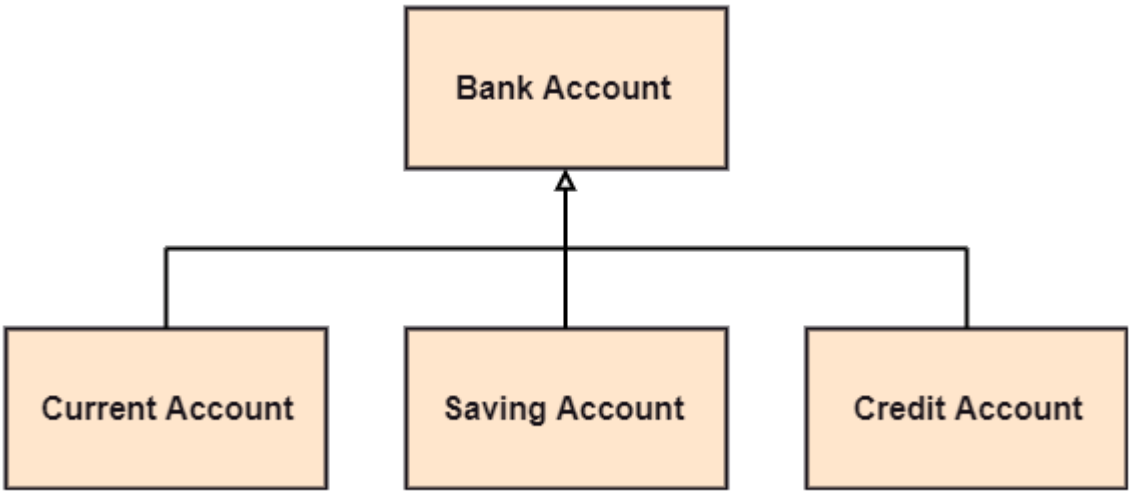

Рисунок 2.4 – Зв’язок наслідування

Залежність – це семантичний зв’язок між двома або більше класами, коли зміна в одному класі викликає зміни в іншому класі. Це формує більш слабкі відносини. Приклад представлений на рисунку 2.5, де ім’я студента залежить від ідентифікаційного номеру студента.

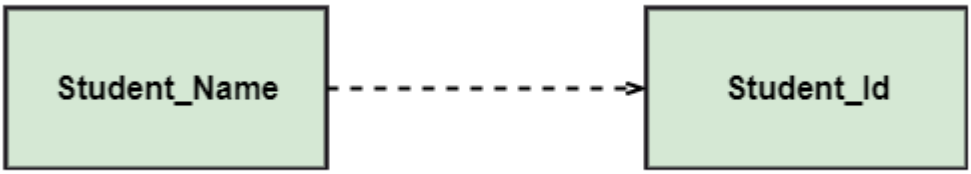

Рисунок 2.5 – Зв’язок залежності

Кратність – визначає певний діапазон допустимих екземплярів атрибутів. Якщо діапазон не вказано, кратність за замовчуванням розглядається як одиниця. Приклад представлений на рисунку 2.6, де кілька пацієнтів госпіталізуються в одну лікарню.

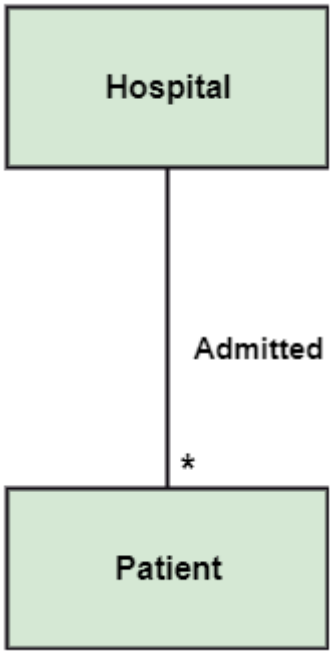

Рисунок 2.6 – Кратність

Агрегація – це підмножина асоціацій, яка представляє зв’язок. Це конкретніше, ніж асоціація. У таких відносинах дочірній клас може існувати незалежно від свого батьківського класу. Приклад представлений на рисунку 2.7, де компанія охоплює декілька співробітників, і навіть якщо один працівник звільняється, компанія все одно існує.

Рисунок 2.7 – Агрегація

Композиція є підмножиною агрегації. Вона зображує залежність між батьківським і його дочірнім класом, що означає, що якщо одна частина видаляється, то інша частина також. Приклад представлений на рисунку 2.8, де контактна книга складається з кількох контактів, і якщо ви видалите книгу, усі контакти також буде втрачено [10].

Рисунок 2.8 – Композиція

2.2. Шаблон проєктування

При розробці комплексних програм використовують шаблон (патерн) проєктування, ідея використання їх полягає в полегшенні виправлення помилок на этапі написання коду, а також організації логіки роботи в програмі. Шаблон проєктування – це ефективний спосіб рішення проблем, що часто зустрічаються при проектуванні програми. Він не є готовим прикладом, у який можна просто вставити в код програми, найчастіше це зразок вирішення проблеми, який відображає взаємозв’язок між об’єктами, без уточнення на те, як саме буде реалізований цей зв’язок. Патерни часто плутають з алгоритмами, адже обидва поняття описують типові рішення певних відомих проблем. Але якщо алгоритм – це точний набір дій, то шаблон – це високорівневий опис рішення, реалізація якого може відрізнятись у двох різних програмах. Патерн

допомагає зменшити витрати часу на розробку програми, так як вже є готові рішення, які потрібно просто використати, а не витрачати час та сили створення чогось «нового» та заплутаного, и навряд чи «нове» буде спроможне вирішити всі проблемі додатку та буде ефективно працювати. Також він стандартизує код програми та підвищує роботу в команді, так як ви не будете витрачати купу часу, аби пояснити як працює одна із систем додатку, можна буде просто назвати назву патерна проектування.

Класифікація шаблонів відрізняється за рівнем складності, деталізацією та охопленням проектованої системи. Низькорівневі та прості шаблони називаються ідіомами. Вони не є універсальними, оскільки застосовні лише в рамках однієї мови програмування. Універсальні – архітектурні шаблони, які можна реалізувати практично для будь-якої мови. Вони необхідні для проектування всієї програми, а не окремих її елементів. Крім того, шаблони відрізняються призначенням. Є три основні групи:

- Породжуючі – забезпечують гнучке створення об'єктів без внесення в програму зайвих залежностей.
- Структурні – показують різні способи побудови зв'язків між об'єктами.
- Поведінкові – забезпечують ефективну комунікацію між об'єктами [11].

В даній кваліфікаційній роботі буде використано шаблон MVC, котрий є універсальним. Наразі він є популярним патером, часто його використовуються в веб-додатках, аби відокремити логіку роботи від інтерфейсу. Також іноді трапляється, що його використовують там, де в цьому немає необхідності, це зазвичай роблять новачки. MVC для даного додатку гарного підійде, в нас є внутрішня логіка роботи, як: зберігання, додавання, зміна задач, контролювання їх стану, прорахування календарю задач; так і графічний інтерфейс, котрий має різні методи представлення інформації о задачах, функціональні поля та кнопки, вікна повідомлень про помилки користувача та оповіщення про задачі.

MVC (Model-View-Controller) – це архітектурний шаблон, який ділить систему на три складових модулі: модель (Model), представлення (View), контролер (Controller). Користувач дає команди програмі через різні елементи інтерфейсу. Контролер перехоплює ці команди та перетворює дані в Моделі. Модель оновлюється та повідомляє Представленню про те, що потрібно перемалювати інтерфейс, щоб відобразити зміни в даних. На рисунку 2.9 представлено взаємодію модулів шаблону MVC.

Рисунок 2.9 – Робота Model-View-Controller

Модель містить в собі функціональну бізнес-логіку додатку, тобто дані та методі програми. Модель є повністю незалежною від інших елементів системи, саме від неї залежить, щоб буде демонструвати Представлення, та як працюватиме Контролер. Модель нічого не знає про модуль Представлення, що і як воно там відображається користувачу. Це дозволяє змінювати представлення даних, не чіпаючи саму Модель. Вона має такі ознаки:

- Модель – це бізнес-логіка програми;
- Модель має знання про себе саму і не знає про Контролери і Представлення;

- Для деяких проектів Модель це просто шар даних (DAO, база даних, XML-файл) [12].

Представлення відображає дані у зручному та зрозумілому вигляді, які були отримані від Моделі. Представлення не може безпосередньо впливати на Контролер або Модель. Можна сказати, що Представлення має доступ «тільки на читання» до даних.

Контролер обробляє будь-які дії користувача. Саме через Контролер користувач вносить зміни до Моделі. Точніше дані, які зберігаються в Моделі. Він також може приймати та обробляти дані від Представлення і потім вносити до Моделі.

При використанні архітектурного патерну проектування MVC, який включає в собі три модулі, було додано ще один, який відповідає за сповіщення про задачі. На рисунку 2.10 приклад імплементації MVC патерну в нашу роботу, де зазначені класи були відповідно розбиті на модулі, як фізично, тобто зберігаються в відповідних пакетах, так і логічно, де класи одного модуля не залежать від іншого і могли розроблятися паралельно.

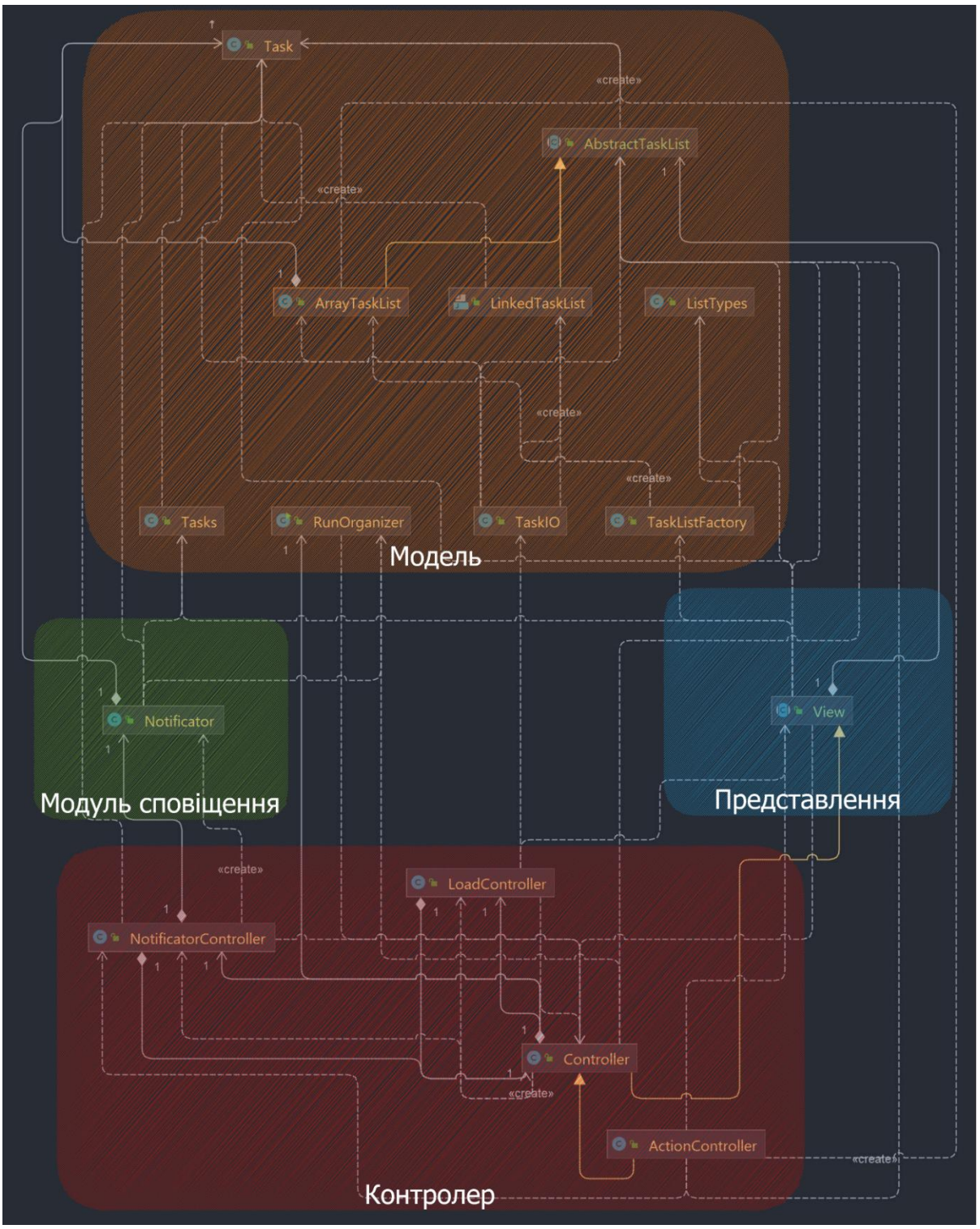

Рисунок 2.10 – Реалізація MVC в програмі

2.3. Проектування UI/UX дизайну

Графічний інтерфейс для десктопного додатку, а особливо органайзеру, є важливим елементом розробки, адже через нього будуть проводитися всі дії,

перегляд наявної інформації, додавання, зміна, видалення інформації. Без нього би довелося вводити команди вручну у консоль, або щось подібне. Також для цієї роботи інтерфейс повинен бути простим, щоб користувача не лякала велика наявність різних полів та кнопок; зручним, щоб користувач міг легко і чітко знайти потрібну йому інформацію, а не клацати все підряд в надії, що це буде саме те поле або кнопка, яка йому потрібна. І не останню роль в усьому цьому відіграє привабливість інтерфейсу, якщо інтерфейс користувачу сподобався, то шанс, що користувач використає цей додаток, збільшується, і в майбутньому може збільшити потенційних користувачів даної програми.

UI-дизайн – це графічний, візуальний макет програми. Він відповідає за кольорову палітру програми, оформлення кнопок, полів, шрифт та розмір тексту, різноманітність іконок та зображень. Також сюди входить анімація, переходи між сторінками або вікнами. Саме дизайн вирішує як буде виглядати додаток, він надає тої привабливості інтерфейсу, що стимулює користувача натиснути на ту чи інше зображення або кнопку. Дизайн звісно підтримуються певною тематикою, стилем, щоб відповідати меті, особистості програми. З нього дизайнери можуть зчитати характер бренду та його емоційний посил. Також важливо мати на увазі особливості зорового сприйняття, тобто робити текст контрастним до фону, уникати надто яскравих кольорів, щоб очі користувача швидко не втомлювалися, враховати різні особливості моніторів – їх роздільну здатність та кольору – щоб скрізь сайт відображався коректно.

UX-дизайн – це досвід користувача. Він визначається тим, як користувач взаємодіє з інтерфейсом. Чи є досвід взаємодії з додатком гладким та інтуїтивним чи незграбним і заплутаним? Навігація додатком виглядає логічною чи хаотичною? Досвід користувача визначається тим, наскільки легко чи важко взаємодіяти з елементами інтерфейсу, створеними UI-дизайнерами. UX-дизайн відповідає за те, як саме буде працювати UI-інтерфейс, для цього UX-дизайнери враховують психологічне сприйняття та звички користувачів. Найчастіше найзручніший інтерфейс – це той, який схожий на інші, адже тоді користувач без проблем знайде потрібні поля,

кнопки та сторінки. Тут також існують шаблони, як і в проєктуванні, які демонструють як користувачі взаємодіють з інтерфейсом, куди нажимають, де тримають курсор, скільки хвилин були на той чи іншій сторінці. Крім шаблонів, UX-дизайн досліджує шлях користувача – від потрапляння на сайт до цільової дії, коли відвідувач оформив замовлення або перейшов за посиланням. Враховують різні сценарії, наприклад, одні користувачі схильні швидко робити покупки, а інші довго дивитимуться каталог або відволікатимуться на інші розділи інтерфейсу [13].

Отже, врахувавши всі ці момент був розроблений фінальний макет графічного інтерфейсу додатку, представлений на рисунках 2.11-2.12, який потім потрібно відтворити за допомогою програмного коду на платформі JavaFX.

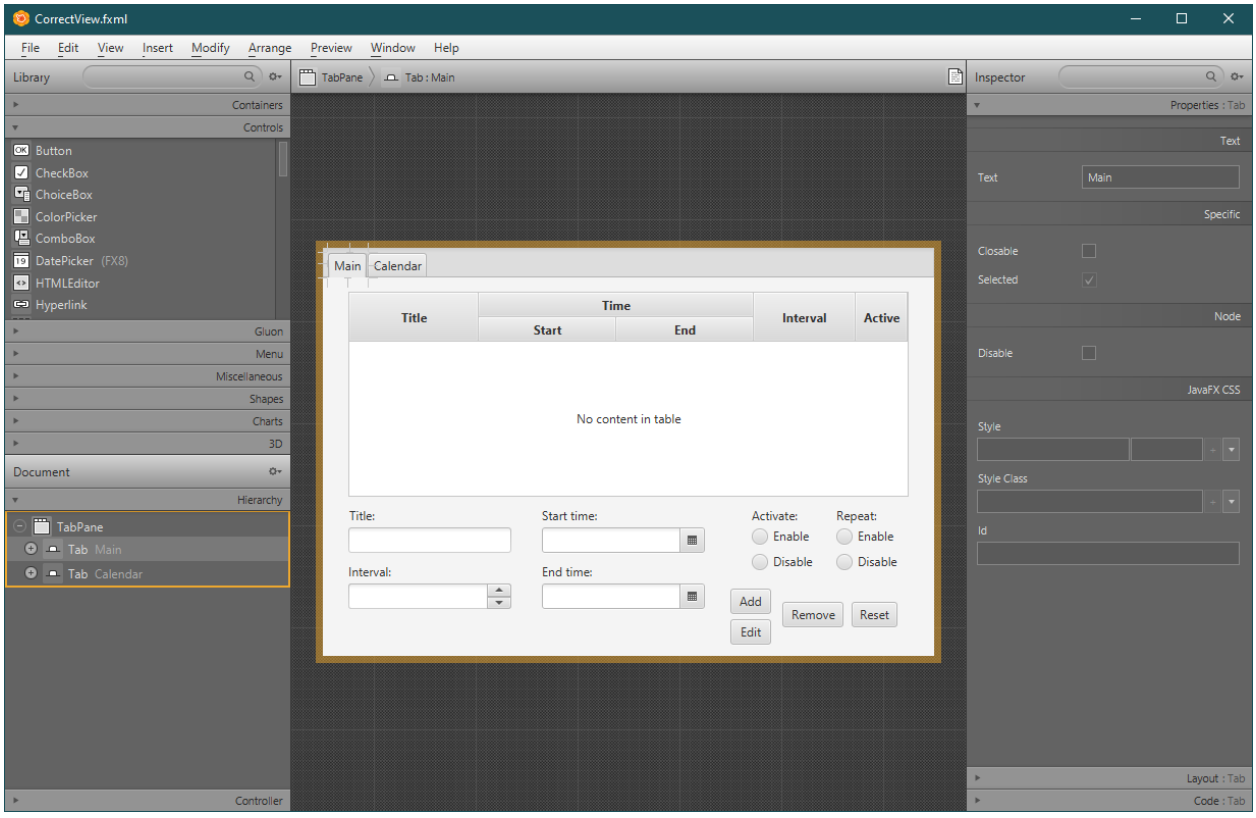

Рисунок 2.11 – Макет головно вікна додатку

Інтерфейс був розроблений у стриманих, переважно сірих, кольорах, що надає користувачу ефект сконцентрованості та звичності, так як це органайзер,

то ним буду користуватися дуже часто, і ця палітра налаштує користувача на вірну роботу.

Головне вікно містить в собі:

- таблицю з задачами, їхній опис: назва, час початку та кінця, активна задача та інтервал (через скільки часу повториться наступна задача), якщо задача повторювальна;
- поля для вводу інформації та зміни вже існуючих задач;
- перемикачі для активація та повтореності задачі;
- кнопки додавання, зміни, видалення задачі та скидання всіх полів та вибраних елементів інтерфейсу.

Здається, що інтерфейс трішки громіздкий, але це не так, всі елементи розташовані поруч, щоб мати швидкий доступ до них, та тримати їх у полі зору, якщо користувач вів дані невірно.

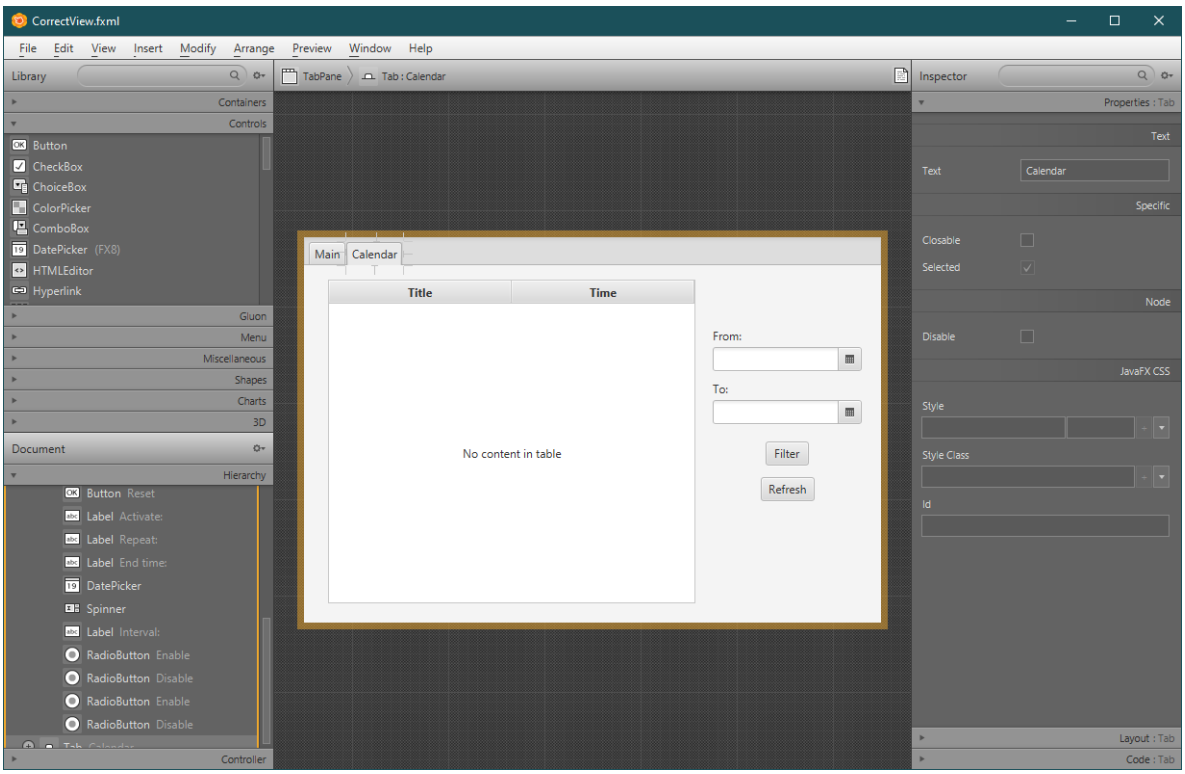

Рисунок 2.12 – Макет вікна з календарем

Вікно «Календар» містить в собі таблицю з назвою задачі та точним часом її виконання, а поля відповідають за проміжок часу в календарі.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1. Розробка бізнес-логіки

Головна ціль органайзеру – зберігання задач користувача. Та весь функціонал будь-якого органайзеру зосереджений на маніпуляції з задачами, зміна їх параметрів, додавання нової задачі, видалення, створення календарю задач тощо. Тому було розроблено клас «Task», на основі якого будуть створені об’єкти задач, котрі зберігають властивості задачі. Тобто, один об’єкт відповідає одній задачі користувача.

```
public class Task implements Cloneable, Externalizable {
    private String title;
    private LocalDateTime time;
    private LocalDateTime start;
    private LocalDateTime end;
    private Duration interval;
    private boolean active;
    private boolean repeated;

    public Task() {
    }

    public Task(String title, LocalDateTime time) throws
IllegalArgumentException {
        if (time == null) {
            throw new IllegalArgumentException("Time must not be null.");
        }
        this.title = title;
        this.time = time;
        this.active = false;
        this.repeated = false;
    }
}
```

Також клас містить основний метод, котрий прораховую час наступного виконання задачі, якщо вона є повторювальною. Цей метод буде багато ще разів зустрічатися, аби, наприклад, створити той же календар.

```
public LocalDateTime nextTimeAfter(LocalDateTime current) throws
IllegalArgumentException {
    if (current == null) {
```

```
        throw new IllegalArgumentException("Specified time must not be
null.");
    }
    if (isActive()) {
        if (!isRepeated()) {
            return (current.isAfter(time) || current.isEqual(time) ? null :
time);
        } else {
            if (current.isBefore(end)) {
                for (LocalDateTime i = start; i.isBefore(end) ||
i.isEqual(end); i = i.plus(interval)) {
                    if (current.isBefore(i)) {
                        return i;
                    }
                }
            }
            return null;
        }
    } else {
        return null;
    }
}
```

Відповідні задачі користувача потрібно десь зберігати під час виконання роботи. Тому було розроблено власне сховище задач, аби мати доступ до них в одному місці, можна було швидко проводити будь-які операції з задачами: пошук, додання, видалення. Клас «ArrayTaskList» та «LinkedTaskList» ефективно зберігають всі задачі та надають доступ до них під час виконання програми. Було розроблено два класи, так як вони мають різні цілі. «ArrayTaskList» працює як масив, при додаванні нового елементу в заповнений масив буде працювати повільно, тому що буде створювати новий більший за розміром масив, який буде копіювати елементи зі старого. При видаленні елемента з початку або кінця масиву, привиде до зсуву масиву, що також буде позначатися на швидкодії. Але при пошуку елемента за індексом працює дуже швидко, а також був створений метод «trimCapacity» для збільшення ефективності пошуку за властивістю об’єкта, котрий зменшує місткість масиву, якщо його розмір в півтора рази менша за місткість. Нижче наведено його реалізацію.

```
private void trimCapacity() {
    if (size != 0 && ((float)tasks.length / (float)size) >= RATIO
```

```
            && tasks.length > DEFAULT_CAPACITY) {
        Task[] tempTasks;
        if ((tasks.length / RATIO) <= DEFAULT_CAPACITY) {
            tempTasks = new Task[DEFAULT_CAPACITY];
        } else {
            tempTasks = new Task[Math.round(tasks.length / RATIO)];
        }
        System.arraycopy(tasks, 0, tempTasks, 0, size);
        tasks = tempTasks;
    }
}
```

«LinkedTaskList» працює як двонаправлений зв’язний список, тобто швидке додавання та видалення елементів в кінці, але довгий пошук елемента, якщо він знадиться в середині списку. Реалізація його представлена в вигляді вузлів, котрі зберігаються інформацію про наступний та попередній елемент.

```
private static class Node {
    Task item;
    Node next;
    Node previous;

    Node(Task item, Node previous, Node next) {
        this.item = item;
        this.previous = previous;
        this.next = next;
    }
```

У таблиці 3.1 представлені характеристики сховищ.

Таблиця 3.1 – Характеристик сховищ даних

	Часова складність			
Тип сховища	Індекс	Пошук	Додавання	Видалення
LinkedTaskList	O(n)	O(n)	O(1)	O(1)
ArrayTaskList	O(1)	O(n)	O(n)	O(n)

Після реалізації фундаменту програми, можна реалізовувати календар задач. Він буде представляти з себе відсортовану мапу (сховище, яке зберігає пару – ключ та значення) задач, де ключ – це назва або назви задач, а значенням

виступає конкретна дата та час виконання. Нижче наведено реалізацію, яка складається з двох методів.

```
public static SortedMap<LocalDateTime, Set<Task>> calendar(Iterable<Task>
tasks, LocalDateTime start, LocalDateTime end) {
    SortedMap<LocalDateTime, Set<Task>> taskMap = new TreeMap<>();
    Set<LocalDateTime> timeSet = new HashSet<>();
    Iterable<Task> incomingTasks = Tasks.incoming(tasks, start, end);
    for (Task temp : incomingTasks) {
        if (temp.isRepeated()) {
            timeSet.addAll(Tasks.getTimelineOfRepeatedTask(temp, start,
end));
        } else {
            timeSet.add(temp.getTime());
        }
    }
    for (LocalDateTime tempTime : timeSet) {
        Set<Task> tasksSet = new HashSet<>();
        taskMap.put(tempTime, tasksSet);
        for (Task tempTask : incomingTasks) {
            if (tempTask.isRepeated()) {
                for (LocalDateTime tempTaskTime :
Tasks.getTimelineOfRepeatedTask(tempTask, start, end)) {
                    if (tempTime.equals(tempTaskTime)) {
                        tasksSet.add(tempTask);
                        break;
                    }
                }
            } else {
                if (tempTime.equals(tempTask.getTime())) {
                    tasksSet.add(tempTask);
                }
            }
        }
    }
    return taskMap;
}

private static List<LocalDateTime> getTimelineOfRepeatedTask(Task
repeatedTask, LocalDateTime start, LocalDateTime end) {
    List<LocalDateTime> taskTimingSet = new LinkedList<>();
    for (LocalDateTime i = repeatedTask.getStartTime();
                         i.isBefore(repeatedTask.getEndTime()) ||
i.isEqual(repeatedTask.getEndTime());
                          i = i.plusSeconds(repeatedTask.getRepeatInterval()))
{
        if (i.isAfter(start) && (i.isBefore(end) || i.isEqual(end))) {
            taskTimingSet.add(i);
        }
    }
    return taskTimingSet;
}
```

Зберігання задач користувача, коли додаток вимкнено, відбувається у файл, який зберігається разом з додатком. Для реалізації цього потрібно було написати спеціальні методи для серіалізації та десеріалізації об’єктів. Серіалізація – це процес перетворення вмісту об’єкту класу в потік байт, а десеріалізації – навпаки, перетворення потоку байтів в об’єкт класу. Нижче представлено реалізацію серіалізації та десеріалізації.

```
@Override
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeUTF(title);
    out.writeBoolean(active);
    if (interval == null) {
        out.writeLong(0L);
    } else {
        out.writeLong(interval.getSeconds());
    }
    if (this.isRepeated()) {
        out.writeByte(start.getSecond());
        out.writeByte(start.getMinute());
        out.writeByte(start.getHour());
        out.writeByte(start.getDayOfMonth());
        out.writeByte(start.getMonthValue());
        out.writeInt(start.getYear());

        out.writeByte(end.getSecond());
        out.writeByte(end.getMinute());
        out.writeByte(end.getHour());
        out.writeByte(end.getDayOfMonth());
        out.writeByte(end.getMonthValue());
        out.writeInt(end.getYear());
    } else {
        out.writeByte(time.getSecond());
        out.writeByte(time.getMinute());
        out.writeByte(time.getHour());
        out.writeByte(time.getDayOfMonth());
        out.writeByte(time.getMonthValue());
        out.writeInt(time.getYear());
    }
}

@Override
public void readExternal(ObjectInput in) throws IOException {
    title = in.readUTF();
    active = in.readBoolean();
    long interval = in.readLong();
    if (interval != 0L) {
        byte[] startTimes = readExternalTimeOfBytes(in);
        int startYear = in.readInt();
        byte[] endTimes = readExternalTimeOfBytes(in);
        int endYear = in.readInt();
```

```
        start = LocalDateTime.of(startYear, startTimes[4], startTimes[3],
startTimes[2], startTimes[1], startTimes[0]);
        end = LocalDateTime.of(endYear, endTimes[4], endTimes[3],
endTimes[2], endTimes[1], endTimes[0]);
        this.interval = Duration.ofSeconds(interval);
        repeated = true;
    } else {
        byte[] times = readExternalTimeOfBytes(in);
        int timeYear = in.readInt();
        time = LocalDateTime.of(timeYear, times[4], times[3], times[2],
times[1], times[0]);
        repeated = false;
    }
}
```

Ці методи тільки перетворюють об’єкт в байти чи навпаки, но ще потрібно записувати та зчитувати ці ж байти.

```
public static void write(AbstractTaskList taskList, OutputStream out) {
    try (ObjectOutputStream oos = new ObjectOutputStream(out)) {
        oos.writeObject(taskList);
        //taskList.writeExternal(oos);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
public static void read(AbstractTaskList taskList, InputStream in) {
    try (ObjectInputStream ois = new ObjectInputStream(in)) {
        AbstractTaskList tempTaskList = (AbstractTaskList) ois.readObject();
        for (Task temp : tempTaskList) {
            taskList.add(temp);
        }
        //taskList.readExternal(ois);
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}
public static void writeBinary(AbstractTaskList taskList, File file) {
    try (FileOutputStream fos = new FileOutputStream(file)) {
        write(taskList, fos);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
public static void readBinary(AbstractTaskList taskList, File file) {
    try (FileInputStream fis = new FileInputStream(file)) {
        read(taskList, fis);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
```

```
        e.printStackTrace();
    }
}
```

Створення сповіщення о задачах описується створенням циклу, який йде по календарю та звіряє майбутній час, тобто поточний плюс декілька хвилин, з часом виконання задачі. Якщо така задача знаходиться, то органайзер заздалегідь у спливаючому вікні повідомляє про задачі. Нижче наведена реалізація сповіщення.

```
private Map.Entry<LocalDateTime, Set<Task>> notifyTask() {
    for (Map.Entry<LocalDateTime, Set<Task>> entry :
Tasks.calendar(tasksList, LocalDateTime.now(),
LocalDateTime.now().plusMinutes(1).plusSeconds(notificationTime)).entrySet())
{
        if
(entry.getKey().withNano(0).isEqual(LocalDateTime.now().plusSeconds(notificat
ionTime).withNano(0))) {
            return entry;
        }
    }
    return null;
}
@Override
public void run() {
    while (true) {
        try {
            Map.Entry<LocalDateTime, Set<Task>> entry = notifyTask();
            if (entry != null) {
                Platform.runLater(() -> {
                    showAlert(entry);
                });
            }
            sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

На цьому закінчена розробка бізнес-логіки додатку, тобто модулю програми Модель із патерну проектування MVC, який відповідальний за виконання основних дій органайзеру. Лістинг повного коду наведено в додатку А.

3.2. Розробка графічного інтерфейсу

Органайзер виконаний у мінімалістичному стилі, стриманій, однобарвній сірій палітрі для підвищення концентрації користувача на виконані дій з додатком. Програма має два вікна – головне вікно («Main») та вікно календаря («Calendar»): де в головному вікні відбуваються всі основні дії, такі як додавання, зміна, видалення, перегляд задач, а у календарі відображається проміжок календарю та сам календар задач.

Головне вікно містить в собі таблицю всіх завдання, які користувач додав. Воно відображає властивості задачі, де «Title» відповідає за назву задачі, «Time Start» за початок виконання задачі, «Time End» за початок завершення повторювальної задачі, «Interval» за проміжок, коли повинна початися наступна повторювальна задача, «Active» за стан задачі, активна вона чи ні, що впливає на її відображенні в календарю та створення сповіщення для неї. На рисунку 3.1 представлена таблиця для перегляду всіх задач.

Title	Time		Interval	Active
	Start	End		
Task1	24.06.2022 22:39:51	25.06.2022 22:39:54	3600	-
Task2	30.06.2022 15:08:37	-	-	+

Рисунок 3.1 – Таблиця перегляду задач у вікні «Main»

Присутні чотири поля для вводу, такі як «Title», куди потрібно вводити назву задачі, «Start time» для вибору дати та вводу часу початку виконання задачі, «End time» для вибору дати та вводу часу кінця виконання задачі, «Interval» для позначки часу, коли повинна початись наступна задача. Поля

«Title» та «Start time» присутні завжди в додатку, а ось «End time» і «Interval» присутні тоді, коли користувач додає або змінює повторювальну задачу. Це зроблено для того, щоб користувача не відволікали зайві поля. На рисунку 3.2 продемонстрований вигляд полів.

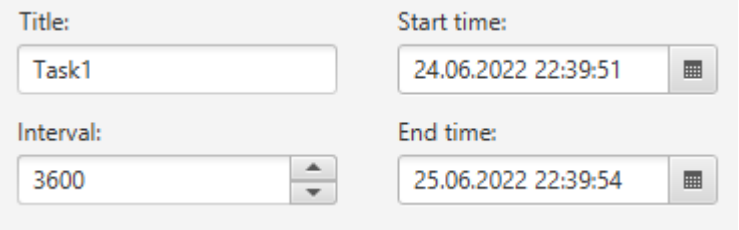

Рисунок 3.2 – Вигляд полів у вікні «Main»

Поля мають свою валідацію, тобто при введені недопустимого значення, користувачу буде виведено відповідне вікно з помилкою. Це допомагає виключити непорозуміння між програмою та користувачем, коли користувач ввів одні дані, а програма показує зовсім інші. Приклад одного із вікон помилок представлений на рисунку 3.3.

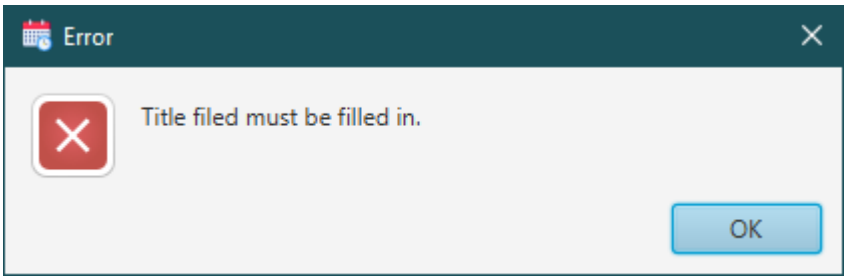

Рисунок 3.3 – Вікно помилки, яке сповіщає про некоректність назви задачі

Також є два перемикачі «Activate», «Repeat» та чотири кнопки «Add», «Edit», «Remove», «Reset». Перемикач «Activate» відповідає за те, який стан матиме задача, де про активну задачу сповіщувач буде повідомляти та задача буде включена до календаря, а неактивна – буде тільки зберігатися в списку задач для подальшої її зміни або видаленні. «Repeat» надає можливість зробити задачу повторювального типу, що з’являються приховані поля. Кнопки «Add» та «Remove» відповідають за додавання та видалення задачі

відповідно, «Edit» відповідає за зміну існуючої задачі. Для того щоб зробити це, потрібно із таблиці обрати потрібну задачу, полям присвоюються властивості задачі, змінити необхідні властивості та натиснути кнопку «Edit». «Reset» допомагає очистити всі поля, перемикачі та прибрати виділення з таблиці, аби було зручніше додавати нові задачі. На рисунку 3.4 представлений вигляд вікна «Main».

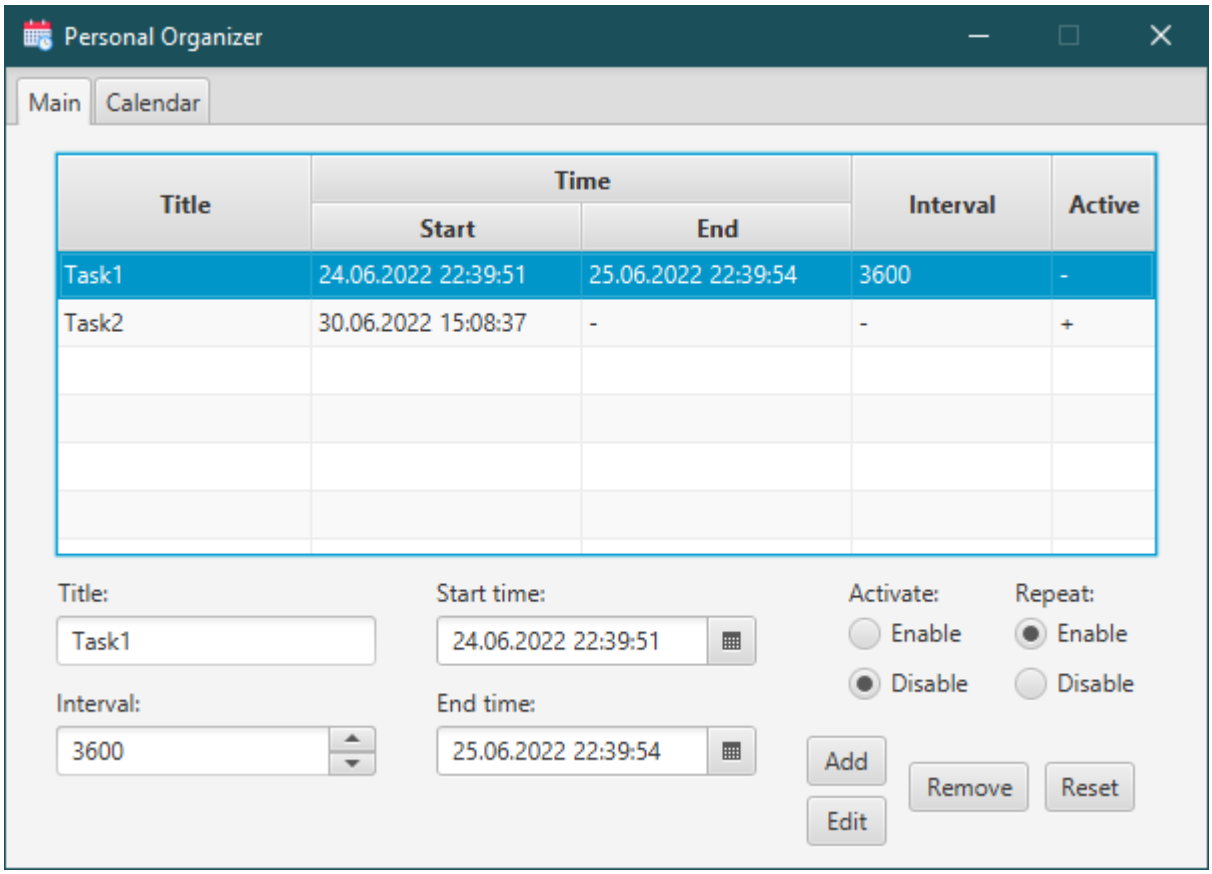

Рисунок 3.4 – Вигляд вікна «Main»

Вікно «Calendar» містить в собі таблицю задач, які будуть виконуватися в певний проміжок часу, котрий вказаний в полях «From» і «To». Таблиця, продемонстрована на рисунку 3.5, має дві колонки «Title» та «Time», де перша – це назва задачі, друга – це час виконання її. Таблиця відсортована за часом, спочатку йдуть ранні задачі, а потім пізні, і тим самим це створює календар задач на певний проміжок часу, встановлений користувачем.

Title	Time
Task1	25.06.2022 11:39:51
Task1	25.06.2022 12:39:51
Task1	25.06.2022 13:39:51
Task1	25.06.2022 14:39:51
Task1	25.06.2022 15:39:51
Task1	25.06.2022 16:39:51
Task1	25.06.2022 17:39:51
Task1	25.06.2022 18:39:51
Task1	25.06.2022 19:39:51
Task1	25.06.2022 20:39:51
Task1	25.06.2022 21:39:51
Task1	25.06.2022 22:39:51
Task2	30.06.2022 15:08:37

Рисунок 3.5 – Таблиця у вікні «Calendar»

Кнопка «Filter» запускає прорахунок календарю, а кнопка «Refresh» оновлює список календарю, що відбувається значно швидше, ніж створення з нулю, а також зберігає проміжок часу. На рисунку 3.6 представлено вигляд вікна «Calendar».

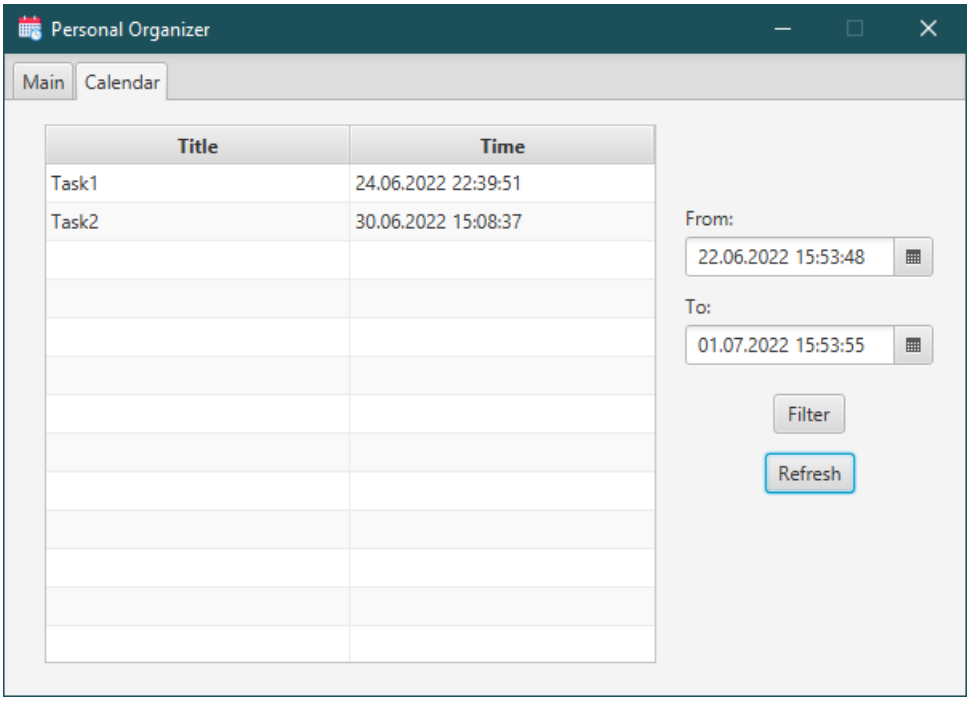

Рисунок 3.6 – Вигляд вікна «Calendar»

Органайзер володіє функцією зберігання даних та сповіщення о задачах. Аби зберігання було непомітним та не вимагало користувача згадувати про це, то цей процес відбувається автоматично при вимкнені додатку. Система сповіщення вмикається разом із додатком, тому піклуватися о включенні її необхідно. Коли до початку завдання залишається хвилина, то з'являється вікно сповіщення, котре містить в собі інформацію про назву задачі та час її виконання. На рисунку 3.7 представлено вікно сповіщення.

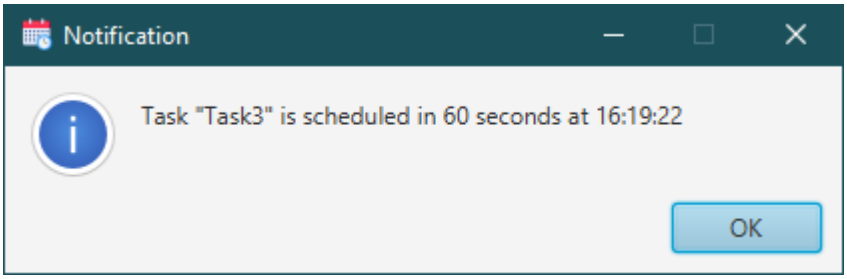

Рисунок 3.7 – Вікно сповіщення о задачі

Лістинг коду, який відповідає за графічний інтерфейс та його функціонал, представлений в додатку Б.

ВИСНОВКИ

Підсумовуючи хід виконання кваліфікаційної роботи, було отримано наступні результати:

- Було досліджено та аналізовано актуальність теми роботи, визначено поняття програмного органайзеру, його складові, які існують види.
- Проведено літературний огляд сучасних джерел за тематикою розробки десктопного додатку-органайзеру, на основі чого сформульовано задачу виконання, мету роботи, було проаналізовано схожі додатки.
- Визначено, які інструменти будуть ефективними та зручними для виконання даної роботи, а також зазначено задачі, які повинні бути виконані для завершення розробки додатку.
- Визначено структуру взаємодії з додатком, структури програми та патерн проектування системи для ефективної та сучасної розробки додатку.
- Розроблено макет графічного інтерфейсу додатку з дотриманням всіх сучасних вимог до UI/UX-дизайну.
- Був розроблений програмний органайзер згідно патерну проектування MVC, який задовольняє вимогам звичайного органайзеру.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Академик | Органайзер [Електронний ресурс] – Режим доступу до ресурсу: https://dic.academic.ru/dic.nsf/ruwiki/1077129.
2. Todoist | Features [Електронний ресурс] – Режим доступу до ресурсу: https://todoist.com/features
3. MyLifeOrganized [Електронний ресурс] – Режим доступу до ресурсу: https://www.mylifeorganized.net/index-ru.shtml
4. Tebapit | Що таке Java? [Електронний ресурс] – Режим доступу до ресурсу: https://tebapit.com/що-таке-java/
5. Вікіпедія | Java [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Java
6. Metanit | Введение в Java FX [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/java/javafx/1.1.php
7. What is Communication Diagram? [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-communication-diagram/
8. Вікіпедія | Діаграма класів [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Діаграма_класів
9. Википедия | Диаграмма класов [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Диаграмма_классов
10. UML Class Diagram [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/uml-class-diagram
11. Классификация паттернов [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/ru/design-patterns/classification
12. Хабр | Паттерны для новичков: MVC vs MVP vs MVVM [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/215605/
13. Что такое UX/UI-дизайн и как быстро освоить профессию [Електронний ресурс] – Режим доступу до ресурсу: https://bbbl.dev/articles/how-to-become-ux-ui-designer#title1

ДОДАТОК А

Лістинг коду бізнес-логіки додатку

RunOrganizer.java

```
package bublyk.app.organaizer;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;
import org.apache.log4j.PropertyConfigurator;
import bublyk.app.organaizer.controller.Controller;

import java.util.Objects;

public class RunOrganizer extends Application {
   @Override
   public void start(Stage stage) throws Exception {

PropertyConfigurator.configure(RunOrganizer.class.getResource("log4j.properti
es"));
      FXMLLoader loader = new
FXMLLoader(RunOrganizer.class.getResource("view.fxml"));
      Scene scene = new Scene(loader.load());
      Controller controller = loader.getController();
      controller.logger.debug("App is running.");
      stage.getIcons().add(new
Image(Objects.requireNonNull(RunOrganizer.class.getResource("OrganizerIcon.pn
g")).toExternalForm()));
      stage.setResizable(false);
      stage.setTitle("Personal Organizer");
      stage.setScene(scene);
      stage.show();
      stage.setOnCloseRequest(windowEvent -> {
         controller.writingData();
         controller.logger.debug("App is closed.");
      });
   }

   public static void main(String[] args) {
      launch();
   }
}
```

Model

AbstractTaskList.java

```
package bublyk.app.organaizer.model;

import java.io.*;
import java.time.LocalDateTime;
import java.util.Iterator;
```

```
import java.util.stream.Stream;

public abstract class AbstractTaskList implements Iterable<Task>, Cloneable,
Externalizable {
    public abstract void add(Task task);

    public abstract boolean remove(Task task);

    public abstract int size();

    public abstract Task getTask(int index);

    public final AbstractTaskList incoming(LocalDateTime from, LocalDateTime
to) throws IllegalArgumentException {
        if (from == null || to == null) {
            throw new IllegalArgumentException("Timestamps must equal to zero
or be greater than it.");
        }
        if (from.isAfter(to)) {
            throw new IllegalArgumentException("Time \"to\" must be greater
than \"from\".");
        }
        AbstractTaskList tempTaskList = getTaskList();
        getStream().filter((e) -> e.nextTimeAfter(from) != null &&
(e.nextTimeAfter(from).isBefore(to) || e.nextTimeAfter(from).isEqual(to)))
                .forEach(tempTaskList::add);
        return tempTaskList;
    }

    protected abstract AbstractTaskList getTaskList();

    @Override
    public AbstractTaskList clone() throws CloneNotSupportedException {
        return (AbstractTaskList) super.clone();
    }

    public abstract Stream<Task> getStream();

    @Override
    public abstract Iterator<Task> iterator();

    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeInt(size());
        for (Task temp : this) {
            temp.writeExternal(out);
        }
    }

   @Override
    public void readExternal(ObjectInput in) throws IOException,
ClassNotFoundException {
        int size = in.readInt();
        for (int i = 0; i < size; ++i) {
            Task temp = new Task();
            temp.readExternal(in);
```

```
            add(temp);
        }
    }
}
```

ArrayTaskList.java

```
package bublyk.app.organaizer.model;

import java.util.Arrays;
import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.Objects;
import java.util.stream.Stream;

public class ArrayTaskList extends AbstractTaskList {
    private final static int DEFAULT_CAPACITY = 10;
    private final static float RATIO = 1.5f;
    private int size;
    private Task[] tasks = new Task[DEFAULT_CAPACITY];

    @Override
    public void add(Task task) throws NullPointerException {
        if (task == null) {
            throw new NullPointerException("Cannot add null pointer.");
        }
        if (size == tasks.length) {
            Task[] tempTasks = new Task[Math.round(size * RATIO)];
            System.arraycopy(tasks, 0, tempTasks, 0, size);
            tasks = tempTasks;
        }
        tasks[size] = task;
        size++;
    }

    @Override
    public boolean remove(Task task) throws NullPointerException {
        if (task == null) {
            throw new NullPointerException("Cannot remove null pointer.");
        }
        if (size != 0) {
            for (int i = 0; i < size; ++i) {
                if (task.equals(tasks[i])) {
                    if (i != size - 1) {
                        System.arraycopy(tasks, i + 1, tasks, i, size - i -
1);
                    }
                    tasks[size - 1] = null;
                    size--;
                    trimCapacity();
                    return true;
                }
            }
        }
        return false;
    }
```

```
    @Override
    public int size() {
        return size;
    }

    private void trimCapacity() {
        if (size != 0 && ((float)tasks.length / (float)size) >= RATIO
                && tasks.length > DEFAULT_CAPACITY) {
            Task[] tempTasks;
            if ((tasks.length / RATIO) <= DEFAULT_CAPACITY) {
                tempTasks = new Task[DEFAULT_CAPACITY];
            } else {
                tempTasks = new Task[Math.round(tasks.length / RATIO)];
            }
            System.arraycopy(tasks, 0, tempTasks, 0, size);
            tasks = tempTasks;
        }
    }

    @Override
    public Task getTask(int index) throws IndexOutOfBoundsException {
        if (index >= size) {
            throw new IndexOutOfBoundsException("The index is out of
range.");
        }
        return tasks[index];
    }

    @Override
    protected ArrayTaskList getTaskList() {
        return new ArrayTaskList();
    }

    @Override
    public Iterator<Task> iterator() {
        return new Iterator<Task>() {
            private int size = size();
            private int currentElement = -1;
            private int nextElement;

            @Override
            public boolean hasNext() {
                return size > nextElement && tasks[nextElement] != null;
            }

            @Override
            public Task next() throws NoSuchElementException {
                if (!hasNext()) {
                    throw new NoSuchElementException("Iteration has no more
elements.");
                }
                nextElement++;
                currentElement++;
                return tasks[currentElement];
            }
```

```
            @Override
            public void remove() throws IllegalStateException {
                if (currentElement == -1) {
                    throw new IllegalStateException();
                }
                ArrayTaskList.this.remove(tasks[currentElement]);
                currentElement--;
                nextElement--;
                size--;
            }
        };
    }

    @Override
    public String toString() {
        StringBuilder tempString = new StringBuilder("ArrayTaskList(" +
size() + "): [");
        if (size() > 0) {
            for (Task temp : this) {

tempString.append(temp.toString()).append(";").append("\n\t\t\t\t\t");
            }
            tempString.delete(tempString.length() - 7, tempString.length());
        }
        return tempString.append("]").toString();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        ArrayTaskList tempArray = (ArrayTaskList) o;
        return size == tempArray.size && Arrays.equals(tasks,
tempArray.tasks);
    }

    @Override
    public int hashCode() {
        int result = Objects.hash(size);
        result = 31 * result + Arrays.hashCode(tasks);
        return result;
    }

    @Override
    public ArrayTaskList clone() throws CloneNotSupportedException {
        ArrayTaskList clone = (ArrayTaskList) super.clone();
        clone.tasks = tasks.clone();
        for (int i = 0; i < clone.size(); ++i) {
            clone.tasks[i] = clone.tasks[i].clone();
        }
        return clone;
    }
```

```
    @Override
    public Stream<Task> getStream() {
        return Arrays.stream(this.tasks).limit(size);
    }
}
```

LinkedTaskList.java

```
package bublyk.app.organaizer.model;

import java.util.*;
import java.util.stream.Stream;
import java.util.stream.StreamSupport;

public class LinkedTaskList extends AbstractTaskList {
    private int size;
    private Node first;
    private Node last;

    private static class Node {
        Task item;
        Node next;
        Node previous;

        Node(Task item, Node previous, Node next) {
            this.item = item;
            this.previous = previous;
            this.next = next;
        }

        private Node nodeClone() throws CloneNotSupportedException {
            Node clone = new Node(item.clone(), null, null);
            for (Node tempNext = next, temp = clone; tempNext != null;
tempNext = tempNext.next, temp = temp.next) {
                tempNext = new Node(tempNext.item.clone(), temp,
tempNext.next);
                temp.next = tempNext;
            }
            return clone;
        }

        private Node lastNodeClone() {
            Node clone = this;
            for (Node temp = next; temp != null; temp = temp.next) {
                if (temp.next == null) {
                    clone = temp;
                    break;
                }
            }
            return clone;
        }
    }

    private Node getNode(int index) {
        Node temp;
        if (index + 1 <= (Math.round((float)size / 2.))) {
```

```
            temp = first;
            for (int i = 0; i < index; ++i) {
                temp = temp.next;
            }
        } else {
            temp = last;
            for (int i = size - 1; i > index; --i) {
                temp = temp.previous;
            }
        }
        return temp;
    }

    private void createNode(Task item) {
        Node last = this.last;
        Node newNode = new Node(item, last, null);
        this.last = newNode;
        if (last == null) {
            first = newNode;
        } else {
            last.next = newNode;
        }
        size++;
    }

    private void deleteNode(Node node) {
        if (size > 1) {
            if (node.previous == null) {
                first = node.next;
                node.next.previous = null;
                node.next = null;
            } else if (node.next == null) {
                last = node.previous;
                node.previous.next = null;
                node.previous = null;
            } else {
                node.previous.next = node.next;
                node.next.previous = node.previous;
                node.next = null;
                node.previous = null;
            }
        } else {
            first = null;
            last = null;
        }
        node.item = null;
        size--;
    }

    @Override
    public void add(Task task) throws NullPointerException {
        if (task == null) {
            throw new NullPointerException("Cannot add null pointer.");
        }
        createNode(task);
    }
```

```
    @Override
    public boolean remove(Task task) throws NullPointerException {
        if (task == null) {
            throw new NullPointerException("Cannot remove null pointer.");
        }
        if (size != 0) {
            for (Node temp = first; temp != null; temp = temp.next) {
                if (task.equals(temp.item)) {
                    deleteNode(temp);
                    return true;
                }
            }
        }
        return false;
    }

    @Override
    public int size() {
        return size;
    }

    @Override
    public Task getTask(int index) throws IndexOutOfBoundsException {
        if (index >= size) {
            throw new IndexOutOfBoundsException("The index is out of
range.");
        }
        return getNode(index).item;
    }

    @Override
    protected LinkedTaskList getTaskList() {
        return new LinkedTaskList();
    }

    @Override
    public Iterator<Task> iterator() {
        return new Iterator<Task>() {
            private Node currentElement;
            private Node nextElement = first;

            @Override
            public boolean hasNext() {
                return nextElement != null;
            }

            @Override
            public Task next() throws NoSuchElementException {
                if (!hasNext()) {
                    throw new NoSuchElementException("Iteration has no more
elements.");
                }
                currentElement = nextElement;
                nextElement = nextElement.next;
                return currentElement.item;
```

```
            }

            @Override
            public void remove() throws IllegalStateException {
                if (currentElement == null) {
                    throw new IllegalStateException();
                }
                LinkedTaskList.this.remove(currentElement.item);
                currentElement = null;
            }
        };
    }

    @Override
    public String toString() {
        StringBuilder tempString = new StringBuilder("LinkedTaskList(" +
size() + "): [");
        if (size() > 0) {
            for (Task temp : this) {

tempString.append(temp.toString()).append(";").append("\n\t\t\t\t\t");
            }
            tempString.delete(tempString.length() - 7, tempString.length());
        }
        return tempString.append("]").toString();
    }

   @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        LinkedTaskList tempLinked = (LinkedTaskList) o;
        if (size() != tempLinked.size()) {
            return false;
        }
        for (Iterator<Task> i1 = this.iterator(), i2 = tempLinked.iterator();
i1.hasNext() && i2.hasNext();) {
            if (!i1.next().equals(i2.next())) {
                return false;
            }
        }
        return true;
    }

    @Override
    public int hashCode() {
        if (this.first == null) {
            return 0;
        }
        int result = Objects.hash(size);
        int tempResult = 1;
        for (Task temp : this) {
```

```
            tempResult = 31 * tempResult + (temp == null ? 0 :
temp.hashCode());
        }
        result = 31 * result + tempResult;
        return result;
    }

    @Override
    public LinkedTaskList clone() throws CloneNotSupportedException {
        LinkedTaskList clone = (LinkedTaskList) super.clone();
        if (first != null) {
            clone.first = first.nodeClone();
            clone.last = clone.first.lastNodeClone();
        }
        return clone;
    }

    @Override
    public Spliterator<Task> spliterator() {
        return Spliterators.spliterator(iterator(), size, Spliterator.SIZED |
Spliterator.ORDERED | Spliterator.IMMUTABLE);
    }

    @Override
    public Stream<Task> getStream() {
        return StreamSupport.stream(spliterator(), false);
    }
}
```

ListTypes.java

```
package bublyk.app.organaizer.model;

public class ListTypes {
    public enum types{
        ARRAY, LINKED
    }
}
```

Task.java

```
package bublyk.app.organaizer.model;

import java.io.*;
import java.time.*;
import java.util.Objects;

public class Task implements Cloneable, Externalizable {
    private String title;
    private LocalDateTime time;
    private LocalDateTime start;
    private LocalDateTime end;
    private Duration interval;
    private boolean active;
    private boolean repeated;

    public Task() {
    }
```

```
    public Task(String title, LocalDateTime time) throws
IllegalArgumentException {
        if (time == null) {
            throw new IllegalArgumentException("Time must not be null.");
        }
        this.title = title;
        this.time = time;
        this.active = false;
        this.repeated = false;
    }

    public Task(String title, LocalDateTime start, LocalDateTime end, int
interval) throws IllegalArgumentException {
        if (start == null || end == null) {
            throw new IllegalArgumentException("Timestamps must not be
null.");
        }
        if (start.isAfter(end)) {
            throw new IllegalArgumentException("The end of task must be
greater than the start of task.");
        }
        if (interval <= 0) {
            throw new IllegalArgumentException("Interval must be greater than
zero.");
        }
        this.title = title;
        this.start = start;
        this.end = end;
        this.interval = Duration.ofSeconds(interval);
        this.active = false;
        this.repeated = true;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public boolean isActive() {
        return active;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    public LocalDateTime getTime() {
        return (isRepeated() ? start : time);
    }

    public void setTime(LocalDateTime time) throws IllegalArgumentException {
        if (time == null) {
```

```
            throw new IllegalArgumentException("Time must not be null.");
        }
        if (isRepeated()) {
            this.repeated = false;
            this.start = null;
            this.end = null;
            this.interval = null;
        }
        this.time = time;
    }

    public LocalDateTime getStartTime() {
        return (isRepeated() ? start : time);
    }

    public LocalDateTime getEndTime() {
        return (isRepeated() ? end : time);
    }

    public int getRepeatInterval() {
        return (isRepeated() ? (int) interval.getSeconds() : 0);
    }

    public void setTime(LocalDateTime start, LocalDateTime end, int interval)
throws IllegalArgumentException {
        if (start == null || end == null) {
            throw new IllegalArgumentException("Timestamps must not be
null.");
        }
        if (start.isAfter(end)) {
            throw new IllegalArgumentException("The end of task must be
greater than the start of task.");
        }
        if (interval <= 0) {
            throw new IllegalArgumentException("Interval must be greater than
zero.");
        }
        if (!isRepeated()) {
            this.repeated = true;
            this.time = null;
        }
        this.start = start;
        this.end = end;
        this.interval = Duration.ofSeconds(interval);
    }

    public boolean isRepeated() {
        return repeated;
    }

    public LocalDateTime nextTimeAfter(LocalDateTime current) throws
IllegalArgumentException {
        if (current == null) {
            throw new IllegalArgumentException("Specified time must not be
null.");
        }
```

```
        if (isActive()) {
            if (!isRepeated()) {
                return (current.isAfter(time) || current.isEqual(time) ? null
: time);
            } else {
                if (current.isBefore(end)) {
                    for (LocalDateTime i = start; i.isBefore(end) ||
i.isEqual(end); i = i.plus(interval)) {
                        if (current.isBefore(i)) {
                            return i;
                        }
                    }
                }
                return null;
            }
        } else {
            return null;
        }
    }

    @Override
    public String toString() {
        if (isRepeated()) {
            return "Task \"" + title + "\": {start = " + start
                    + "; end = " + end + "; interval = " +
interval.getSeconds()
                    + "; active -> " + active + "; repeated -> " + repeated +
"}";
        } else {
            return "Task \"" + title + "\": {time = " + time
                    + "; active -> " + active + "; repeated -> " + repeated +
"}";
        }
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        Task task = (Task) o;
        return Objects.equals(time, task.time) && Objects.equals(start,
task.start) && Objects.equals(end, task.end)
                && Objects.equals(interval, task.interval) && active ==
task.active
                && repeated == task.repeated && Objects.equals(title,
task.title);
    }

    @Override
    public int hashCode() {
        return Objects.hash(title, time, start, end, interval, active,
repeated);
```

```
    }

    @Override
    public Task clone() throws CloneNotSupportedException {
        return (Task) super.clone();
    }

    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeUTF(title);
        out.writeBoolean(active);
        if (interval == null) {
            out.writeLong(0L);
        } else {
            out.writeLong(interval.getSeconds());
        }
        if (this.isRepeated()) {
            out.writeByte(start.getSecond());
            out.writeByte(start.getMinute());
            out.writeByte(start.getHour());
            out.writeByte(start.getDayOfMonth());
            out.writeByte(start.getMonthValue());
            out.writeInt(start.getYear());

            out.writeByte(end.getSecond());
            out.writeByte(end.getMinute());
            out.writeByte(end.getHour());
            out.writeByte(end.getDayOfMonth());
            out.writeByte(end.getMonthValue());
            out.writeInt(end.getYear());
        } else {
            out.writeByte(time.getSecond());
            out.writeByte(time.getMinute());
            out.writeByte(time.getHour());
            out.writeByte(time.getDayOfMonth());
            out.writeByte(time.getMonthValue());
            out.writeInt(time.getYear());
        }
    }

    @Override
    public void readExternal(ObjectInput in) throws IOException {
        title = in.readUTF();
        active = in.readBoolean();
        long interval = in.readLong();
        if (interval != 0L) {
            byte[] startTimes = readExternalTimeOfBytes(in);
            int startYear = in.readInt();
            byte[] endTimes = readExternalTimeOfBytes(in);
            int endYear = in.readInt();
            start = LocalDateTime.of(startYear, startTimes[4], startTimes[3],
startTimes[2], startTimes[1], startTimes[0]);
            end = LocalDateTime.of(endYear, endTimes[4], endTimes[3],
endTimes[2], endTimes[1], endTimes[0]);
            this.interval = Duration.ofSeconds(interval);
            repeated = true;
```

```
        } else {
            byte[] times = readExternalTimeOfBytes(in);
            int timeYear = in.readInt();
            time = LocalDateTime.of(timeYear, times[4], times[3], times[2],
times[1], times[0]);
            repeated = false;
        }
    }

    private byte[] readExternalTimeOfBytes(ObjectInput in) throws IOException
{
        byte[] times = new byte[5];
        for (int i = 0; i < times.length; ++i) {
            times[i] = in.readByte();
        }
        return times;
    }
}
```

TaskIO.java

```
package bublyk.app.organaizer.model;

import com.google.gson.*;

import java.io.*;
import java.time.Duration;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class TaskIO {

    public static void write(AbstractTaskList taskList, OutputStream out) {
        try (ObjectOutputStream oos = new ObjectOutputStream(out)) {
            oos.writeObject(taskList);
            //taskList.writeExternal(oos);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void read(AbstractTaskList taskList, InputStream in) {
        try (ObjectInputStream ois = new ObjectInputStream(in)) {
            AbstractTaskList tempTaskList = (AbstractTaskList)
ois.readObject();
            for (Task temp : tempTaskList) {
                taskList.add(temp);
            }
            //taskList.readExternal(ois);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static void writeBinary(AbstractTaskList taskList, File file) {
        try (FileOutputStream fos = new FileOutputStream(file)) {
            write(taskList, fos);
```

```
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void readBinary(AbstractTaskList taskList, File file) {
        try (FileInputStream fis = new FileInputStream(file)) {
            read(taskList, fis);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void write(AbstractTaskList taskList, Writer out) {
        try (BufferedWriter bufW = new BufferedWriter(out)) {
            GsonBuilder gsonBuilder;
            if (LinkedTaskList.class.equals(taskList.getClass())) {
                gsonBuilder = gsonBuilderSerializerForLinkedTaskList();
            } else {
                gsonBuilder = gsonBuilderSerializerForArrayTaskList();
            }
            Gson gson = gsonBuilder.setPrettyPrinting().create();
            bufW.write(gson.toJson(taskList, taskList.getClass()));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void read(AbstractTaskList taskList, Reader in) {
        try (BufferedReader bufR = new BufferedReader(in)) {
            GsonBuilder gsonBuilder;
            if (LinkedTaskList.class.equals(taskList.getClass())) {
                gsonBuilder = gsonBuilderDeserializerForLinkedTaskList();
            } else {
                gsonBuilder = gsonBuilderDeserializerForArrayTaskList();
            }
            Gson gson = gsonBuilder.create();
            for (Task temp : gson.fromJson(bufR, taskList.getClass())) {
                taskList.add(temp);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void writeText(AbstractTaskList taskList, File file) {
        try (FileWriter fw = new FileWriter(file)) {
            write(taskList, fw);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    public static void readText(AbstractTaskList taskList, File file) {
        try (FileReader fr = new FileReader(file)) {
            read(taskList, fr);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static GsonBuilder gsonBuilderSerializerForLinkedTaskList() {
        GsonBuilder gsonBuilder = gsonBuilderSerializerForLocalDateTime();
        gsonBuilder.registerTypeAdapter(LinkedTaskList.class,
(JsonSerializer<LinkedTaskList>) (linkedTaskList, type,
jsonSerializationContext) -> {
            JsonObject jo = new JsonObject();
            JsonArray tasks = new JsonArray();
            for (Task temp : linkedTaskList) {
                tasks.add(jsonSerializationContext.serialize(temp));
            }
            jo.addProperty("size", linkedTaskList.size());
            jo.add("tasks", tasks);
            return jo;
        });
        return gsonBuilder;
    }

    private static GsonBuilder gsonBuilderDeserializerForLinkedTaskList() {
        GsonBuilder gsonBuilder = gsonBuilderDeserializerForLocalDateTime();
        gsonBuilder.registerTypeAdapter(LinkedTaskList.class,
(JsonDeserializer<LinkedTaskList>) (json, type, jsonDeserializationContext) -
> {
            LinkedTaskList linkedTaskList = new LinkedTaskList();
            int size = json.getAsJsonObject().get("size").getAsInt();
            for (int i = 0; i < size; ++i) {

linkedTaskList.add(jsonDeserializationContext.deserialize(json.getAsJsonObjec
t().get("tasks").getAsJsonArray().get(i).getAsJsonObject(), Task.class));
            }
            return linkedTaskList;
        });
        return gsonBuilder;
    }

    private static GsonBuilder gsonBuilderSerializerForArrayTaskList() {
        GsonBuilder gsonBuilder = gsonBuilderSerializerForLocalDateTime();
        gsonBuilder.registerTypeAdapter(Task[].class,
(JsonSerializer<Task[]>) (tasks, type, jsonSerializationContext) -> {
            JsonArray ja = new JsonArray();
            for (Task temp : tasks) {
                if (temp == null) {
                    break;
                }
                ja.add(jsonSerializationContext.serialize(temp));
            }
```

```
            return ja;
        });
        return gsonBuilder;
    }

    private static GsonBuilder gsonBuilderDeserializerForArrayTaskList() {
        return gsonBuilderDeserializerForLocalDateTime();
    }

    private static GsonBuilder gsonBuilderSerializerForLocalDateTime() {
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.registerTypeAdapter(LocalDateTime.class,
(JsonSerializer<LocalDateTime>) (ldt, type, jsonSerializationContext) ->
                //new JsonPrimitive(ldt.toEpochSecond(ZoneOffset.UTC)));
                new JsonPrimitive(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss").format(ldt)));
        gsonBuilder.registerTypeAdapter(Duration.class,
(JsonSerializer<Duration>) (duration, type, jsonSerializationContext) ->
                new JsonPrimitive(duration.getSeconds()));
        return gsonBuilder;
    }

    private static GsonBuilder gsonBuilderDeserializerForLocalDateTime() {
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.registerTypeAdapter(LocalDateTime.class,
(JsonDeserializer<LocalDateTime>) (json, type, jsonDeserializationContext) ->
                //LocalDateTime.ofEpochSecond(json.getAsLong(), 0,
ZoneOffset.UTC));
                LocalDateTime.parse(json.getAsString(),
DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")));
        gsonBuilder.registerTypeAdapter(Duration.class,
(JsonDeserializer<Duration>) (json, type, jsonDeserializationContext) ->
                Duration.ofSeconds(json.getAsLong()));
        return gsonBuilder;
    }
}
```

TaskListFactory.java

```
package bublyk.app.organaizer.model;

public class TaskListFactory {

    public static AbstractTaskList createTaskList(ListTypes.types type)
throws IllegalArgumentException {
        switch (type) {
            case ARRAY:
                return new ArrayTaskList();
            case LINKED:
                return new LinkedTaskList();
            default:
                throw new IllegalArgumentException("This type doesn't
exist.");
        }
    }
}
```

Tasks.java

```
package bublyk.app.organaizer.model;

import java.lang.reflect.InvocationTargetException;
import java.time.LocalDateTime;
import java.util.*;

public class Tasks {

    public static Iterable<Task> incoming(Iterable<Task> tasks, LocalDateTime
from, LocalDateTime to) {
        if (from == null || to == null) {
            throw new IllegalArgumentException("Timestamps must equal to zero
or be greater than it.");
        }
        if (from.isAfter(to)) {
            throw new IllegalArgumentException("Time \"to\" must be greater
than \"from\".");
        }
        Iterable<Task> clone;
        try {
            clone = (Iterable<Task>)
tasks.getClass().getMethod("clone").invoke(tasks);
        } catch (NoSuchMethodException | IllegalAccessException |
InvocationTargetException e) {
            System.out.println("Cannot create new object for \"tasks\".
Result and your iterable object will be changed too.");
            e.printStackTrace(System.out);
            clone = tasks;
        }
        for (Iterator<Task> it = clone.iterator(); it.hasNext();) {
            Task temp = it.next();
            if (temp.nextTimeAfter(from) == null ||
temp.nextTimeAfter(from).isAfter(to)) {
                it.remove();
            }
        }
        return clone;
    }

    public static SortedMap<LocalDateTime, Set<Task>> calendar(Iterable<Task>
tasks, LocalDateTime start, LocalDateTime end) {
        SortedMap<LocalDateTime, Set<Task>> taskMap = new TreeMap<>();
        Set<LocalDateTime> timeSet = new HashSet<>();
        Iterable<Task> incomingTasks = Tasks.incoming(tasks, start, end);
        for (Task temp : incomingTasks) {
            if (temp.isRepeated()) {
                timeSet.addAll(Tasks.getTimelineOfRepeatedTask(temp, start,
end));
            } else {
                timeSet.add(temp.getTime());
            }
        }
        for (LocalDateTime tempTime : timeSet) {
            Set<Task> tasksSet = new HashSet<>();
```

```
            taskMap.put(tempTime, tasksSet);
            for (Task tempTask : incomingTasks) {
                if (tempTask.isRepeated()) {
                    for (LocalDateTime tempTaskTime :
Tasks.getTimelineOfRepeatedTask(tempTask, start, end)) {
                        if (tempTime.equals(tempTaskTime)) {
                            tasksSet.add(tempTask);
                            break;
                        }
                    }
                } else {
                    if (tempTime.equals(tempTask.getTime())) {
                        tasksSet.add(tempTask);
                    }
                }
            }
        }
        return taskMap;
    }

    private static List<LocalDateTime> getTimelineOfRepeatedTask(Task
repeatedTask, LocalDateTime start, LocalDateTime end) {
        List<LocalDateTime> taskTimingSet = new LinkedList<>();
        for (LocalDateTime i = repeatedTask.getStartTime();
                            i.isBefore(repeatedTask.getEndTime()) ||
i.isEqual(repeatedTask.getEndTime());
                             i =
i.plusSeconds(repeatedTask.getRepeatInterval())) {
            if (i.isAfter(start) && (i.isBefore(end) || i.isEqual(end))) {
                taskTimingSet.add(i);
            }
        }
        return taskTimingSet;
    }
}
```

Notification System

Notificator.java

```
package bublyk.app.organaizer.notification;

import bublyk.app.organaizer.RunOrganizer;
import javafx.application.Platform;
import javafx.scene.control.Alert;
import javafx.scene.image.Image;
import javafx.stage.Modality;
import javafx.stage.Stage;
import bublyk.app.organaizer.model.Task;
import bublyk.app.organaizer.model.Tasks;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Map;
import java.util.Objects;
```

```
import java.util.Set;

public class Notificator extends Thread {
    private Iterable<Task> tasksList;
    private final static DateTimeFormatter DATE_TIME_FORMATTER =
DateTimeFormatter.ofPattern("HH:mm:ss");
    private final static int DEFAULT_NOTIFICATION_TIME = 60;
    private int notificationTime;
    private final Image icon;

    public Notificator(Iterable<Task> tasksList, Image icon) {
        this.tasksList = tasksList;
        this.icon = icon;
        notificationTime = DEFAULT_NOTIFICATION_TIME;
    }

    public Notificator(Iterable<Task> tasksList) {
        this.tasksList = tasksList;
        this.icon = new
Image(Objects.requireNonNull(RunOrganizer.class.getResource("OrganizerIcon.pn
g")).toExternalForm());
        notificationTime = DEFAULT_NOTIFICATION_TIME;
    }

    private Map.Entry<LocalDateTime, Set<Task>> notifyTask() {
        for (Map.Entry<LocalDateTime, Set<Task>> entry :
Tasks.calendar(tasksList, LocalDateTime.now(),
LocalDateTime.now().plusMinutes(1).plusSeconds(notificationTime)).entrySet())
{
            if
(entry.getKey().withNano(0).isEqual(LocalDateTime.now().plusSeconds(notificat
ionTime).withNano(0))) {
                return entry;
            }
        }
        return null;
    }

    @Override
    public void run() {
        while (true) {
            try {
                Map.Entry<LocalDateTime, Set<Task>> entry = notifyTask();
                if (entry != null) {
                    Platform.runLater(() -> {
                        showAlert(entry);
                    });
                }
                sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    private void showAlert(Map.Entry<LocalDateTime, Set<Task>> result) {
```

```
        StringBuilder message = new StringBuilder();
        String verb;
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        Stage alertStage = (Stage)
alert.getDialogPane().getScene().getWindow();
        alertStage.getIcons().add(icon);
        alert.setTitle("Notification");
        alert.setHeaderText(null);
        alert.initModality(Modality.NONE);
        if (result.getValue().size() > 1) {
            message.append("Tasks ");
            verb = " are";
        } else {
            message.append("Task ");
            verb = " is";
        }
        for (Task temp : result.getValue()) {
            message.append("\"").append(temp.getTitle()).append("\", ");
        }
        message.delete(message.length() - 2,  message.length())
                .append(verb)
                .append(" scheduled in ")
                .append(notificationTime).append(" seconds at ")
                .append(result.getKey().format(DATE_TIME_FORMATTER));
        alert.setContentText(message.toString());
        alert.showAndWait();
    }

    public void setNotificationTime(int seconds) {
        notificationTime = seconds;
    }

    public void setTasksList(Iterable<Task> tasksList) {
        this.tasksList = tasksList;
    }

}
```

ДОДАТОК Б

Лістинг коду графічного інтерфейсу та його функціоналу

View

view.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.RadioButton?>
<?import javafx.scene.control.Spinner?>
<?import javafx.scene.control.Tab?>
<?import javafx.scene.control.TabPane?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.control.ToggleGroup?>
<?import javafx.scene.layout.AnchorPane?>
<?import tornadofx.control.DateTimePicker?>

<TabPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/17" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="bublyk.app.organaizer.controller.ActionController">
    <Tab closable="false" text="Main">
        <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="180.0"
prefWidth="200.0">
            <TableView fx:id="mainTable" layoutX="25.0" layoutY="14.0"
onMouseClicked="#selectColumn" prefHeight="201.0" prefWidth="558.0"
AnchorPane.bottomAnchor="156.0" AnchorPane.leftAnchor="25.0"
AnchorPane.rightAnchor="25.0" AnchorPane.topAnchor="14.0">
                <columns>
                    <TableColumn fx:id="titleMainColumn" maxWidth="600.0"
minWidth="50.0" prefWidth="200.0" text="Title" />
                    <TableColumn fx:id="timeMainColumn" maxWidth="700.0"
minWidth="200.0" prefWidth="291.0" text="Time">
                  <columns>
                     <TableColumn fx:id="startMainColumn" maxWidth="350.0"
minWidth="100.0" prefWidth="150.0" text="Start" />
                      <TableColumn fx:id="endMainColumn" maxWidth="350.0"
minWidth="100.0" prefWidth="150.0" text="End" />
                  </columns>
                    </TableColumn>
                    <TableColumn fx:id="intervalMainColumn" maxWidth="400.0"
minWidth="50.0" prefWidth="100.0" text="Interval" />
                    <TableColumn fx:id="activeMainColumn" maxWidth="150.0"
minWidth="35.0" prefWidth="35.0" text="Active" />
                </columns>
                <columnResizePolicy>
                    <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
                </columnResizePolicy>
            </TableView>
            <TextField fx:id="titleField" layoutX="25.0" layoutY="245.0"
prefWidth="160.0" />
```

```
            <Label layoutX="26.0" layoutY="225.0" prefHeight="17.0"
prefWidth="30.0" text="Title:" />
            <Label fx:id="startTimeLabel" layoutX="215.0" layoutY="225.0"
prefHeight="17.0" prefWidth="61.0" text="Start time:" />
            <DateTimePicker fx:id="startTimeField" layoutX="215.0"
layoutY="245.0" prefHeight="25.0" prefWidth="160.0" />
            <Button layoutX="400.0" layoutY="335.0" mnemonicParsing="false"
onMouseClicked="#editButtonAction" prefWidth="40.0" text="Edit" />
            <Button layoutX="400.0" layoutY="305.0" mnemonicParsing="false"
onMouseClicked="#addButtonAction" prefWidth="40.0" text="Add" />
            <Button layoutX="451.0" layoutY="318.0" mnemonicParsing="false"
onMouseClicked="#removeButtonAction" prefWidth="60.0" text="Remove" />
            <Button layoutX="519.0" layoutY="318.0" mnemonicParsing="false"
onMouseClicked="#resetButtonAction" text="Reset" />
         <Label layoutX="421.0" layoutY="225.0" prefHeight="17.0"
prefWidth="51.0" text="Activate:" />
         <Label layoutX="504.0" layoutY="225.0" text="Repeat:" />
         <Label fx:id="endTimeLabel" layoutX="215.0" layoutY="280.0"
prefHeight="17.0" prefWidth="56.0" text="End time:" />
         <DateTimePicker fx:id="endTimeField" layoutX="215.0" layoutY="300.0"
prefHeight="25.0" prefWidth="160.0" />
         <Spinner fx:id="intervalField" editable="true" layoutX="25.0"
layoutY="300.0" prefWidth="160.0" />
         <Label fx:id="intervalLabel" layoutX="25.0" layoutY="280.0"
text="Interval:" />
         <RadioButton fx:id="activeRadioTrue" layoutX="421.0" layoutY="245.0"
mnemonicParsing="false" text="Enable">
            <toggleGroup>
               <ToggleGroup fx:id="activateGroup" />
            </toggleGroup>
         </RadioButton>
         <RadioButton fx:id="activeRadioFalse" layoutX="421.0"
layoutY="270.0" mnemonicParsing="false" text="Disable"
toggleGroup="$activateGroup" />
         <RadioButton fx:id="repeatRadioTrue" layoutX="504.0" layoutY="245.0"
mnemonicParsing="false" text="Enable">
            <toggleGroup>
               <ToggleGroup fx:id="repeatGroup" />
            </toggleGroup>
         </RadioButton>
         <RadioButton fx:id="repeatRadioFalse" layoutX="504.0"
layoutY="270.0" mnemonicParsing="false" text="Disable"
toggleGroup="$repeatGroup" />
        </AnchorPane>
    </Tab>
   <Tab closable="false" text="Calendar">
     <content>
       <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="180.0"
prefWidth="200.0">
            <children>
                  <DateTimePicker fx:id="fromField" layoutX="425.0"
layoutY="85.0" prefHeight="25.0" prefWidth="155.0" />
                  <DateTimePicker fx:id="toField" layoutX="425.0"
layoutY="140.0" prefHeight="25.0" prefWidth="155.0" />
                  <Label layoutX="425.0" layoutY="65.0" prefHeight="17.0"
prefWidth="35.0" text="From:" />
```

```
                  <Label layoutX="425.0" layoutY="120.0" prefHeight="17.0"
prefWidth="22.0" text="To:" />
                  <Button layoutX="480.0" layoutY="183.0"
mnemonicParsing="false" onMouseClicked="#calendarButtonAction"
prefWidth="45.0" text="Filter" />
               <TableView fx:id="calendarTable" layoutX="37.0" layoutY="14.0"
onMouseClicked="#selectColumn" prefHeight="337.0" prefWidth="380.0"
AnchorPane.bottomAnchor="20.0" AnchorPane.leftAnchor="25.0"
AnchorPane.rightAnchor="193.0" AnchorPane.topAnchor="15.0">
                  <columns>
                     <TableColumn fx:id="titleCalendarColumn"
maxWidth="310.0" minWidth="70.0" prefWidth="190.0" text="Title" />
                     <TableColumn fx:id="timeCalendarColumn" maxWidth="310.0"
minWidth="70.0" prefWidth="190.0" text="Time" />
                  </columns>
                  <columnResizePolicy>
                     <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
                  </columnResizePolicy>
               </TableView>
               <Button layoutX="475.0" layoutY="220.0"
mnemonicParsing="false" onMouseClicked="#refreshButtonAction" text="Refresh"
/>
            </children>
         </AnchorPane>
     </content>
   </Tab>
</TabPane>
```

View.java

```
package bublyk.app.organaizer.view;

import bublyk.app.organaizer.controller.Controller;
import bublyk.app.organaizer.model.*;
import javafx.beans.property.SimpleObjectProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.VBox;
import org.apache.log4j.Logger;
import tornadofx.control.DateTimePicker;

import java.io.IOException;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

public abstract class View {
    protected final static DateTimeFormatter DATE_TIME_FORMATTER =
DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss");

    protected final static int MIN_SPINNER_VALUE = 0;
    protected final static int MAX_SPINNER_VALUE = Integer.MAX_VALUE;
```

```
    protected final static int INIT_SPINNER_VALUE = 0;

    protected LocalDateTime cachedFromField;
    protected LocalDateTime cachedToField;

    protected final AbstractTaskList list =
TaskListFactory.createTaskList(ListTypes.types.ARRAY);

    public final Logger logger = Logger.getLogger(View.class);

    @FXML
    protected TableView<Task> mainTable;
    @FXML
    protected TableColumn<Task, String> titleMainColumn;
    @FXML
    protected TableColumn<Task, String> timeMainColumn;
    @FXML
    protected TableColumn<Task, String> startMainColumn;
    @FXML
    protected TableColumn<Task, String> endMainColumn;
    @FXML
    protected TableColumn<Task, String> activeMainColumn;
    @FXML
    protected TableColumn<Task, String> intervalMainColumn;

    @FXML
    protected TextField titleField;
    @FXML
    protected Label startTimeLabel;
    @FXML
    protected DateTimePicker startTimeField;
    @FXML
    protected Label endTimeLabel;
    @FXML
    protected DateTimePicker endTimeField;
    @FXML
    protected Label intervalLabel;
    @FXML
    protected Spinner<Integer> intervalField;

    @FXML
    protected ToggleGroup activateGroup;
    @FXML
    protected RadioButton activeRadioTrue;
    @FXML
    protected RadioButton activeRadioFalse;
    @FXML
    protected ToggleGroup repeatGroup;
    @FXML
    protected RadioButton repeatRadioTrue;
    @FXML
    protected RadioButton repeatRadioFalse;

    @FXML
    protected TableView<View.CalendarTableHelper> calendarTable;
    @FXML
```

```
    private TableColumn<View.CalendarTableHelper, String>
titleCalendarColumn;
    @FXML
    private TableColumn<View.CalendarTableHelper, String> timeCalendarColumn;

    @FXML
    protected DateTimePicker fromField;
    @FXML
    protected DateTimePicker toField;

    protected void initializeView() {
        titleMainColumn.setCellValueFactory(new
PropertyValueFactory<>("title"));
        timeMainColumn.setCellValueFactory(param -> {
            Task task = param.getValue();
            LocalDateTime time = task.getTime();
            return new
SimpleObjectProperty<>(time.format(DATE_TIME_FORMATTER));
        });
        startMainColumn.setCellValueFactory(param -> {
            Task task = param.getValue();
            LocalDateTime time = task.getStartTime();
            return new
SimpleObjectProperty<>(time.format(DATE_TIME_FORMATTER));
        });
        endMainColumn.setCellValueFactory(param -> {
            Task task = param.getValue();
            LocalDateTime time = task.getEndTime();
            if (task.isRepeated()) {
                return new
SimpleObjectProperty<>(time.format(DATE_TIME_FORMATTER));
            } else {
                return new SimpleObjectProperty<>("-");
            }
        });
        activeMainColumn.setCellValueFactory(param -> {
            Task task = param.getValue();
            boolean active = task.isActive();
            String str = active ? "+" : "-";
            return new SimpleObjectProperty<>(str);
        });
        intervalMainColumn.setCellValueFactory(param -> {
            Task task = param.getValue();
            int interval = task.getRepeatInterval();
            String str = interval == 0 ? "-" : Integer.toString(interval);
            return new SimpleObjectProperty<>(str);
        });
        repeatGroup.selectedToggleProperty().addListener((observableValue,
toggle, t1) -> {
            if (repeatGroup.getSelectedToggle() == repeatRadioFalse) {
                startTimeLabel.setText("Time:");
                intervalLabel.setVisible(false);
                intervalField.setVisible(false);
                endTimeLabel.setVisible(false);
                endTimeField.setVisible(false);
            } else if (repeatGroup.getSelectedToggle() == repeatRadioTrue) {
```

```
                startTimeLabel.setText("Start time:");
                intervalLabel.setVisible(true);
                intervalField.setVisible(true);
                endTimeLabel.setVisible(true);
                endTimeField.setVisible(true);
            }
        });
        intervalField.setValueFactory(new
SpinnerValueFactory.IntegerSpinnerValueFactory(MIN_SPINNER_VALUE,
MAX_SPINNER_VALUE, INIT_SPINNER_VALUE));
        titleCalendarColumn.setCellValueFactory(new
PropertyValueFactory<>("titles"));
        timeCalendarColumn.setCellValueFactory(param -> {
            Controller.CalendarTableHelper helper = param.getValue();
            LocalDateTime time = helper.getTime();
            return new
SimpleObjectProperty<>(time.format(DATE_TIME_FORMATTER));
        });
        repeatRadioFalse.setSelected(true);
        dateStyle();
    }

    protected void loadMainTable() {
        ObservableList<Task> taskList = FXCollections.observableArrayList();
        for (Task temp : list) {
            taskList.add(temp);
        }
        mainTable.setItems(taskList);
    }

    protected void loadCalendarTable() {
        SortedMap<LocalDateTime, Set<Task>> map = Tasks.calendar(list,
cachedFromField, cachedToField);
        List<CalendarTableHelper> calendarTableHelperList = new
ArrayList<>(map.size());
        for (Map.Entry<LocalDateTime, Set<Task>> entry : map.entrySet()) {
            calendarTableHelperList.add(new
View.CalendarTableHelper(entry.getKey(), entry.getValue()));
        }
        ObservableList<View.CalendarTableHelper> calendar =
FXCollections.observableList(calendarTableHelperList);
        calendarTable.setItems(calendar);
    }

    protected static class CalendarTableHelper {
        private LocalDateTime time;
        private String titles;
        private Set<Task> tasks;

        public CalendarTableHelper(LocalDateTime time, Set<Task> tasks) {
            this.time = time;
            this.tasks = tasks;
            transformToString();
        }

        private void transformToString() {
```

```
            StringBuilder str = new StringBuilder();
            for (Task temp : tasks) {
                str.append(temp.getTitle()).append("\n");
            }
            titles = str.toString();
        }

        public void setTime(LocalDateTime time) {
            this.time = time;
        }

        public void setTitles(String titles) {
            this.titles = titles;
        }

        public void setTasks(Set<Task> tasks) {
            this.tasks = tasks;
        }

        public LocalDateTime getTime() {
            return time;
        }

        public String getTitles() {
            return titles;
        }

        public Set<Task> getTasks() {
            return tasks;
        }
    }

    @FXML
    private void selectColumn() {
        if (mainTable.getSelectionModel().getSelectedItem() != null) {
            Task temp = mainTable.getSelectionModel().getSelectedItem();
            titleField.setText(temp.getTitle());
            if (temp.isActive()) {
                activeRadioTrue.setSelected(true);
            } else {
                activeRadioFalse.setSelected(true);
            }
            if (temp.isRepeated()) {
                repeatRadioTrue.setSelected(true);
                startTimeField.setDateTimeValue(temp.getStartTime());
                endTimeField.setDateTimeValue(temp.getEndTime());

intervalField.getValueFactory().setValue(temp.getRepeatInterval());
            } else {
                repeatRadioFalse.setSelected(true);
                startTimeField.setDateTimeValue(temp.getTime());
            }
        }
    }

    protected void unselectColumn() {
```

```
        mainTable.getSelectionModel().clearSelection();
        titleField.clear();
        startTimeField.setDateTimeValue(null);
        endTimeField.setDateTimeValue(null);
        intervalField.getValueFactory().setValue(INIT_SPINNER_VALUE);
        activeRadioTrue.setSelected(false);
        activeRadioFalse.setSelected(false);
    }

    public void showError(String headerMessage, String contentMessage) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.initOwner(titleField.getScene().getWindow());
        alert.setTitle("Error");
        alert.setHeaderText(headerMessage);
        alert.setContentText(contentMessage);
        alert.showAndWait();
    }

    public void showError(Exception exception) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.initOwner(titleField.getScene().getWindow());
        alert.setTitle("Error");
        alert.setHeaderText(null);
        alert.setContentText(exception.getMessage());
        alert.showAndWait();
    }

    public void showError(String headerMessage, Exception exception) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.initOwner(titleField.getScene().getWindow());
        alert.setTitle("Error");
        alert.setHeaderText(headerMessage);
        Label label = new Label("Stack Trace:");
        VBox alertContent = new VBox();
        TextArea textArea = new TextArea();
        String contentMessage;
        try (StringWriter sw = new StringWriter();
             PrintWriter pw = new PrintWriter(sw)) {
            exception.printStackTrace(pw);
            contentMessage = sw.toString();
            textArea.setText(contentMessage);
        } catch (IOException e) {
            logger.error("Alert stack trace error.", e);
        }
        textArea.setEditable(false);
        alertContent.getChildren().addAll(label, textArea);
        alert.getDialogPane().setContent(alertContent);
        alert.showAndWait();
    }

    protected void dateStyle() {
        startTimeField.setDateTimeValue(null);
        endTimeField.setDateTimeValue(null);
        toField.setDateTimeValue(null);
        fromField.setDateTimeValue(null);
        startTimeField.setFormat("dd.MM.yyyy HH:mm:ss");
```

```
        endTimeField.setFormat("dd.MM.yyyy HH:mm:ss");
        toField.setFormat("dd.MM.yyyy HH:mm:ss");
        fromField.setFormat("dd.MM.yyyy HH:mm:ss");
    }
}
```

Controller

Controller.java

```
package bublyk.app.organaizer.controller;

import bublyk.app.organaizer.RunOrganizer;
import javafx.fxml.FXML;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import bublyk.app.organaizer.model.AbstractTaskList;
import bublyk.app.organaizer.view.View;

import java.util.*;

public class Controller extends View {
    protected final Image icon = new
Image(Objects.requireNonNull(RunOrganizer.class.getResource("OrganizerIcon.pn
g")).toExternalForm());

    protected final LoadController loadController = new LoadController(this);
    protected final NotificatorController notificator = new
NotificatorController(this);

    @FXML
    protected void initialize() {
        initializeView();
        loadController.readingData();
        notificator.runNotificator();
        loadMainTable();
    }

    protected boolean textFieldIsEmpty(TextField textField) {
        if (textField.getText() == null) {
            return true;
        }
        else {
            return textField.getText().isEmpty();
        }
    }

    public void writingData() {
        loadController.writingData();
    }

    public AbstractTaskList getTaskList() {
       return list;
    }
```

```
}
```

ActionController.java

```
package bublyk.app.organaizer.controller;

import javafx.fxml.FXML;
import bublyk.app.organaizer.model.Task;

public class ActionController extends Controller {

    @FXML
    private void addButtonAction() {
        Task temp;
        if (activateGroup.getSelectedToggle() != null) {
            try {
                if (!textFieldIsEmpty(titleField)) {
                    if (repeatRadioTrue.isSelected()) {
                        temp = new Task(titleField.getText(),
startTimeField.getDateTimeValue(), endTimeField.getDateTimeValue(),
intervalField.getValue());
                    } else if (repeatRadioFalse.isSelected()) {
                        temp = new Task(titleField.getText(),
startTimeField.getDateTimeValue());
                    } else {
                        throw new IllegalArgumentException("Select whether
the task is repeated.");
                    }
                    temp.setActive(activeRadioTrue.isSelected());
                    list.add(temp);
                    logger.info("Task was added.");
                    unselectColumn();
                    notificator.updateNotificator(list);
                    loadMainTable();
                } else {
                    throw new IllegalArgumentException("Title filed must be
filled in.");
                }
            } catch (IllegalArgumentException e) {
                showError(e);
                logger.error("Add task error.", e);
            }
        } else {
            showError("Unable to add task.", "Please, select an activity
status for the task.");
        }
    }

    @FXML
    private void editButtonAction() {
        if (mainTable.getSelectionModel().getSelectedItem() != null) {
            Task temp = mainTable.getSelectionModel().getSelectedItem();
            if (activateGroup.getSelectedToggle() != null) {
                try {
                    if (!textFieldIsEmpty(titleField)) {
                        temp.setTitle(titleField.getText());
```

```
                        if (repeatRadioTrue.isSelected()) {
                            temp.setTime(startTimeField.getDateTimeValue(),
endTimeField.getDateTimeValue(), intervalField.getValue());
                        } else {
                            temp.setTime(startTimeField.getDateTimeValue());
                        }
                        temp.setActive(activeRadioTrue.isSelected());
                        logger.info("Task was edited.");
                        unselectColumn();
                        notificator.updateNotificator(list);
                        mainTable.refresh();
                    } else {
                        throw new IllegalArgumentException("Title filed must
be filled in.");
                    }
                } catch (IllegalArgumentException e) {
                    showError(e);
                    logger.error("Edit task error.", e);
                }
            } else {
                showError("Unable to edit task.", "Please, select an activity
status for the task.");
            }
        } else {
            showError("Unable to edit task.", "Please, select a row to
edit.");
        }
    }

    @FXML
    private void removeButtonAction() {
        if (mainTable.getSelectionModel().getSelectedItem() != null) {
            Task temp = mainTable.getSelectionModel().getSelectedItem();
            list.remove(temp);
            logger.info("Task was removed.");
            unselectColumn();
            notificator.updateNotificator(list);
            loadMainTable();
        } else {
            showError("Unable to remove task.", "Please, select a row to
remove.");
        }
    }

    @FXML
    private void resetButtonAction() {
        unselectColumn();
        repeatRadioTrue.setSelected(false);
        repeatRadioFalse.setSelected(true);
        loadMainTable();
    }

    @FXML
    private void calendarButtonAction() {
        try {
            cachedFromField = fromField.getDateTimeValue();
```

```
            cachedToField = toField.getDateTimeValue();
            loadCalendarTable();
            logger.info("Calendar was loaded.");
        } catch (IllegalArgumentException e) {
            showError(e);
            logger.error("Calendar error.", e);
        }
    }

    @FXML
    private void refreshButtonAction() {
        calendarTable.getSelectionModel().clearSelection();
        if (cachedFromField != null && cachedToField != null) {
            fromField.setDateTimeValue(cachedFromField);
            toField.setDateTimeValue(cachedToField);
            loadCalendarTable();
        }
    }
}
```

LoadController.java

```
package bublyk.app.organaizer.controller;

import bublyk.app.organaizer.model.TaskIO;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class LoadController {
    private final Controller controller;
    private final static String PATH_TO_LIST = "data/tasks.bin";

    protected LoadController(Controller controller) {
        this.controller = controller;
    }

    protected void readingData() {
        Path path = Paths.get(PATH_TO_LIST);
        if (Files.notExists(path)) {
            try {
                if (Files.notExists(path.getParent())) {
                    Files.createDirectory(path.getParent());
                }
                Files.createFile(path);
            } catch (IOException e) {
                controller.logger.error("Reading data error.", e);
                controller.showError("Unsuccessful reading from file \"" +
path + "\".", e);
            }
        } else {
            TaskIO.readBinary(controller.getTaskList(), path.toFile());
        }
    }
```

```
    protected void writingData() {
        Path path = Paths.get(PATH_TO_LIST);
        if (Files.notExists(path)) {
            try {
                if (Files.notExists(path.getParent())) {
                    Files.createDirectory(path.getParent());
                }
                Files.createFile(path);
            } catch (IOException e) {
                controller.logger.error("Writing data error.", e);
                controller.showError("Unsuccessful writing to file \"" + path
+ "\".", e);
            }
        }
        TaskIO.writeBinary(controller.getTaskList(), path.toFile());
    }

}
```

NotificatorController.java

```
package bublyk.app.organaizer.controller;

import bublyk.app.organaizer.notification.Notificator;
import bublyk.app.organaizer.model.Task;

/**
 * The class that manages notifications.
 */
public class NotificatorController {
    private final Notificator notificator;
    private final Controller controller;

    /**
     * The constructor that sets controller which will manage this
controller.
     * @param controller controller that manages this object.
     */
    protected NotificatorController(Controller controller) {
        this.controller = controller;
        notificator = new Notificator(controller.getTaskList(),
controller.icon);
    }

    /**
     * The method that runs daemon thread of notification.
     */
    protected void runNotificator() {
        notificator.setDaemon(true);
        notificator.start();
        controller.logger.debug("Notification is running.");
    }

    /**
     * The method that stops thread of notification.
     */
    protected void stopNotificator() {
```

```
        if (notificator.isAlive()) {
            notificator.stop();
        }
    }

    /**
     * The method that updates task list for notification.
     * @param list task list which notificator will operate.
     */
    public void updateNotificator(Iterable<Task> list) {
        notificator.setTasksList(list);
    }

    /**
     * The method that changes the time for how long the notification should
arrive before the start of the task.
     * @param seconds time in seconds when the notification about the task
will come.
     */
    protected void updateNotificationTime(int seconds) {
        notificator.setNotificationTime(seconds);
    }

}
```