МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Крос-платформний додаток контролю фінансів»
здобувача групи ІН - 91 Рудкіна Віталія Владиславовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Віталій РУДКІН
(підпис)

Керівник,
асистент кафедри комп'ютерних
наук, кандидат фізико-математичних
наук Олександр ВЛАСЕНКО _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-91 Рудкіна Віталія Владиславовича

1. Тема роботи: «Крос-платформний додаток контролю фінансів»
затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для створення крос-платформних додатків. 3) Розробка крос-платформного додатку контролю фінансів. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій, що використовуються для створення крос-платформних додатків.*		
3	*Розробка крос-платформного додатку контролю фінансів.*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____

АНОТАЦІЯ

Записка: 81 стр., 20 рис., 20 використаних джерел, 2 додатки

Обґрунтування актуальності теми роботи – тема бакалаврської роботи є актуальною, оскільки у сучасному світі все більше людей цінують управління своїми фінансами. Завдяки крос-платформному додатку, який працює на різних пристроях та платформах, люди матимуть зручний спосіб контролювати та аналізувати свої доходи, витрати та інвестиції.

Об’єкт дослідження – процес розробки крос-платформного додатку.

Мета роботи – розробка та імплементація крос-платформного додатка контролю фінансів, який надасть користувачам зручність та ефективність у керуванні їх фінансами на різних пристроях та платформах.

Методи дослідження – технології розробки крос-платформних додатків, інструменти для створення веб інтерфейсів, розробка архітектури системи, тестування системи.

Результати – створено функціональний крос-платформний додаток контролю фінансів, який працює на різних пристроях та платформах, таких як мобільні телефони, планшети та персональні комп'ютери. Додаток має широкий набір функціональних можливостей, які дозволяють користувачам відстежувати доходи та витрати, створювати бюджети, аналізувати фінансові показники. Результати тестування та порівняння з існуючими платформою-залежними додатками підтверджують ефективність крос-платформного підходу у розробці додатків контролю фінансів.

КРОС-ПЛАТФОРМНИЙ ДОДАТОК, КОНТРОЛЬ ФІНАНСІВ, ДОХОДИ, ВИТРАТИ, БЮДЖЕТ, ELECTRON, ANGULAR

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі, де фінансова стабільність та ефективне управління грошима мають велике значення для багатьох людей, контроль фінансів стає необхідною складовою успіху та забезпечення фінансової безпеки. Однак, ручний контроль та відстеження фінансових операцій можуть займати багато часу та бути неефективними. Тому створення крос-платформного додатка, що надає зручні та потужні інструменти для контролю фінансів, має великий практичний інтерес.

Об’єкт дослідження. Об'єктом дослідження є процес контролю фінансів та розробка крос-платформного додатка, який надає зручні та ефективні можливості для відстеження та управління фінансовими ресурсами. Дослідження включає аналіз існуючих рішень та вибір оптимальних технологій для реалізації додатка.

Предмет дослідження. Предметом дослідження є крос-платформний додаток контролю фінансів, який базується на фреймворку Electron та фронтенд-фреймворку Angular. Додаток дозволяє користувачам ефективно відстежувати свої фінансові ресурси, аналізувати витрати та прибутки, створювати категорії та отримувати звіти за допомогою інтуїтивно зрозумілого та зручного інтерфейсу.

Гіпотеза. Припускається, що розроблений крос-платформний додаток контролю фінансів забезпечить користувачам зручні та потужні інструменти для ефективного контролю та управління їх фінансовими ресурсами. Додаток буде простим у використанні, має бути стабільним та надійним, а також забезпечувати захист персональних фінансових даних.

Новизна. Основна новизна даного дослідження полягає в поєднанні фреймворку Electron та фронтенд-фреймворку Angular для розробки крос-платформного додатка контролю фінансів. Комбінація цих технологій надає можливість створювати потужні та функціональні інструменти для фінансового управління з використанням сучасних веб-технологій.

Структура. Дана робота складається з наступних розділів:

1. Вступ
2. Інформаційний огляд
3. Вибір методів рішення задачі
4. Практична реалізація
5. Висновки
6. Список використаних джерел
7. Додатки

У розділі "Інформаційний огляд" буде проведений аналіз існуючих рішень та розглянуті важливі аспекти контролю фінансів. У розділі "Вибір методів рішення задачі" будуть розглянуті технології, фреймворки та інструменти, використані при розробці додатка. Розділ "Практична реалізація" детально описуватиме структуру та функціональні можливості крос-платформного додатка контролю фінансів. Висновки підсумують отримані результати, а розділ "Список використаних джерел" міститиме перелік літературних джерел та документації, використаних під час роботи над бакалаврською роботою.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд фінансового управління

Управління фінансами є важливим процесом у житті людини або компанії для підтримання життєдіяльності, росту, розвитку ті досяганні цілей. Основними складовими фінансової діяльності є:

– Планування;

– Організація;

– Спрямування;

– Контроль.

Найголовнішим завданням при управлінні фінансами є раціональне та ефективне використання фінансів(ресурсами). Для цього є необхідним набуття вміння аналізувати та приймати зважені та обґрунтовані рішення щодо фінансів, а також вміти управляти фінансовими ризиками.[17]

Крім того, фінансовий менеджмент передбачає підготовку фінансової звітності та звітів, які надають точну та своєчасну інформацію про фінансові результати діяльності організації. Ці звіти можуть бути використані для оцінки фінансового стану організації та прийняття обґрунтованих рішень щодо майбутніх фінансових показників[3].

Таким чином, фінансовий менеджмент є критично важливим аспектом успіху будь-якої організації або приватної особи. Ефективне управління фінансовими ресурсами може допомогти досягти бажаних цілей і завдань, мінімізуючи при цьому фінансові ризики.

1.2 Крос-платформні додатки

Крос-платформні програми, також відомі як крос-платформне програмне забезпечення або крос-платформне ПЗ, – це програми, які можуть працювати на декількох операційних системах або платформах. Це означає, що

один і той самий додаток можна використовувати на різних пристроях, таких як комп'ютери, смартфони та планшети.

Розробка крос-платформних додатків набуває все більшого значення в останні роки, оскільки кількість операційних систем і пристроїв зростає. Оскільки існує так багато різних операційних систем і пристроїв, розробникам може бути важко створювати окрему версію програми для кожної системи. Крос-платформні додатки вирішують цю проблему, дозволяючи розробникам створювати один додаток, який може працювати на декількох платформах.

Крос-платформні додатки можна розробляти за допомогою різних мов програмування та фреймворків, таких як Java, C++. Одними з найпопулярніших інструментів розробки крос-платформних додатків є Xamarin, React Native та ElectronJs.[1]

Переваги розробки крос-платформних додатків включають скорочення часу та вартості розробки та зручність використання користувачем, а також більш узгоджений користувацький досвід на різних пристроях та платформах. Оскільки попит на крос-платформні додатки продовжує зростати, все більше розробників, ймовірно, звертатимуться до цих інструментів для створення додатків, які можуть бути використані широкою аудиторією.[2]

1.3 Функції додатку для контролю фінансів

Додаток для управління фінансами – це застосунок, розроблений, аби допомогти користувачеві більш ефективно управляти своїми фінансами. Деякі з ключових особливостей і функцій програми для управління фінансами включають в себе наступні:

Створення бюджету та відстеження витрат: важливою функцією додатка є можливість створювати і відстежувати бюджет. Користувачі можуть встановлювати ліміти витрат для різних категорій, таких як продукти, розваги та таксі, і відслідковувати свої витрати, щоб переконатися, що ліміт не перевищено.

Синхронізація та відстеження рахунків: значна кількість додатків для управління фінансами дозволяють користувачам підключати свої банківські рахунки, кредитні картки та інші фінансові рахунки до додатка. Це полегшує відстеження всіх фінансів, своєчасне та швидке оновлення інформації за ними та звісно спрощує контроль за рахунок єдиного місця для відстеження.

Відстеження інвестицій: деякі додатки мають у наборі функціоналу відстеження інвестиційних портфелів, наданні змістовних інформаційних графіків щодо кожної інвестиції у режимі реального часу.

Постановка фінансових цілей: значна кількість додатків мають можливість допомагати досягати певних фінансових цілей за рахунок створення окремих скриньок для накопичення заощаджень. Наприклад, заощадити на придбання автомобіля, будинку, відкриття власного бізнесу тощо. Додаток може допомогти користувачам відстежувати свій прогрес і надавати інформацію та рекомендації, які допоможуть їм досягти поставлених цілей.

Загалом, додаток для управління фінансами може бути потужним інструментом для тих, хто хоче взяти під контроль свої фінанси. Надаючи інформацію про витратні звички, відстежуючи витрати та встановлюючи фінансові цілі, ці програми можуть допомогти користувачам приймати кращі фінансові рішення та досягти більшої фінансової стабільності.

1.4 Інтерфейс додатку

Користувацький інтерфейс додатку для управління фінансами має вирішальне значення для її зручності та ефективності. Інтуїтивно зрозумілий і зручний інтерфейс може полегшити користувачам відстеження своїх фінансів і досягнення фінансових цілей.

Деякі ключові характеристики хорошого користувацького інтерфейсу для програми фінансового менеджменту включають в себе наступні:

Інформаційна панель: Інформаційна панель є головним екраном програми і повинна надавати користувачам швидкий огляд їхнього фінансового стану. Вона повинна відображати важливу інформацію, таку як залишки на рахунках, витрати та доходи, у чіткій та стислій формі.

Навігація: Меню навігації має бути максимально простим і зручним у використанні. Користувачі повинні мати можливість отримати доступ до всіх функцій програми з меню за допомогою мінімальної кількості кліків.

Візуалізація даних: Візуалізація даних відіграє важливу роль у представленні структурованих даних користувачеві . Діаграми, графіки та інші візуальні засоби можуть допомогти користувачам легше розуміти свої фінансові дані та приймати кращі рішення.

Особисті налаштування: Користувачі повинні мати можливість налаштовувати додаток відповідно до своїх конкретних потреб. Серед основних ідей можна відокремити: вибір рахунків, які відображатимуться на інформаційній панелі, налаштування категорій витрат або коригування бюджетних лімітів.

Сповіщення: Сповіщення можуть допомогти користувачам завжди бути повідомленим про зміни у фінансових рахунках, наближенні до лімітів тощо.

Загалом, хороший користувацький інтерфейс програми для управління фінансами повинен бути інтуїтивно зрозумілим, простим у використанні та надавати користувачам інформацію, необхідну для прийняття обґрунтованих фінансових рішень. Дотримуючись вище наведених принципів можна створити конкуруючий продукт з відмінною якістю для кінцевого користувача.

2 ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

2.1 Вибір технологій для розробки крос-платформного додатку

При виборі технологіїй для розробки крос-платформного додатку потрібно врахувати низку факторів, які мають вплив на швидкість розробки, легкість підтримки додатку, продуктивність тощо.

Порівняємо найбільш популярні технології для розробки крос-платформних додатків[5][4] (див. Таблиця 2.1 та Таблиця 2.2)

Таблиця 2.1 – Порівняльна таблиця переваг та недоліків популярних технологій для розробки крос-платформних додатків

Технологія	Опис	Переваги	Недоліки
Electron	Базується на Node.js та Chromium, використовує веб-технології для розробки десктопних додатків	– Широкий вибір налаштувань та додаткових модулів – Можливість використовувати веб-технології – Підтримка різних платформ – Забезпечує високу продуктивність та швидкість розробки	– Великий розмір додатку – Високі вимоги до ресурсів – Не підходить для мобільних додатків
React Native	Базується на React, використовує JavaScript для розробки мобільних додатків	– Висока продуктивність – Можливість розробки під Android та iOS – Наявність великої кількості готових компонентів та бібліотек	– Є проблеми з сумісністю зі сторонніми бібліотеками – Обмежена підтримка більш нових версій Android та iOS

		– Швидкість розробки	
Flutter	Базується на Dart, використовує для розробки мобільних додатків та веб-інтерфейсів	– Висока продуктивність – Швидкість розробки – Можливість використовувати Dart – Наявність великої кількості готових компонентів та бібліотек	– Не підтримує Web-розробку – Є проблеми зі стабільністю на старих пристроях – Обмежена кількість бібліотек – Не дуже багата екосистема
Xamarin	Основана на .NET Framework, використовує C# для розробки мобільних додатків та десктопних застосунків	– Швидкість розробки – Гарна підтримка Microsoft – Багата екосистема – Велика кількість готових компонентів	– Не дуже ефективна робота на старих пристроях – Відсутність підтримки деяких платформ

Таблиця 2.2 – Порівняльна таблиця використовуваних технологій та складності їх вивчення

Технологія	Мова програмування	UI-фреймворк	Складність вивчення
Electron	Javascript/Typescript	React, Angular, Vue	Середня
React Native	Javascript/Typescript	Flutter SDK	Висока
Flutter	Dart	React Native	Середня
Xamarin	C#, F#	Xamarin.Forms	Висока

Виходячи з порівняльних таблиць (Таблиця 2.1 та Таблиця 2.2) найкращим вибором є Electron, оскільки наявні знання та досвід з цією технологією значно зменшать час розробки та дозволять реалізувати більш складний функціонал з мінімальною кількістю або відсутністю дефектів, основними можливостями додатку буде[14]:

– Додатки будуть працювати на Windows, Linux та MacOS, а тому матимуть широкий попит через доступність на найпопулярніших платформах

– Також додаток матиме високу продуктивність, оскільки базується на Chromium

– Легкість розробки скорочує час від планування додатку, побудови архітектури до кінцевої точки – випуску першої стабільної версії для кінцевого користувача.

2.2 Вибір архітектури додатку

Одним з можливих варіантів архітектури додатку на основі Electron та Angular є розподіл додатка на дві частини front-end та back-end. Front-end частина буде відповідати за відображення інтерфейсу для користувача та взаємодії з ним, з використанням Angular. Back-end буде відповідати за зберігання даних, їх обробку та надання front-end частині, а також взаємодія за допомогою Electron API з платформою на якій запущено додаток.[6]

Відповідно до цього, front-end з точки зору найкращих практик, можна буде поділити на окремі компоненти, для оптимізації та групування схожих візуальних елементів. Наприклад, модуль для відображення та модуль для створення нового профілю для управління фінансами, модуль для відображення статистики за певними даними з вибором зручної діаграми або графіка тощо. Для оптимізації завантаження компонентів доцільним є використання вбудованої до Angular технології “Lazy loading”, що зменшує навантаження на систему завантажуючи модулі та компоненти лише у разі їх виклику або використання[9].

Back-end частину можна спроектувати відповідно до багатошарової архітектури, у якій роль API (Router layer) візьмуть на себе канали для прослуховування викликаної події та надалі виклику відповідних сервісів. Роль сервісів (Service layer) залишиться незмінною – сервіси матимуть доступ до шару доступу до даних та інших сервісів з метою реалізації бізнес логіки

back-end’у. Та безпосередньо шар доступу до даних (Data access layer), який матиме змогу роботи з провайдером зберігання даних, який буде розглянуто далі.

Зв'язок між Electron та Angular зі збереженням рекомендацій з безпеки, які описані у документації Electron[], є використання IPC, що є реалізацією патерну RPC(Remote Procedure Call). Цей патерн дозволяє викликати функції які знаходяться в іншому процесі. В Electron буде використовуватись модуль ipcMain, який буде забезпечувати взаємодію між головним та процесом рендерингу (візуалізації). Натомість в Angular відповідно буде використовуватись модуль ipcRenderer, для впровадження аналогічної функціональності – прослуховування подій з викликом відповідних функції та викликом функцій у іншому процесі.[12]

Задля встановлення безпечного зв'язку між процесом рендерингу та головним доцільним є використання preload файлу та contextBridge, які забезпечать надання API до процесу рендерингу, для взаємодії до різних функції. Єдиним обмеженням є мануальний контроль того, які саме функції передаються до рендеринг процесу, щоби застережити від можливості зловживання API зі сторони рендеринг процесу.

2.3 Вибір методів зберігання даних

При розробці крос-платформного додатку на Electron для зберігання даних можна використовувати різноманітні методи, такі як збереження в локальну базу даних, використання RESTful API для зв'язку з сервером, або використання electron-store (npm-пакет).

Electron-store – це простий та легкий у використанні модуль для збереження та отримання даних в електронному додатку. Він працює на основі локального сховища ключ-значення та забезпечує автоматичне збереження та відновлення даних між запусками додатку.

Основні переваги використання electron-store для зберігання даних:

– Простота використання та розгортання

– Автоматичне збереження та відновлення даних

– Підтримка асинхронного доступу до даних

– Підтримка роботи з об'єктами та масивами

– Можливість створення різних версій сховища даних з можливістю міграцій для оновлення схеми

Однак, electron-store має певні обмеження, такі як обмежену підтримку запитів та обмеження обсягу даних, що можна зберегти та обмеження типів для зберігання даних. Тому при використанні цього методу зберігання даних потрібно враховувати особливості при реалізації додатку.[13]

2.4 Вибір інструменту для розробки інтерфейсу користувача

Розробка інтерфейсу користувача – це важлива складова процесу створення додатку, яка вимагає уваги та ретельного планування. Один з можливих варіантів для реалізації інтерфейсу користувача в крос-платформному додатку з використанням Electron та Angular – використання Angular Material.

Angular Material – це набір готових компонентів, які розроблені спеціально для використання в Angular додатках. Ці компоненти мають вбудований дизайн та реалізовані компоненти, що сприяє швидкій та зручній розробці інтерфейсу користувача.

Angular Material має кілька переваг для розробки інтерфейсу користувача в крос-платформних додатках з використанням Electron та Angular:

– Зручність: компоненти Angular Material дозволяють розробникам швидко та зручно створювати естетичний та логічний дизайн.

– Адаптивність: компоненти Angular Material дозволяють автоматично пристосовуватися до розміру екрану, що робить їх більш гнучкими для використання в різних пристроях та розмірах екрану.

– Сумісність: Angular Material інтегрується з Angular, тому використання цих компонентів не вимагає додаткового навчання та дозволяє легко додавати нові функції до вашого додатку.

– Можливість особистих налаштувань компонентів та модулів: Angular Material компоненти є відкритими до зміни та перезапису їх відображення, що дозволяє розробнику за необхідності налаштувати компоненти до відповідних вимог до інтерфейсу та функціоналу.

– Широкий вибір завчасно реалізованих компонентів: Angular Material має широкий вибір реалізованих компонентів, наприклад, можливість зміни теми, використання вбудованих компонентів: індикатор завантаження, слайдер, повзунок для вибору значення, поле вводу тощо.

Загалом, використання Angular Material є відмінним вибором для швидкої розробки інтерфейсу та подальшого підтримання з метою додавання нового функціоналу та зміни поточного.[8]

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Розробка архітектури додатку

Оскільки архітектурою додатка було обрано розподілення його на дві частини, back-end та front-end відповідно, то загальний вигляд структури проєкту з відповідних директорій з файлами – *«app»* та *«src/app»* відповідно().

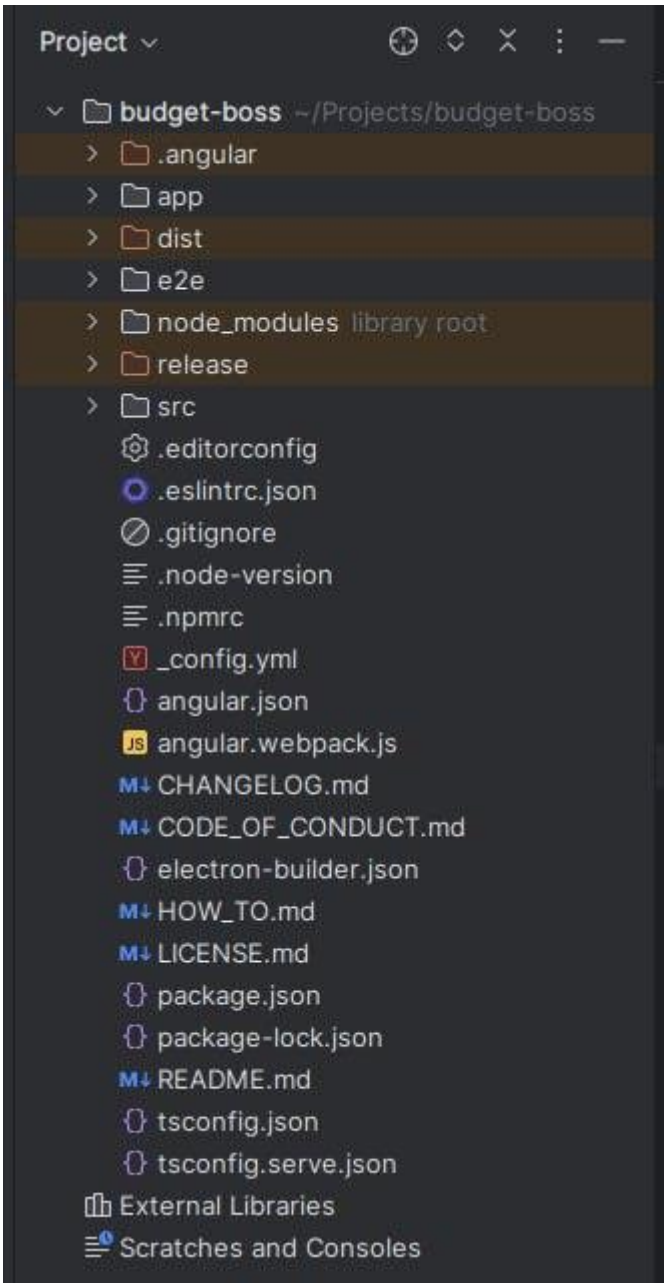
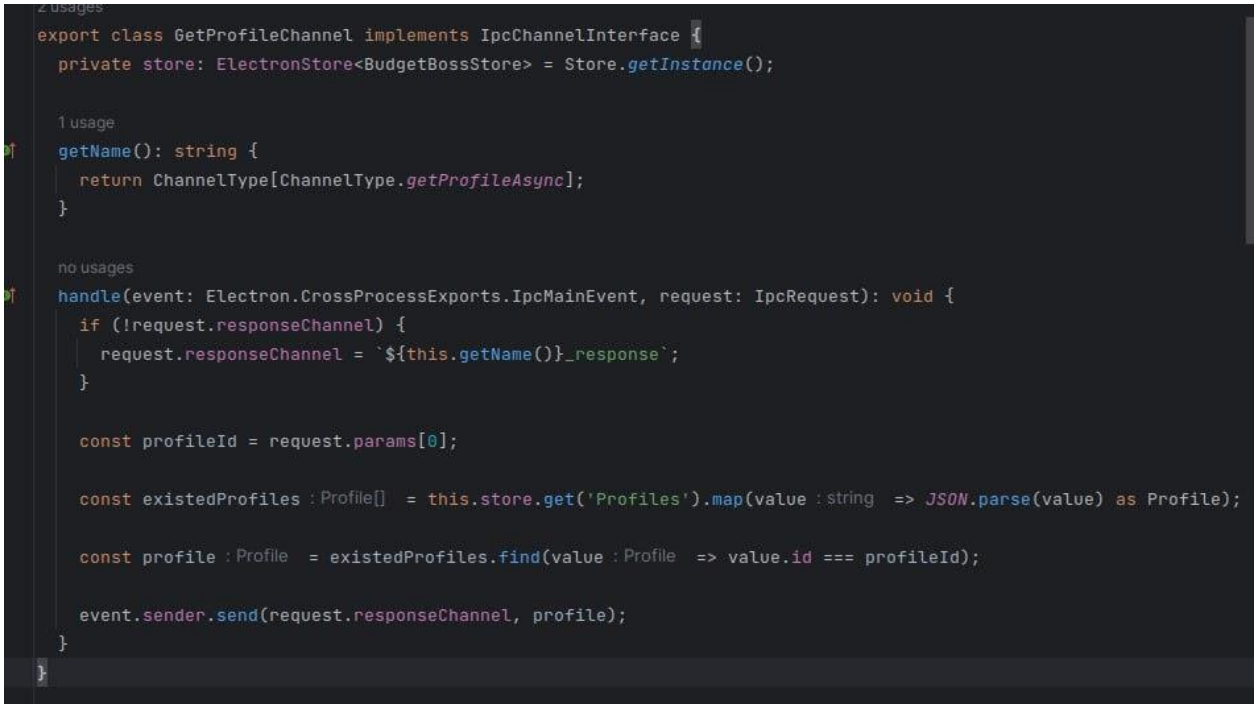


Рисунок 3.1 – Загальна структура проєкту

Back-end частину було реалізовано відповідно до багатошарової архітектури:

— router layer – канали для прослуховування подій



```
2 usages
export class GetProfileChannel implements IpcChannelInterface {
  private store: ElectronStore<BudgetBossStore> = Store.getInstance();

  1 usage
  getName(): string {
    return ChannelType[ChannelType.getProfileAsync];
  }

  no usages
  handle(event: Electron.CrossProcessExports.IpcMainEvent, request: IpcRequest): void {
    if (!request.responseChannel) {
      request.responseChannel = `${this.getName()}_response`;
    }

    const profileId = request.params[0];

    const existedProfiles : Profile[]  = this.store.get('Profiles').map(value : string  => JSON.parse(value) as Profile);

    const profile : Profile  = existedProfiles.find(value : Profile  => value.id === profileId);

    event.sender.send(request.responseChannel, profile);
  }
}
```

Рисунок 3.2 – Приклад реалізації одного з каналів для прослуховування події

— service layer – рівень з необхідними сервісами

— data access layer відповідає сервіс Store, що допомагає зберігати та керувати даними

```
export class Store {
  private static instance: ElectronStore<BudgetBossStore> = null;

  // eslint-disable-next-line @typescript-eslint/no-empty-function
  no usages
  private constructor() {
    // empty ctor
  }

  5+ usages
  public static getInstance(): ElectronStore<BudgetBossStore> {
    if (!Store.instance) {
      Store.instance = new ElectronStore<BudgetBossStore>( options: {
        schema,
        name: 'budget-boss-config',
        clearInvalidConfig: true,
        watch: true,
        beforeEachMigration: (store : Conf<BudgetBossStore> , context : BeforeEachMigrationContext ) : void  => {
          console.log(`[app] [main-config] migrate from ${context.fromVersion} → ${context.toVersion}`);
        },
        migrations: {
        },
      });
    }

    return Store.instance;
  }
}
```

Рисунок 3.3 – Реалізації сховища даних з налаштуваннями

Реалізацією зв'язку двох частину було зроблено за допомогою використання IPC(inter-process communication). Для збереження безпеки між лвома частинами був реалізований сервіс IpcService з необхідними методами для відправки повідомлення та його прийняття відповідно протилежною стороною.[12]

Методи сервісу використовуються у preload файлі, який за допомогою contextBridge надає API для його виклику до частини рендерингу.

```
export type IpcRendererApi = {
  sendPingMessage: (text: string) => Promise<PingPongChannelResponse>;
  StoreApi: {
    getValue: (key: string) => Promise<StoreGetValueChannelResponse>;
    setValue: (key: string, value: any) => Promise<any>;
  };
  UpdatesApi: {
    getAppVersion: () => Promise<AppVersionChannelResponse>;
  };
  ActionsApi: {
    sendAppAction: (action: string) => Promise<any>;
    removeListeners: (channels: ChannelType[]) => Promise<void>;
  };
  ProfilesApi: {
    getProfileAsync: (id: string) => Promise<Profile>;
    getProfilesAsync: () => Promise<Profile[]>;
    addProfileAsync: (profile: Partial<Profile>) => Promise<Profile>;
    deleteProfileAsync: (profile: Profile) => Promise<void>;
  };
};

const ipcService: IpcService = new IpcService();
```

Рисунок 3.4 – Приклад об'явлення інтерфейсів API

```
  StoreApi: {
    getValue: (key: string) =>
      ipcService.send<StoreGetValueChannelResponse>(ChannelType[ ChannelType.storeGetValue ], request: { params: [key]
    setValue: (key: string, value: string) => ipcService.send(ChannelType[ ChannelType.storeSetValue ], request: { par
  },
  sendPingMessage: async (text : string ) : Promise<PingPongChannelResponse>  => ipcService.send<PingPongChannelResponse>(
    responseChannel: 'pong',
    params: [text],
  }),
  ProfilesApi: {
    getProfileAsync: (id: string): Promise<Profile> =>
      ipcService.send<Profile>(ChannelType[ChannelType.getProfileAsync], request: { params: [id] }),
    getProfilesAsync: (): Promise<Profile[]> =>
      ipcService.send<Profile[]>(ChannelType[ChannelType.getProfilesAsync]),
    addProfileAsync: (profile: Partial<Profile>): Promise<Profile> =>
      ipcService.send<Profile>(ChannelType[ChannelType.addProfileAsync], request: { params: [profile] }),
    deleteProfileAsync: (profile: Profile): Promise<void> =>
      ipcService.send<void>(ChannelType[ChannelType.deleteProfileAsync], request: { params: [profile] }),
  },
};
```

Рисунок 3.5 – Приклад реалізації методів API з використанням сервісу для виклику необхідної події

```
export type NodeApi = {
  path: {
    join: (...paths: string[]) => string;
    dirname: (p: string) => string;
    normalize: (p: string) => string;
    basename: (p: string, extName?: string) => string;
  };
};

const exposedNodeApi: NodeApi = {
  path: {
    dirname: (p: string) => path.dirname(p),
    join: (...paths: string[]) => path.join(...paths),
    normalize: (p: string) => path.normalize(p),
    basename: (p: string, extName?: string) => path.basename(p, extName),
  },
};

contextBridge.exposeInMainWorld( apiKey: 'ipcRendererApi', exposedIpcRendererApiApi);
contextBridge.exposeInMainWorld( apiKey: 'loggerApi', exposedLoggerApi);
contextBridge.exposeInMainWorld( apiKey: 'nodeApi', exposedNodeApi);
```

Рисунок 3.6 – Приклад налаштування contextBridge

3.2 Розробка інтерфейсу користувача

Під час розробки інтерфейсу користувача для додатку було використано фронтенд-фреймворк Angular, що забезпечило швидку та ефективну розробку компонентів і модулів інтерфейсу.

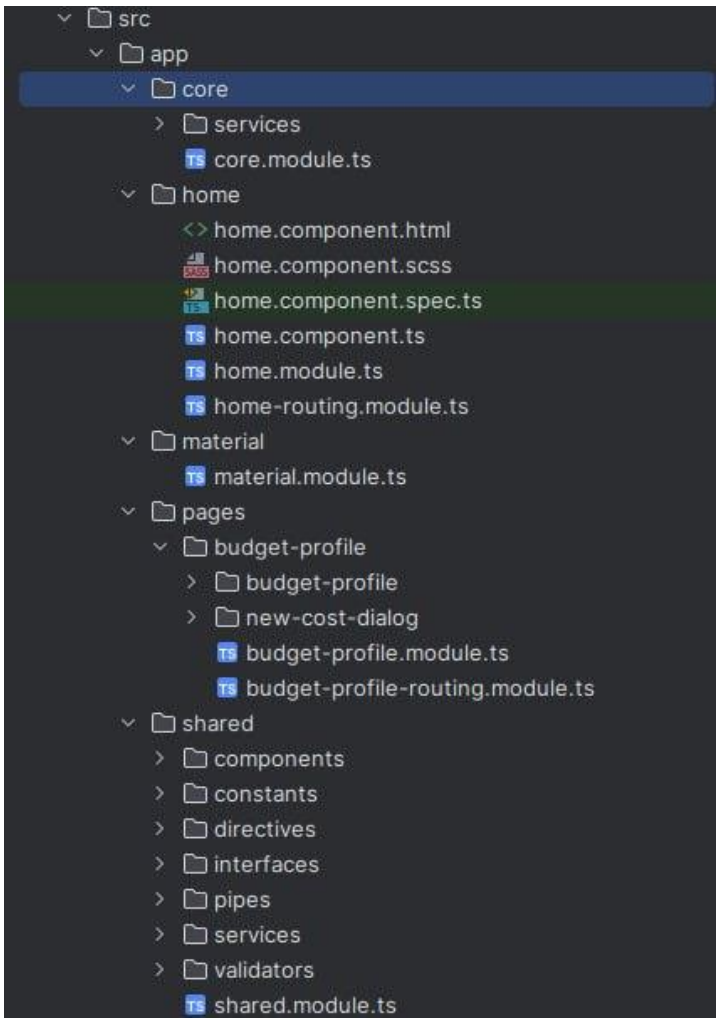


Рисунок 3.7 – Вигляд структури файлів Angular проєкту

Один з ключових елементів, що було використано, - це маршрутизація. Завдяки маршрутизації в Angular, було можливо створювати різні шляхи (URL) для кожного компонента і керувати навігацією між сторінками додатку. Це дозволило створити зручну навігаційну структуру та забезпечити користувачу зручний доступ до різних функціональних модулів.[20]

```
<ngx-charts-advanced-pie-chart
  (window:resize)="onResize($event)"
  [view]="view"
  [scheme]="'picnic'"
  [results]="chartData"
  [gradient]="gradient"
  (select)="onSelect($event)"
>
</ngx-charts-advanced-pie-chart>
```

Рисунок 3.8 – Приклад використання ngx-charts для візуалізації даних

Для візуалізації даних та створення графіків і діаграм в додатку була використана бібліотека ngx-charts. Це дозволило легко створювати різноманітні графіки, діаграми та інші візуальні елементи, що допомагають користувачеві отримати зрозумілу та зручну інформацію про свої фінансові ресурси.[15][18]

```
const material: any[] = [
  MatButtonModule,
  MatListModule, MatToolbarModule, MatIconModule, MatTooltipModule,
  MatFormFieldModule, MatInputModule, MatSelectModule, MatNativeDateModule,
  MatTableModule, MatDialogModule, MatDatepickerModule,
];

3 usages
@NgModule({
  declarations: [],
  imports: [...material],
  exports: [...material]
})
export class MaterialModule {
}
```

Рисунок 3.9 – Модуль з імпортованими компонентами, що використовуються у проєкті

```
<mat-form-field>
  <mat-label>{{ 'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.ENTER_COST_NAME' | translate}}</mat-label>
  <input matInput [formControl]="newCostForm.controls.name">
</mat-form-field>
<mat-form-field>
  <mat-label>{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.CHOOSE_CATEGORY' | translate}}</mat-label>
  <mat-select [formControl]="newCostForm.controls.category" required>
    <mat-option>--</mat-option>
    <mat-option *ngFor="let category of availableCategories" [value]="category">
      {{category}}
    </mat-option>
  </mat-select>
</mat-form-field>
<mat-form-field>
  <mat-label>{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.CHOOSE_DATE' | translate}}</mat-label>
  <input matInput [matDatepicker]="picker" [formControl]="newCostForm.controls.date">
  <mat-hint>{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.DATE_TEMPLATE' | translate}}</mat-hint>
  <mat-datepicker-toggle matIconSuffix [for]="picker"></mat-datepicker-toggle>
  <mat-datepicker #picker></mat-datepicker>
</mat-form-field>
<mat-form-field>
  <mat-label>{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.ENTER_VALUE' | translate}}</mat-label>
  <input matInput type="number" [formControl]="newCostForm.controls.value" placeholder="0" class="right-align">
  <span matTextSuffix>.00</span>
</mat-form-field>
```

Рисунок 3.10 – Приклад використання компонентів з Angular Material

Одним із ключових аспектів розробки інтерфейсу було використання компонентів та стилів з Angular Material. Angular Material - це набір готових компонентів, які пропонуються Angular, що спрощує створення стильного та сучасного інтерфейсу. За допомогою Angular Material було реалізовано такі елементи як кнопки, таблиці, форми, модальні вікна та багато інших, що допомогли створити привабливий та функціональний інтерфейс додатку.

Одним із зручних елементів, що було використано, - це модальні вікна. Модальні вікна дозволяють показувати важливу інформацію або взаємодіяти з користувачем у вигляді вікна, яке з'являється над основним вмістом додатку. Це дозволяє зосередитись на конкретній задачі та забезпечити зручну та контекстну інтеграцію з користувачем.

```
{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.BUTTONS.CANCEL' | translate}}
```

Рисунок 3.11 – Приклад використання pipe

```
<mat-form-field>
  <mat-label>{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.CHOOSE_DATE' | translate}}</mat-label>
  <input matInput [matDatepicker]="picker" [formControl]="newCostForm.controls.date">
  <mat-hint>{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.DATE_TEMPLATE' | translate}}</mat-hint>
  <mat-datepicker-toggle matIconSuffix [for]="picker"></mat-datepicker-toggle>
  <mat-datepicker #picker></mat-datepicker>
</mat-form-field>
```

Рисунок 3.12 – Приклад використання pipe translate для форматування даних (можливості перекладу)

Для форматування та трансформування даних в додатку були використані пайпи. Пайпи в Angular дозволяють легко змінювати формат даних, виконувати фільтрацію, сортування та інші операції. Наприклад, за допомогою пайпів можна форматувати дати, числа, грошові значення тощо, що покращує зрозумілість та зручність відображення даних для користувача.[7]

Загалом, використання Angular, ngx-charts, маршрутизації, Angular Material, модальних вікон, пайпів та інших функціональних можливостей дозволило створити зручний та привабливий інтерфейс для додатку контролю фінансів. Вони забезпечили швидку розробку, зручну навігацію, візуалізацію даних та покращили взаємодію з користувачем.

Додатковою функціональністю, що була використана в додатку, є можливість валідації введених даних. Angular надає зручні інструменти для валідації форм, які дозволяють перевіряти правильність введених значень і надавати користувачеві відповідні повідомлення про помилки.

Крім того, була використана можливість створення власних компонентів та сервісів. Це дозволяє організувати код додатку в логічні блоки, що спрощує його розуміння та підтримку.

Також були використані маршрутизація та Angular Material для створення навігації та розміщення елементів інтерфейсу. Маршрутизація дозволяє переходити між різними сторінками додатку без перезавантаження сторінки, що покращує його продуктивність та користувацький досвід. Angular Material надає набір готових компонентів та стилів, що спрощує розробку інтерфейсу та забезпечує його єдність.

Модальні вікна були використані для створення popup-вікон, які використовуються для показу додаткової інформації, підтвердження видалення або виконання інших дій, не переключаючи контекст додатку.

Узагальнюючи, використання всіх цих функціональних можливостей Angular дозволило створити додаток з багатим функціоналом, зручним інтерфейсом та хорошою продуктивністю.

3.3 Розробка функціональності додатку

На фінальній стадії розробки додатку, він має таку функціональність:

— Створення відокремлених профілів для контролю не суміжних фінансів.

— Можливість візуалізації даних щодо поточного стану фінансів для конкретного профілю.

— Можливість додавання надходжень або витрат, а також їх видалення.

— Можливість відображення усіх даних для профіля у вигляді таблиці.

— Збереження усіх даних у json файл з використанням electron-store.

Вигляд основних компонентів додатка продемонстровано на рисунках нижче.

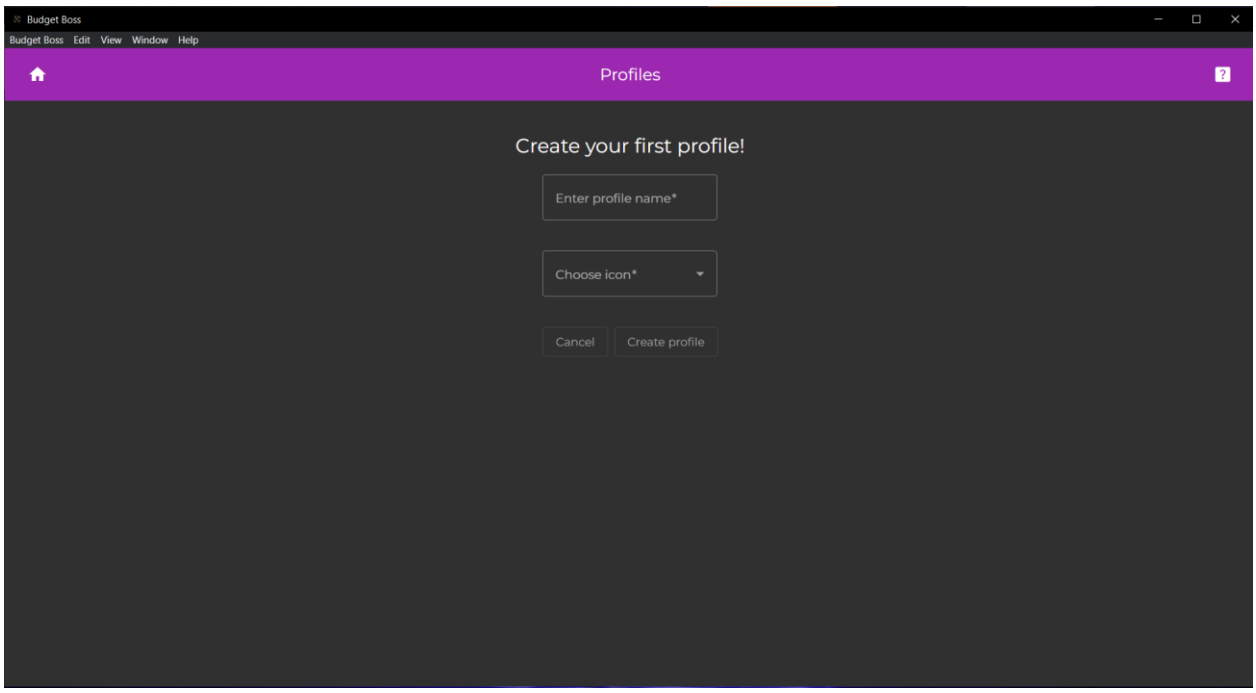


Рисунок 3.13 – Початковий екран додатку при першому запуску

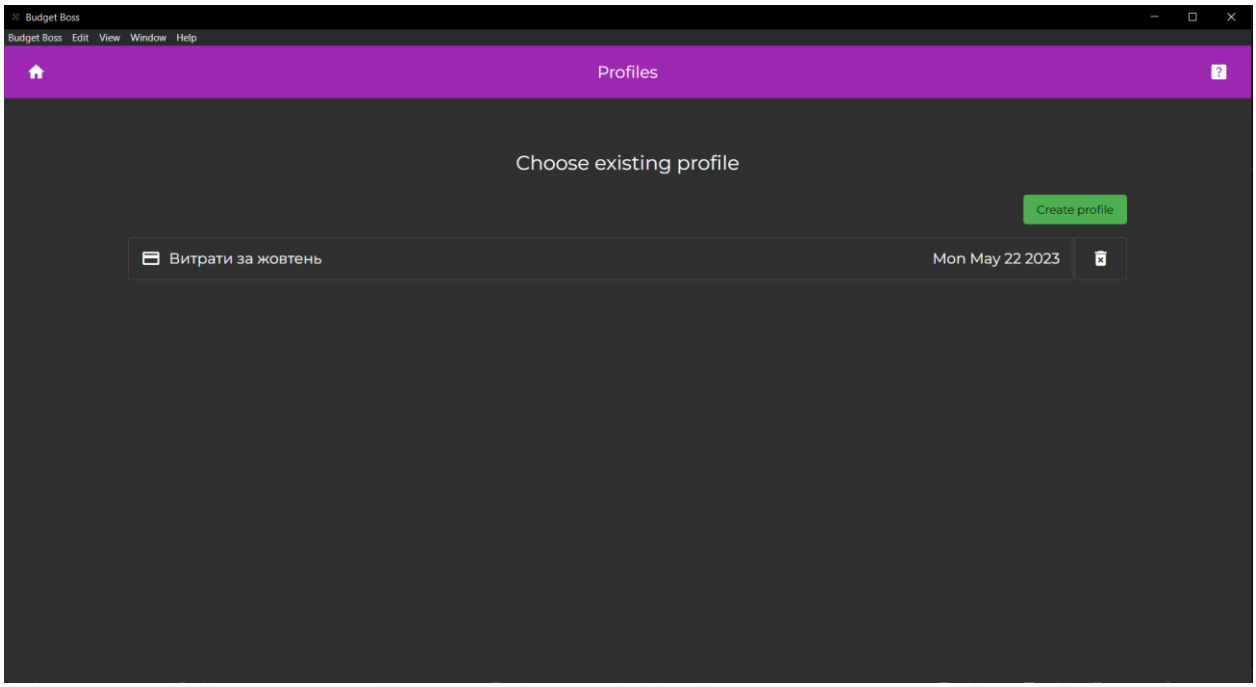


Рисунок 3.14 – Головне вікно при наявному профілі

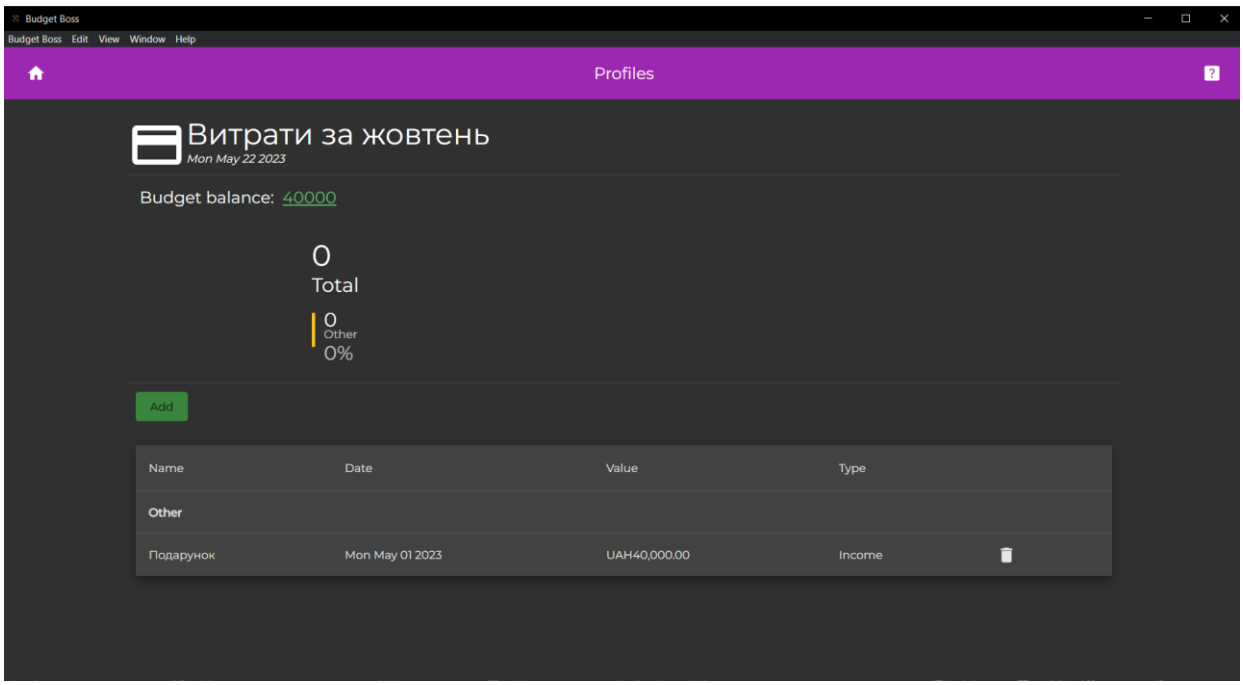


Рисунок 3.15 – Вікно профілю без доданих витрат

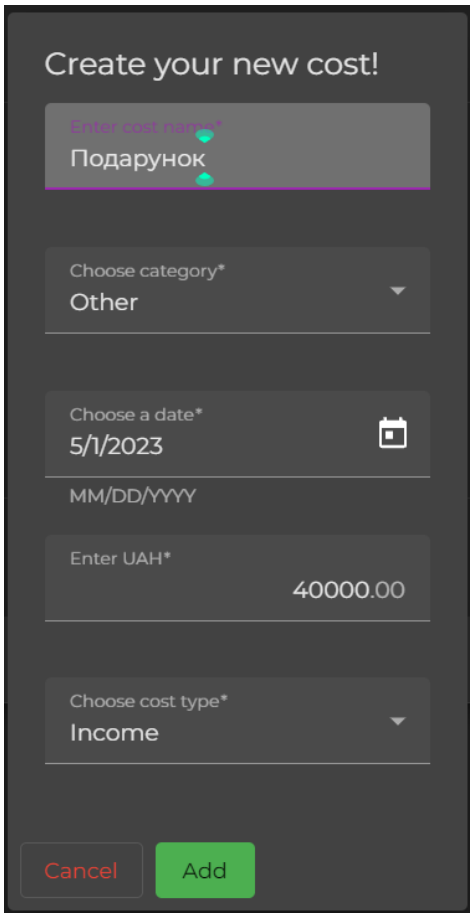


Рисунок 3.16 – Модальне вікно додавання нової транзакції(доходу)

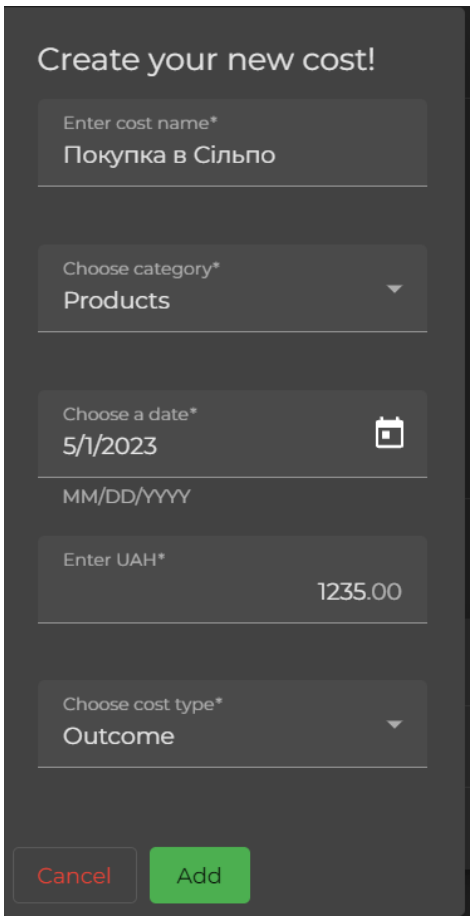


Рисунок 3.17 – Модальне вікно додавання транзації(витрата)

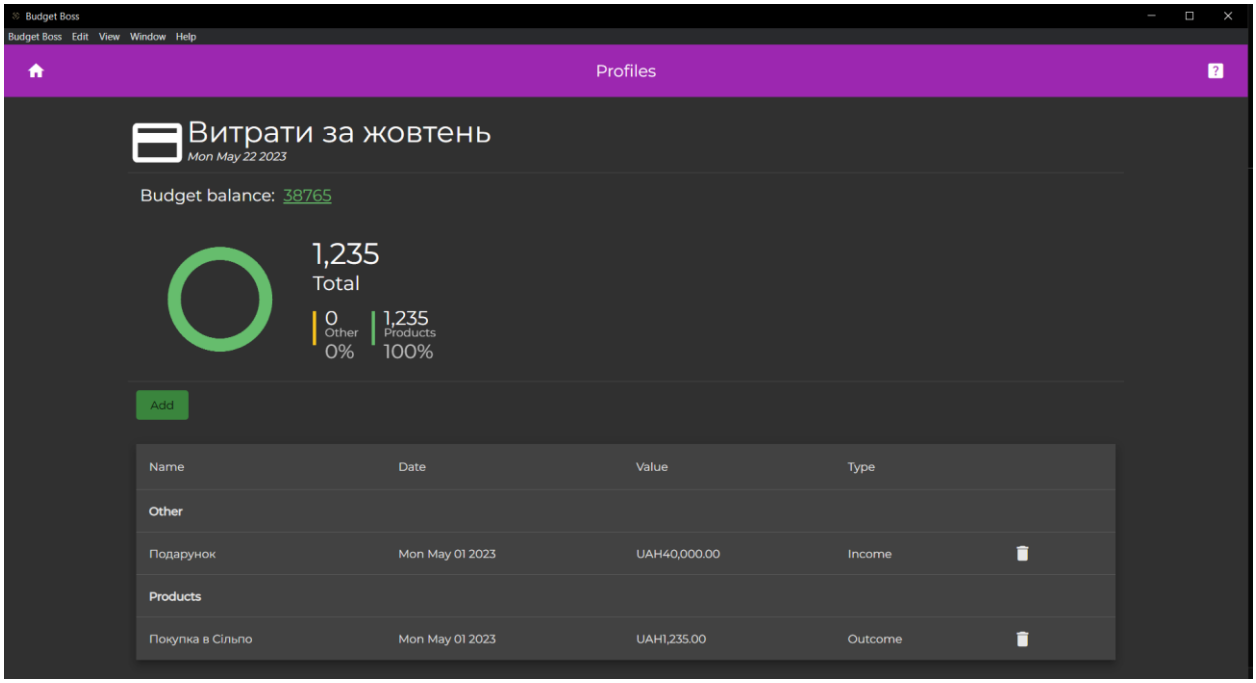


Рисунок 3.18 – Вікно відображення стану бюджету за наявних витрат та доходів

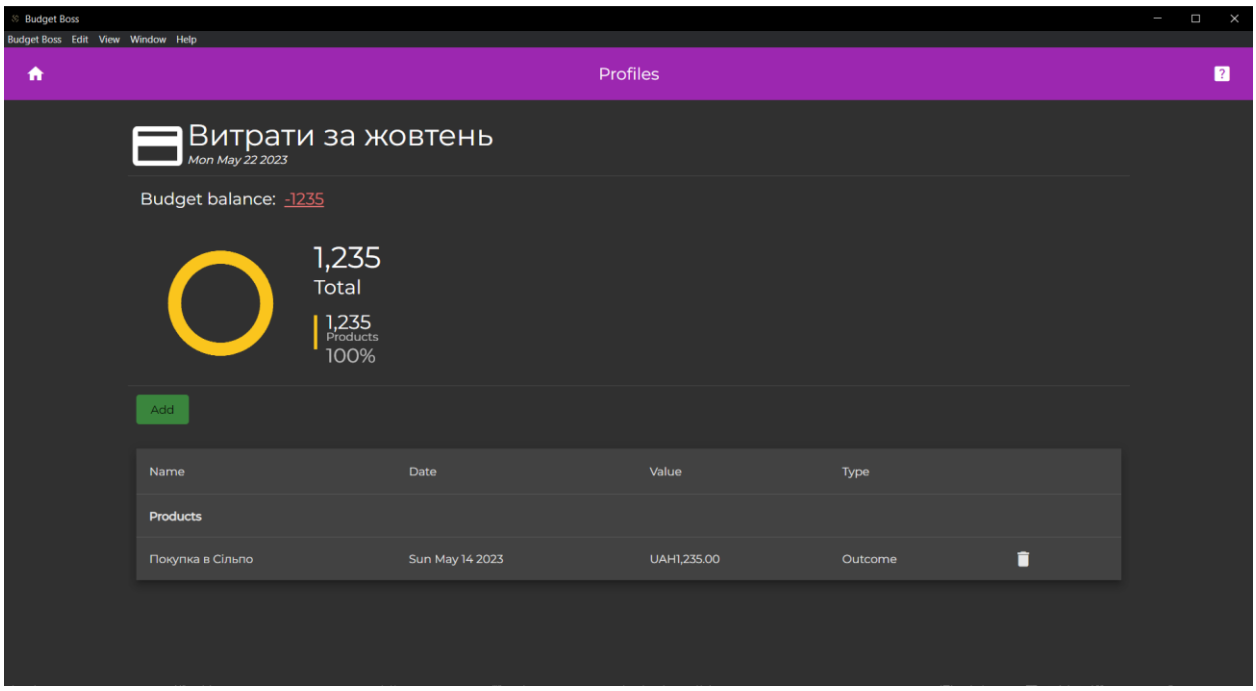


Рисунок 3.19 – Вікно відображення стану бюджету лише за наявності витрат

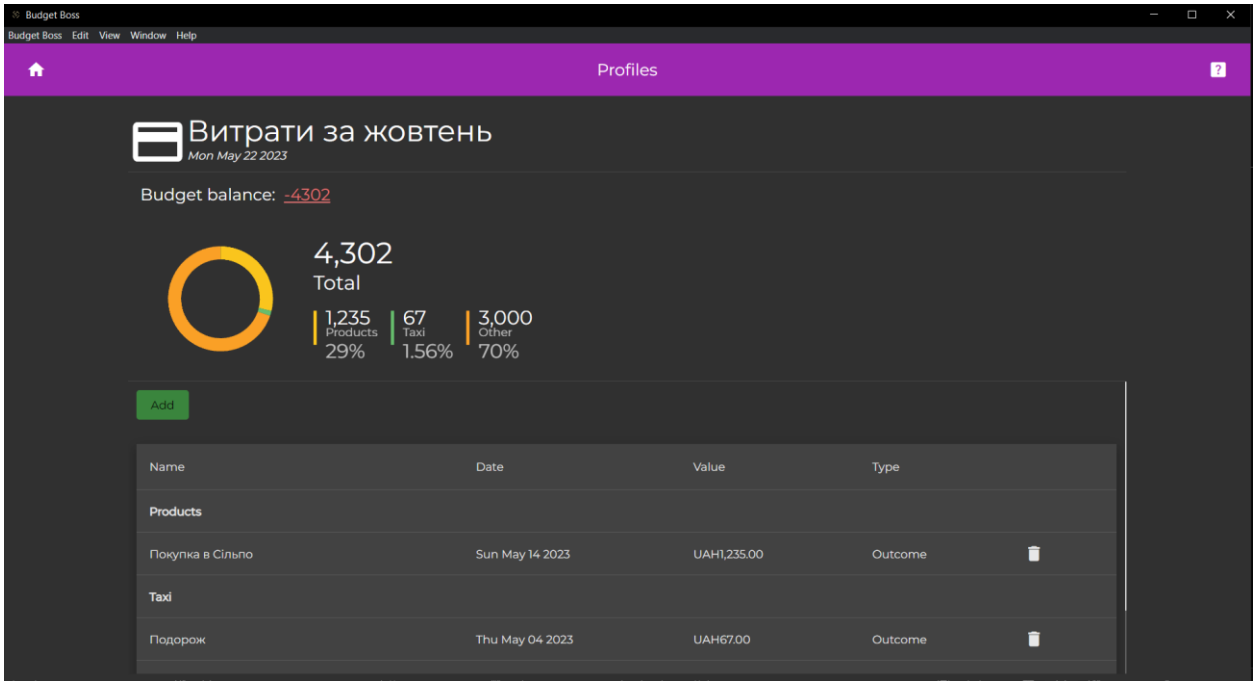


Рисунок 3.20 – Вікно відображення стану витрат у декількох категоріях з відповідною діаграмою до них

3.4 Розгортання та випуск додатку

Розгортання та випуск додатку є важливою частиною процесу розробки, який дозволяє розмістити додаток у відповідному сховищі та зробити його доступним для користувачів. Основні кроки розгортання та випуску додатку можуть включати наступні етапи:

- Конфігурація середовища: Перед розгортанням додатку необхідно налаштувати середовище, де буде зібраний додаток. Оскільки особливості electron-builder дозволяють зібрати додаток під ОС Windows на платформах Windows, Linux або MacOS, у свою чергу збірка під ОС Linux можлива лише на платформах Linux та MacOS, а MacOS – лише на MacOS.
- Збирання додатку: Використовуючи інструменти, такі як Angular CLI та electron-builder, необхідно зібрати додаток в один файл, готовий для розгортання. Це включає оптимізацію та зведення всіх компонентів, модулів та залежностей додатку.
- Конфігурація сервера: Якщо використовується серверна сторона, необхідно налаштувати сервер для розгортання додатку. Це може включати налаштування маршрутизації, безпеки, доступу до бази даних та інших параметрів.
- Тестування: Перед випуском додатку, він може бути протестований, щоб переконатися, що все працює належним чином. Це може включати функціональне тестування, тестування продуктивності, перевірку сумісності та інші види тестування.
- Випуск та розповсюдження: Після успішного розгортання та тестування додатку можна випустити його виробниче середовище. Це може включати публікацію додатку в магазині додатків, створення інсталяційних пакетів для завантаження або розміщення додатку на власних серверах.

Використання публічного інстурмента - electron-builder, спрощує процес розгортання крос-платформних додатків на основі Electron. Electron Builder надає зручність та автоматизацію в процесі збирання та розгортання додатку. Він дозволяє налаштувати різні параметри, такі як іконки додатку, інформацію про версію, назву файлу виконання та багато іншого. Крім того, він підтримує

створення інсталяційних пакетів для розповсюдження додатку на різних платформах.[11]

Узагальнюючи, electron-builder є потужним інструментом для розгортання та випуску крос-платформних додатків на основі Electron, який спрощує процес розгортання та забезпечує зручність у налаштуванні та розповсюдженні додатку.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи з теми "Крос-платформний додаток контролю фінансів" було досягнуто мету отримання практичних навичок розробки додатків на базі фреймворка Electron та фронтенд-фреймворка Angular, а також вивчення технологій зберігання та обробки фінансової інформації.

Під час частини практичної реалізації було освоєно проектування, розробка, тестування крос-платформного додатку з використанням сучасних технологій та інструментів. Було розглянуто практичні аспекти роботи з фінансовою інформацією та її зберігання.

Проєкт "Крос-платформний додаток контролю фінансів" було розроблено з використанням фреймворку Electron та фронтенд-фреймворка Angular. Додаток дозволяє користувачам контролювати свої фінансові ресурси, відслідковувати витрати та прибутки, обирати категорії та отримувати статистику за допомогою візуальних елементів.

Крім того, в процесі розробки було приділено особливу увагу використанню Angular Material для розробки інтерфейсу користувача. Angular Material - це набір готових компонентів та стилів, які дозволяють створювати естетичний та сучасний вигляд додатку. Використання Angular Material спрощує процес розробки, оскільки він надає широкий спектр готових компонентів, таких як кнопки, форми, таблиці, модальні вікна та інші, що можуть бути використані безпосередньо у проєкті.

Також, важливим аспектом розробки було забезпечення безпеки використання за допомогою використання contextBridge та preload файлу в Electron. ContextBridge дозволяє контролювати доступ до API між процесами в Electron та запобігає можливості зловживання або несанкціонованого доступу до системних ресурсів. Preload файл використовується для встановлення контексту, у якому можна безпечно виконувати код на стороні рендерера, але з обмеженими правами доступу до системних ресурсів.

В результаті, завдяки використанню Electron, Angular та Angular Material, було розроблено крос-платформний додаток контролю фінансів зі зручним інтерфейсом та надійними механізмами безпеки. Використання цих технологій надає користувачам можливість ефективно керувати своїми фінансами та забезпечує їхню конфіденційність та безпеку використання.

Крос-платформність додатку була забезпечена за допомогою Electron, що дозволило запускати додаток на різних платформах, таких як Windows, MacOS, Linux тощо. Фронтенд-фреймворк Angular було використано для розробки користувацького інтерфейсу та логіки додатку, а також для забезпечення швидкої та ефективної роботи з даними.

Додаток має локальне сховище даних, реалізоване за допомогою npm-пакету electron-store, що дозволяє зберігати дані у json-файлах та здійснювати міграції з використанням коду на TypeScript (JavaScript) для зміни значень у нових версіях.

Завершуючи, розробка архітектури додатку на базі Angular Material та Electron була захопливою та вдалим досвідом бакалаврської роботи. Вона надала можливість поглибити знання у розробці десктопних додатків. Під час роботи було успішно досягнуто поставлені цілі та отримано цінний досвід у розробці крос-платформних десктопних додатків, що буде корисним у подальшій професійній діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Економічна правда. Мобільні застосунки для планування бюджету та контролю витрат: які найбільш популярні [Електронний ресурс] / Економічна правда // Економічна правда. – Режим доступу: https://www.epravda.com.ua/publications/2023/02/22/697326/.

2. Переваги та недоліки кросплатформної та нативної розробки мобільних додатків - merehead [Електронний ресурс] // Merehead. – Режим доступу: https://merehead.com/ua/blog/cross-platform-native-mobile-development/.

3. Фінансовий менеджмент : підручник / М. І. Крупка [та ін.]. – Львів : ЛНУ ім. Ів. Франка, 2019. – 440 с.

4. 10 best cross-platform app frameworks to consider for your app development [Електронний ресурс] // Home | NodeDev. – Режим доступу: https://node.dev/post/10-best-cross-platform-app-frameworks-to-consider-for-your-app-development.

5. 4 Best frameworks for cross-platform desktop app development [Електронний ресурс] // Qt QML Software Development Consulting Specialists - Scythe Studio. – Режим доступу: https://scythe-studio.com/en/blog/4-best-frameworks-for-cross-platform-desktop-app-development.

6. Advanced Electron.js architecture - LogRocket Blog [Електронний ресурс] // LogRocket Blog. – Режим доступу: https://blog.logrocket.com/advanced-electron-js-architecture/.

7. Angular [Електронний ресурс] // Angular. – Режим доступу: https://angular.io/docs.

8. Angular. Angular material [Електронний ресурс] / Angular // Angular Material. – Режим доступу: https://material.angular.io/.

9. Angular architecture patterns and best practices (that help to scale) [Електронний ресурс] // Dev-Academy.com - Web security | Testing & automation

| Application architecture. – Режим доступу: https://dev-academy.com/angular-architecture-best-practices/.

10. Dore K. Best budgeting apps of 2023 [Електронний ресурс] / Kate Dore // Investopedia. – Режим доступу: https://www.investopedia.com/best-budgeting-apps-5085405.

11. Electron-builder [Електронний ресурс] // electron-builder. – Режим доступу: https://www.electron.build/.

12. Electron IPC Response/Request architecture with TypeScript - LogRocket Blog [Електронний ресурс] // LogRocket Blog. – Режим доступу: https://blog.logrocket.com/electron-ipc-response-request-architecture-with-typescript/.

13. GitHub - sindresorhus/electron-store: Simple data persistence for your Electron app or module - Save and load user preferences, app state, cache, etc [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/sindresorhus/electron-store.

14. Introduction | electron [Електронний ресурс] // Build cross-platform desktop apps with JavaScript, HTML, and CSS | Electron. – Режим доступу: https://www.electronjs.org/docs/latest/.

15. Introduction - ngx-charts [Електронний ресурс] // Introduction - ngx-charts. – Режим доступу: https://swimlane.gitbook.io/ngx-charts.

16. Kumar H. Best practices and guidelines for web applications [Електронний ресурс] / Harish Kumar // Halodoc Blog. – Режим доступу: https://blogs.halodoc.io/angular-best-practices/.

17. Martindale J. A. 52 simple ways to manage your money: a weekly journal & workbook to help you take real control of your money! / Judith A. Martindale. – Napersville, Ill : Sourcebooks, 1994. – 212 с.

18. Prasad S. Building data visualizations with angular and ngx-charts - dzone [Електронний ресурс] / Swathi Prasad // dzone.com. – Режим доступу: https://dzone.com/articles/building-data-visualizations-with-angular-and-ngx.

19. PWA vs electron - which architecture wins? | clean commit [Електронний ресурс] // Application Development & Headless eCommerce | Clean Commit. – Режим доступу: https://cleancommit.io/blog/pwa-vs-electron-which-architecture-wins/.

20. Top 11 angular best practices to adapt in 2023 (updated) [let's optimize code] [Електронний ресурс] // Aglowid IT Solutions. – Режим доступу: https://aglowiditsolutions.com/blog/angular-best-practices/.

ДОДАТОК А

Лістинг програмного коду фронтенд частини

home-routing.module.ts

```
import { Component, NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home.component';

export const HOME_ROUTES: Routes = [
  { path: '', component: HomeComponent },
];

@NgModule({
  declarations: [],
  imports: [CommonModule, RouterModule.forChild(HOME_ROUTES)],
  exports: [RouterModule]
})
export class HomeRoutingModule {}
```

home.component.html

```
<div class="wrapper">
  <ng-container *ngIf="profiles && profiles.length > 0 &&
!newProfileCreateSelected; else createProfileTemplate">
    <div class="profile-list-wrapper">
      <p>Choose existing profile</p>
      <button mat-flat-button color="accent" (click)="onNewProfileClick()"
class="create-profile-
```

```
button">{{'PAGES.HOME.FORMS.NEW_PROFILE_FORM.BUTTONS.CREAT
E_PROFILE' | translate}}</button>
    <mat-action-list class="profile-list">
     <ng-container *ngFor="let profile of profiles">
      <div class="profile-list-item">
       <button mat-stroked-button class="profile-list-item-info"
(click)="onProjectSelectClick(profile)">
        <div class="profile-list-button-content">
         <span><mat-icon inline>{{profile.icon}}</mat-icon></span>
         <span>{{profile.name}}</span>
         <span>{{convertDateToString(profile.createdAt)}}</span>
        </div>
       </button>
       <mat-divider vertical/>
       <button mat-stroked-button class="profile-list-item-delete"
            [matTooltip]="'PAGES.HOME.TOOLTIPS.DELETE_PROFILE' |
translate"
            (click)="onDeleteProfileClick(profile)">
        <span><mat-icon inline>delete_forever</mat-icon></span>
       </button>
      </div>
     </ng-container>
    </mat-action-list>
   </div>
  </ng-container>
</div>

<ng-template #createProfileTemplate>
  <div class="empty-profiles-wrapper">
   <span>{{ 'PAGES.HOME.EMPTY_PROFILES' | translate}}</span>
```

```
  <form [formGroup]="newProfileForm" class="form-new-profile">
    <mat-form-field appearance="outline">
      <mat-label>{{
'PAGES.HOME.FORMS.NEW_PROFILE_FORM.LABELS.NAME'                    |
translate}}</mat-label>
      <input matInput [formControl]="newProfileForm.controls.name">
      <mat-error *ngIf="newProfileForm.controls.name.status !== 'VALID'">
        {{
        newProfileForm.controls.name.hasError('required')                  ?
('PAGES.HOME.FORMS.NEW_PROFILE_FORM.ERRORS.REQUIRED'          |
translate) :
          newProfileForm.controls.name.hasError('minlength')               ?
('PAGES.HOME.FORMS.NEW_PROFILE_FORM.ERRORS.MIN_LENGTH'        |
translate) :
            newProfileForm.controls.name.hasError('maxlength')             ?
('PAGES.HOME.FORMS.NEW_PROFILE_FORM.ERRORS.MAX_LENGTH'        |
translate) :
              null
        }}
      </mat-error>
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>{{
'PAGES.HOME.FORMS.NEW_PROFILE_FORM.LABELS.ICON'                    |
translate}}</mat-label>
      <mat-select [formControl]="newProfileForm.controls.icon" required>
        <mat-select-trigger>
          <mat-icon>{{newProfileForm.controls.icon.value}}</mat-icon>
        </mat-select-trigger>
        <mat-option>--</mat-option>
```

```
        <mat-option *ngFor="let icon of availableIcons" [value]="icon">
          <mat-icon inline>{{icon}}</mat-icon>
        </mat-option>
      </mat-select>
      <mat-error *ngIf="newProfileForm.controls.icon.status !== 'VALID'">
        {{
        newProfileForm.controls.icon.hasError('required')                    ?
('PAGES.HOME.FORMS.NEW_PROFILE_FORM.ERRORS.REQUIRED'          |
translate) :
          null
        }}
      </mat-error>
    </mat-form-field>
    <div class="form-actions">
      <button type="button" mat-stroked-button color="warn" [disabled]="!profiles
|| profiles.length < 1" (click)="onCancelClick()">
        {{'PAGES.HOME.FORMS.NEW_PROFILE_FORM.BUTTONS.CANCEL'
| translate}}
      </button>
      <button          type="submit"          mat-stroked-button          color="accent"
(click)="onSubmitClick()" [disabled]="newProfileForm.status !== 'VALID'">
        {{'PAGES.HOME.FORMS.NEW_PROFILE_FORM.BUTTONS.SUBMIT' |
translate}}
      </button>
    </div>
  </form>
 </div>
</ng-template>
```

home.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { IpcConnectorService } from '@shared/services';
import { Profile } from '@shared/interfaces/profile';
import { AbstractControl, FormControl, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})
export class HomeComponent implements OnInit {
  profiles: Profile[] = [];
  newProfileForm: FormGroup<{
    name: FormControl<string>;
    icon: FormControl<string>;
  }>;
  availableIcons: string[] = ['monetization_on', 'credit_card', 'account_balance_wallet', 'euro', 'currency_bitcoin', 'payments', 'savings'];
  newProfileCreateSelected = false;

  constructor(
    private ipcConnectorService: IpcConnectorService,
    private router: Router) { }

  ngOnInit(): void {
    this.ipcConnectorService.getProfilesAsync().then(fetchedProfiles => {
      this.profiles = fetchedProfiles;
      console.log(fetchedProfiles);
```

```
    });
    console.log('Home init');

    this.newProfileForm = new FormGroup({
      name: new FormControl<string>('', [Validators.required,
Validators.minLength(3), Validators.maxLength(25)]),
      icon: new FormControl<string>('', [this.inputValidator(this.availableIcons)])
    });
  }

  convertDateToString(value: number): string {
    return new Date(value).toDateString();
  }

  onProjectSelectClick(profile: Profile) {
    this.router.navigate(['/budget-profile', profile.id]);
  }

  onDeleteProfileClick(profile: Profile) {
    console.log('Profile to delete', profile);
    this.ipcConnectorService.deleteProfileAsync(profile).then(() => {
      this.profiles = this.profiles.filter(value => value.id !== profile.id);
      console.log(this.profiles);
    });
  }

  inputValidator(val) {
    return (control: AbstractControl): { [key: string]: boolean } | null => {
      if (
        control.value !== null && !val.includes(control.value)
```

```
    ) {
      return { inputValidator: true };
    }
    return null;
  };
}

onNewProfileClick() {
  this.newProfileCreateSelected = !this.newProfileCreateSelected;
}

onCancelClick() {
  this.newProfileCreateSelected = false;
}

onSubmitClick() {
  const profileName = this.newProfileForm.controls.name.value;
  const profileIcon = this.newProfileForm.controls.icon.value;

  const profile: Partial<Profile> = {
    name: profileName,
    icon: profileIcon
  };

  this.ipcConnectorService.addProfileAsync(profile).then(value => {
    console.log('received value', value);
    if (value !== null) {
      this.profiles.push(value);
    }
    this.newProfileCreateSelected = false;
```

```
    this.resetForm();
   });
  }

  resetForm() {
   this.newProfileForm.reset();
  }
}
```

material.module.ts

```
import { NgModule } from '@angular/core';
import { MatButtonModule } from '@angular/material/button';

import { MatListModule } from '@angular/material/list';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatIconModule } from '@angular/material/icon';
import { MatTooltipModule } from '@angular/material/tooltip';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { MatSelectModule } from '@angular/material/select';
import { MatNativeDateModule } from '@angular/material/core';
import { MatTableModule } from '@angular/material/table';
import { MatDialogModule } from '@angular/material/dialog';
import { MatDatepickerModule } from '@angular/material/datepicker';

const material: any[] = [
  MatButtonModule,
  MatListModule, MatToolbarModule, MatIconModule, MatTooltipModule,
  MatFormFieldModule, MatInputModule, MatSelectModule,
MatNativeDateModule,
```

```
  MatTableModule, MatDialogModule, MatDatepickerModule,
];

@NgModule({
  declarations: [],
  imports: [...material],
  exports: [...material]
})
export class MaterialModule {
}
```

budget-profile.component.html

```
<div class="wrapper">
  <div class="container">
    <div class="profile-info">
      <span class="profile-icon">
        <mat-icon>{{profile?.icon ?? 'account_box'}}</mat-icon>
      </span>
      <p class="profile-name">{{profile?.name ?? 'Unknown'}}</p>
      <p class="created-date">{{convertDateToString(profile?.createdAt ??
Date.now())}}</p>
    </div>
    <mat-divider/>
    <div class="chart-container">
      <div class="overall-info">
        <p>Budget balance:  </p>
        <span [class]="freeBudget >= 0 ? 'positive-balance' : 'negative-
balance'">{{this.freeBudget}}</span>
      </div>
      <ngx-charts-advanced-pie-chart
```

```
      (window:resize)="onResize($event)"
      [view]="view"
      [scheme]="'picnic'"
      [results]="chartData"
      [gradient]="gradient"
      (select)="onSelect($event)"
    >
    </ngx-charts-advanced-pie-chart>
  </div>
  <mat-divider/>
  <div class="main-info">
    <div class="actions">
      <button mat-flat-button color="accent"
(click)="onAddCostClick()">Add</button>
<!--      <button mat-stroked-button color="warn" >Delete</button>-->
    </div>
    <div class="table-wrapper">
      <table mat-table [dataSource]="tableData" class="mat-elevation-z8">
        <!-- Name Column -->
        <ng-container matColumnDef="name">
          <th mat-header-cell *matHeaderCellDef> Name </th>
          <td mat-cell *matCellDef="let element"> {{element.name}} </td>
        </ng-container>

        <!-- Date Column -->
        <ng-container matColumnDef="date">
          <th mat-header-cell *matHeaderCellDef> Date </th>
          <td mat-cell *matCellDef="let element">
{{convertDateToString(element.date)}} </td>
        </ng-container>
```

```
<!-- Value Column -->
<ng-container matColumnDef="value">
  <th mat-header-cell *matHeaderCellDef> Value </th>
  <td mat-cell *matCellDef="let element"> {{element.value | currency:'UAH'}} </td>
</ng-container>

<!-- Type Column -->
<ng-container matColumnDef="type">
  <th mat-header-cell *matHeaderCellDef> Type </th>
  <td mat-cell *matCellDef="let element"> {{element.type}} </td>
</ng-container>

<!-- Delete Column -->
<ng-container matColumnDef="delete">
  <th mat-header-cell *matHeaderCellDef></th>
  <td mat-cell *matCellDef="let element"><button mat-icon-button (click)="onDeleteCost(element)"><mat-icon>delete</mat-icon></button></td>
</ng-container>

<tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
<tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>

<!-- Group header -->
<ng-container matColumnDef="groupHeader">
  <td colspan="999" mat-cell *matCellDef="let groupBy"><strong>{{groupBy.title}}</strong></td>
</ng-container>
```

```
        <tr mat-row *matRowDef="let row; columns: ['groupHeader']; when: isGroup"> </tr>

      </table>
    </div>
  </div>
 </div>
</div>
```

budget-profile.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { VariableService } from '@shared/services/variable.service';
import { Profile } from '@shared/interfaces/profile';
import { IpcConnectorService } from '@shared/services';
import { BehaviorSubject } from 'rxjs';
import { Cost, GroupBy, ProfileBudget } from '@shared/interfaces/profile-budget';
import { NewCostDialogComponent } from '../new-cost-dialog/new-cost-dialog.component';
import { MatDialog } from '@angular/material/dialog';

@Component({
  selector: 'app-budget-profile',
  templateUrl: './budget-profile.component.html',
  styleUrls: ['./budget-profile.component.scss']
})
export class BudgetProfileComponent implements OnInit {
  profileId: string;
  profile: Profile = null;
  profileBudget = new BehaviorSubject<ProfileBudget>(null);
```

```
  chartData: { name: string; value: number }[] = [];
  view: [number, number] = [800, 200];
  gradient = false;

  tableData: (Cost | GroupBy)[] = [];
  displayedColumns: string[] = ['name', 'date', 'value', 'type', 'delete'];
  freeBudget = 0;
  constructor(private route: ActivatedRoute,
              private variableService: VariableService,
              private ipcConnectorService: IpcConnectorService,
              public dialog: MatDialog) {
  }

  ngOnInit(): void {
    this.profileId = this.route.snapshot.paramMap.get('id');
    this.ipcConnectorService.getProfileAsync(this.profileId).then(value => {
      this.profile = value;


this.ipcConnectorService.getProfileBudgetDataAsync(this.profileId).then((budget:
ProfileBudget) => {
        this.profileBudget.next(budget);
      });
    });

    this.profileBudget.subscribe(budget => {
      if (budget) {
        const incomeCosts = budget.costs.filter(value => value.type === 'Income');
        const outcomeCosts = budget.costs.filter(value => value.type === 'Outcome');
```

```
const res = this.groupBy<any , string>(budget.costs, i => i.category);

const keys = Object.keys(res);

const data = keys.map(value1 => ({
  name: value1,
  value: res[value1].reduce((accumulator: number, currValue) => {
    if (currValue.type === 'Outcome') {
      accumulator += currValue.value;
    }

    return accumulator;
  }, 0)
}));

this.freeBudget = incomeCosts.reduce((accumulator: number, currValue) =>
accumulator + currValue.value, 0) -
  outcomeCosts.reduce((accumulator: number, currValue) => accumulator +
currValue.value, 0);

this.chartData = data;

// Table data processing
const tableData: any[] = [];

for (const key of keys) {
  tableData.push({ title: key, isGroupBy: true } as GroupBy);
  tableData.push(...budget.costs.filter(cost => cost.category === key));
}
```

```
      this.tableData = tableData;
    }
  });
 }

 convertDateToString(value: number): string {
  return new Date(value).toDateString();
 }

 onSelect(data): void {
  console.log('Item clicked', JSON.parse(JSON.stringify(data)));
 }

 onResize(event) {
  this.view = [event.target.innerWidth >= 1000 ? 800 : event.target.innerWidth *
0.8, 200];
 }

 onDeleteCost(cost: Cost) {
  console.log(cost);
  const updatedProfileBudget = this.profileBudget.value;

  updatedProfileBudget.costs.splice(updatedProfileBudget.costs.indexOf(cost), 1);

  this.profileBudget.next(updatedProfileBudget);
 }

 // eslint-disable-next-line @typescript-eslint/naming-convention,@typescript-
eslint/member-ordering
```

```
  readonly Date = Date;

  groupBy = <T, K extends keyof any>(arr: T[], key: (i: T) => K) =>
    arr.reduce((groups, item) => {
      (groups[key(item)] ||= []).push(item);
      return groups;
    }, {} as Record<K, T[]>);

  isGroup(index, item): boolean{
    return item.isGroupBy;
  }

  onAddCostClick() {
    const dialogRef = this.dialog.open(NewCostDialogComponent, {
      disableClose: true,
    });

    dialogRef.afterClosed().subscribe(result => {
      if (result && result.newCost) {
        console.log(result);
        const updatedProfile = this.profileBudget.value;
        updatedProfile.costs.push(result.newCost);
        this.profileBudget.next(updatedProfile);
      }
    });
  }
}
```

new-cost-dialog.component.html

```
<h1 mat-dialog-title>{{
'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.CREATE_COST' |
translate}}</h1>
<div mat-dialog-content [formGroup]="newCostForm" class="dialog-content">
 <mat-form-field>
  <mat-label>{{
'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.ENTER_COST_
NAME' | translate}}</mat-label>
  <input matInput [formControl]="newCostForm.controls.name">
 </mat-form-field>
 <mat-form-field>
  <mat-
label>{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.CHOOS
E_CATEGORY' | translate}}</mat-label>
  <mat-select [formControl]="newCostForm.controls.category" required>
   <mat-option>--</mat-option>
   <mat-option *ngFor="let category of availableCategories" [value]="category">
    {{category}}
   </mat-option>
  </mat-select>
 </mat-form-field>
 <mat-form-field>
  <mat-
label>{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.CHOOS
E_DATE' | translate}}</mat-label>
  <input matInput [matDatepicker]="picker"
[formControl]="newCostForm.controls.date">
  <mat-
hint>{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.DATE_T
EMPLATE' | translate}}</mat-hint>
```

```
    <mat-datepicker-toggle matIconSuffix [for]="picker"></mat-datepicker-toggle>
    <mat-datepicker #picker></mat-datepicker>
  </mat-form-field>
  <mat-form-field>
    <mat-
label>{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.ENTER
_VALUE' | translate}}</mat-label>
    <input matInput type="number" [formControl]="newCostForm.controls.value"
placeholder="0" class="right-align">
    <span matTextSuffix>.00</span>
  </mat-form-field>
  <mat-form-field>
    <mat-
label>{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.LABELS.CHOOS
E_COST_TYPE' | translate}}</mat-label>
    <mat-select [formControl]="newCostForm.controls.type" required>
      <mat-option>--</mat-option>
      <mat-option *ngFor="let type of availableTypes" [value]="type">
        {{type}}
      </mat-option>
    </mat-select>
  </mat-form-field>

</div>
<div mat-dialog-actions>
  <button mat-stroked-button color="warn"
(click)="onCancel()">{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.B
UTTONS.CANCEL' | translate}}</button>
  <button mat-flat-button color="accent" type="submit" (click)="onSubmitClick()"
autofocus [disabled]="newCostForm.status !==
```

```
'VALID'">{{'PAGES.BUDGET_PROFILE.DIALOGS.NEW_COST.BUTTONS.S
UBMIT' | translate}}</button>
</div>
```

new-cost-dialog.component.ts

```
import { Component, OnInit } from '@angular/core';
import { AbstractControl, FormControl, FormGroup, Validators } from
'@angular/forms';
import { MatDialogRef } from '@angular/material/dialog';
import { Cost } from '@shared/interfaces/profile-budget';

@Component({
  selector: 'app-new-cost-dialog',
  templateUrl: './new-cost-dialog.component.html',
  styleUrls: ['./new-cost-dialog.component.scss']
})
export class NewCostDialogComponent implements OnInit {
  newCostForm: FormGroup<{
    name: FormControl<string>;
    category: FormControl<string>;
    value: FormControl<number>;
    date: FormControl<Date>;
    type: FormControl<string>;
  }>;
  availableCategories = ['Products', 'Taxi', 'Other'];
  availableTypes = ['Income', 'Outcome'];

  constructor(
    public dialogRef: MatDialogRef<NewCostDialogComponent>,
  ) {
```

```
  }

  ngOnInit(): void {
    this.newCostForm = new FormGroup({
      name: new FormControl<string>('', [Validators.required,
Validators.minLength(3), Validators.maxLength(25)]),
      category: new FormControl<string>('', [Validators.required,
this.inputValidator(this.availableCategories)]),
      date: new FormControl<Date>(new Date(), Validators.required),
      value: new FormControl<number>(0, [Validators.required, this.nonZero]),
      type: new FormControl<string>(this.availableTypes[0], [Validators.required,
this.inputValidator(this.availableTypes)])
    });
  }

  inputValidator(val) {
    return (control: AbstractControl): { [key: string]: boolean } | null => {
      if (
        control.value !== null && !val.includes(control.value)
      ) {
        return { inputValidator: true };
      }
      return null;
    };
  }

  nonZero(control): { [key: string]: any } {
    if (Number(control.value) < 0) {
      return { nonZero: true };
    } else {
```

```
    return null;
   }
  }

  onCancel() {
   this.dialogRef.close({});
  }

  onSubmitClick() {
   const newCost: Cost = {
    name: this.newCostForm.controls.name.value,
    date: this.newCostForm.controls.date.value.getTime(),
    category: this.newCostForm.controls.category.value,
    type: this.newCostForm.controls.type.value,
    value: this.newCostForm.controls.value.value,
   };

   this.dialogRef.close({ newCost });
  }
}
```

budget-profile-routing.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { Routes, RouterModule } from '@angular/router';
import { BudgetProfileComponent } from './budget-profile/budget-
profile.component';
import { PageNotFoundComponent } from '@shared/components';

export const BUDGET_PROFILE_ROUTES: Routes = [
```

```
  { path: '', component: PageNotFoundComponent },
  { path: ':id', component: BudgetProfileComponent }
];

@NgModule({
  declarations: [],
  imports: [CommonModule,
RouterModule.forChild(BUDGET_PROFILE_ROUTES)],
  exports: [RouterModule]
})
export class BudgetProfileRoutingModule {}
```

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { PageNotFoundComponent } from '@shared/components';
import { HOME_ROUTES } from './home/home-routing.module';
import { BUDGET_PROFILE_ROUTES } from './pages/budget-profile/budget-
profile-routing.module';

const routes: Routes = [
  {
    path: '',
    redirectTo: 'home',
    pathMatch: 'full'
  },
  {
    path: 'home',
    children: HOME_ROUTES
  },
```

```
  {
    path: 'budget-profile',
    children: BUDGET_PROFILE_ROUTES
  },
  {
    path: '**',
    component: PageNotFoundComponent
  }
];

@NgModule({
  imports: [
    RouterModule.forRoot(routes, { useHash: true }),
  ],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
```

app.component.html

```
<div class="app-wrapper">
  <mat-toolbar color="primary" class="main-toolbar">
    <button mat-icon-button (click)="onHomeClick()"><mat-icon>home</mat-
icon></button>
    <span>{{toolbarTitle | translate}}</span>
    <button mat-icon-button [matTooltip]="'APP.TOOLTIPS.GET_HELP' |
translate"><mat-icon>help_center</mat-icon></button>
  </mat-toolbar>
  <router-outlet></router-outlet>
</div>
```

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { TranslateService } from '@ngx-translate/core';
import { APP_CONFIG } from '@env';
import { Router } from '@angular/router';
import { VariableService } from '@shared/services/variable.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit{
  toolbarTitle: string;
  constructor(
    private translate: TranslateService,
    private route: Router,
    private variableService: VariableService,
  ) {
    this.translate.setDefaultLang('en');
    console.log('APP_CONFIG', APP_CONFIG);

    this.toolbarTitle = this.variableService.toolbarTitle.getValue();
  }

  onHomeClick() {
    this.route.navigate(['home']).then();
  }
```

```
  ngOnInit(): void {
    console.log('App on Init');

    this.variableService.toolbarTitle.subscribe(value => this.toolbarTitle = value);
  }
}
```

profile-budget.ts

```
export interface ProfileBudget {
  profileId: string;
  limit: number;
  costs: Cost[];
}

export interface Cost {
  name: string;
  date: number;
  category: string;
  value: number;
  type: string;
}

export interface GroupBy {
  title: string;
  isGroupBy: boolean;
}
```

profile.ts

```
export interface Profile {
  id: string;
```

```
  name: string;
  createdAt: number;
  icon: string;
}
```

ipc-connector.service.ts

```
import { Injectable } from '@angular/core';
import { Profile } from '@shared/interfaces/profile';
import { ProfileBudget } from '@shared/interfaces/profile-budget';

@Injectable({
  providedIn: 'root'
})
export class IpcConnectorService {
  constructor() { }

  public async getProfileAsync(id: string): Promise<Profile> {
    return await window.ipcRendererApi.ProfilesApi.getProfileAsync(id);
  }
  public async getProfilesAsync(): Promise<Profile[]> {
    return await window.ipcRendererApi.ProfilesApi.getProfilesAsync();
  }

  public async addProfileAsync(profile: Partial<Profile>): Promise<Profile | null>
{
    return await window.ipcRendererApi.ProfilesApi.addProfileAsync(profile);
  }

  public async deleteProfileAsync(profile: Profile): Promise<void> {
    return await window.ipcRendererApi.ProfilesApi.deleteProfileAsync(profile);
```

```
  }

  // eslint-disable-next-line max-len
  public async getProfileBudgetDataAsync(id: string): Promise<ProfileBudget> {
    return await window.ipcRendererApi.ProfilesApi.getProfileById(id);
  }
}
```

ДОДАТОК Б

Лістинг програмного коду бекенд частини

preload.ts

```
// eslint-disable-next-line import/no-extraneous-dependencies
import { contextBridge } from 'electron';
import { ipcRenderer } from 'electron/renderer';
import { functions, LogFunctions } from 'electron-log';
import * as path from 'path';
import { ChannelType } from './interfaces/ipc/channel-type.enum';
import { StoreGetValueChannelResponse } from './interfaces/ipc/channels/store-get-value-channel.interface';
import { PingPongChannelResponse } from './ipc/channels/ping-pong.channel';
import { AppVersionChannelResponse } from './interfaces/ipc/channels/app-version-channel.interface';
import { IpcService } from './ipc/ipc-service';
import { Profile } from './interfaces/profile';

export type IpcRendererApi = {
  sendPingMessage: (text: string) => Promise<PingPongChannelResponse>;
  StoreApi: {
    getValue: (key: string) => Promise<StoreGetValueChannelResponse>;
    setValue: (key: string, value: any) => Promise<any>;
  };
  UpdatesApi: {
    getAppVersion: () => Promise<AppVersionChannelResponse>;
  };
  ActionsApi: {
    sendAppAction: (action: string) => Promise<any>;
    removeListeners: (channels: ChannelType[]) => Promise<void>;
```

```
  };
  ProfilesApi: {
    getProfileAsync: (id: string) => Promise<Profile>;
    getProfilesAsync: () => Promise<Profile[]>;
    addProfileAsync: (profile: Partial<Profile>) => Promise<Profile>;
    deleteProfileAsync: (profile: Profile) => Promise<void>;
  };
};

const ipcService: IpcService = new IpcService();

const exposedIpcRendererApiApi: IpcRendererApi = {
  UpdatesApi: {
    getAppVersion:                                    ()                                    =>
ipcService.send<AppVersionChannelResponse>(ChannelType[
ChannelType.appVersion ]),
  },
  ActionsApi: {
    sendAppAction: (action: string) => ipcService.send<any>(ChannelType[
ChannelType.appActions ], { params: [action] }),
    removeListeners: (channels: ChannelType[]) => {
      channels.forEach((channel) => {
        ipcRenderer.removeAllListeners(ChannelType[channel]);
      });
      return Promise.resolve();
    },
  },
  StoreApi: {
    getValue: (key: string) =>
```

```
    ipcService.send<StoreGetValueChannelResponse>(ChannelType[
ChannelType.storeGetValue ], { params: [key] }),
  setValue: (key: string, value: string) => ipcService.send(ChannelType[
ChannelType.storeSetValue ], { params: [key, value] }),
 },
 sendPingMessage: async (text) =>
ipcService.send<PingPongChannelResponse>('ping', {
  responseChannel: 'pong',
  params: [text],
 }),
 ProfilesApi: {
  getProfileAsync: (id: string): Promise<Profile> =>
    ipcService.send<Profile>(ChannelType[ChannelType.getProfileAsync],  {
params: [id] }),
  getProfilesAsync: (): Promise<Profile[]> =>
    ipcService.send<Profile[]>(ChannelType[ChannelType.getProfilesAsync]),
  addProfileAsync: (profile: Partial<Profile>): Promise<Profile> =>
    ipcService.send<Profile>(ChannelType[ChannelType.addProfileAsync],  {
params: [profile] }),
  deleteProfileAsync: (profile: Profile): Promise<void> =>
    ipcService.send<void>(ChannelType[ChannelType.deleteProfileAsync],  {
params: [profile] }),
 },
};

export type LoggerApi = {
 logger: LogFunctions;
};

const exposedLoggerApi: LoggerApi = {
```

```
  logger: functions,
};

export type NodeApi = {
  path: {
    join: (...paths: string[]) => string;
    dirname: (p: string) => string;
    normalize: (p: string) => string;
    basename: (p: string, extName?: string) => string;
  };
};

const exposedNodeApi: NodeApi = {
  path: {
    dirname: (p: string) => path.dirname(p),
    join: (...paths: string[]) => path.join(...paths),
    normalize: (p: string) => path.normalize(p),
    basename: (p: string, extName?: string) => path.basename(p, extName),
  },
};

contextBridge.exposeInMainWorld('ipcRendererApi',
exposedIpcRendererApiApi);
contextBridge.exposeInMainWorld('loggerApi', exposedLoggerApi);
contextBridge.exposeInMainWorld('nodeApi', exposedNodeApi);
```

main.ts

```
// eslint-disable-next-line import/no-extraneous-dependencies
import { app, BrowserWindow, dialog, ipcMain, Menu, screen, nativeImage } from
'electron';
```

```
import * as path from 'path';
import * as fs from 'fs';
import { ApplicationMenu } from './application-menu';
import { error, transports } from 'electron-log';
import { ipcChannelArray } from './ipc/ipc.channels';
import { IpcChannelInterface } from './interfaces/ipc/ipc-channel.interface';
export let mainWindow: BrowserWindow = null;
const args = process.argv.slice(1);
const serve = args.some(val => val === '--serve');
const APP_PROTOCOL = 'budget-boss-protocol';

function registerAppProtocol() {
  if (process.defaultApp) {
    if (process.argv.length >= 2) {
      app.setAsDefaultProtocolClient(APP_PROTOCOL,                process.execPath,
[path.resolve(process.argv[1])]);
    }
  } else {
    app.setAsDefaultProtocolClient(APP_PROTOCOL);
  }
}

function doDeepLinking(link?: string) {
  if (!link) {
    link = process.argv.find(a => a.includes(APP_PROTOCOL));
  }
  if (link && link.trim().startsWith(APP_PROTOCOL)) {
    // eslint-disable-next-line @typescript-eslint/no-unused-vars
    const appRequest = link.split('://')[1];
    ///TODO: Open file in app
```

```
  }
}

function loadView() {
  // Path when running electron executable
  let pathIndex = './index.html';

  if (fs.existsSync(path.join(__dirname, '../dist/index.html'))) {
    // Path when running electron in local folder
    pathIndex = '../dist/index.html';
  }

  const indexFileUrl = new URL(path.join('file:', __dirname, pathIndex));
  mainWindow.loadURL(indexFileUrl.href);
}

function createWindow(): BrowserWindow {
  Menu.setApplicationMenu(ApplicationMenu.getAppMainMenu());

  const size = screen.getPrimaryDisplay().workAreaSize;

  // Create the browser window.
  mainWindow = new BrowserWindow({
    width: size.width,
    height: size.height,
    webPreferences: {
      nodeIntegration: false,
      allowRunningInsecureContent: (serve),
      contextIsolation: true,
      preload: path.join(__dirname, './preload.js'),
```

```
    webSecurity: false,
    sandbox: false,
  },
  center: true,
  title: 'Budget Boss',
  minWidth: 800,
  minHeight: 600,
  icon: nativeImage.createFromPath(path.join(__dirname, './favicon.png')),
});
console.log();

if (serve) {
  const debug = require('electron-debug');
  debug();

  require('electron-reloader')(module);
  mainWindow.loadURL('http://localhost:4200');
} else {
  loadView();

  mainWindow.webContents.on('did-fail-load', loadView);
}

// Emitted when the window is closed.
mainWindow.on('closed', () => {
  // Dereference the window object, usually you would store window
  // in an array if your app supports multi windows, this is the time
  // when you should delete the corresponding element.
  mainWindow = null;
});
```

```
  registerAppProtocol();
  doDeepLinking();

  return mainWindow;
}

function registerIpcChannels(ipcChannels: IpcChannelInterface[]) {
  ipcChannels.forEach(channel => ipcMain.on(channel.getName(), (event, request)
=> channel.handle(event, request)));
}

try {
  transports.file.resolvePath = () => __dirname + '/../logs/logs.log';

  const gotTheLock = app.requestSingleInstanceLock();
  if (!gotTheLock) {
    app.quit();
  } else {
    app.on('second-instance', () => {
      // Someone tried to run a second instance, we should focus our window.
      if (mainWindow) {
        if (mainWindow.isMinimized()) {
          mainWindow.restore();
        }
        mainWindow.focus();
        doDeepLinking();
      }
    });
```

```
  // This method will be called when Electron has finished
  // initialization and is ready to create browser windows.
  // Some APIs can only be used after this event occurs.
  // Added 400 ms to fix the black background issue while using transparent window. More detais at https://github.com/electron/electron/issues/15947
  app.on('ready', () => {
    setTimeout(createWindow, 400);

    app.on('open-url', (event, requestUrl) => {
      doDeepLinking(requestUrl);
    });
  });

  // Quit when all windows are closed.
  app.on('window-all-closed', () => {
    // On OS X it is common for applications and their menu bar
    // to stay active until the user quits explicitly with Cmd + Q
    if (process.platform !== 'darwin') {
      app.quit();
    }
  });

  app.on('activate', () => {
    // On OS X it's common to re-create a window in the app when the
    // dock icon is clicked and there are no other windows open.
    if (mainWindow === null) {
      createWindow();
    }
  });
```

```
    registerIpcChannels(ipcChannelArray);
  }
} catch (e) {
  error('App unexpectedly throw an error!', e);
  dialog.showErrorBox('Unexpected error!', `Message: ${e.message}. StackTrace:
${e.stackTrace}`);
  app.exit(1);
}
```

store.ts

```
import * as ElectronStore from 'electron-store';
import { Schema } from 'electron-store';

export interface BudgetBossStore {
  Profiles: string[];
}

const schema: Schema<BudgetBossStore> = {
  Profiles: {
    type: 'array',
    default: [],
  },
};

export class Store {
  private static instance: ElectronStore<BudgetBossStore> = null;

  // eslint-disable-next-line @typescript-eslint/no-empty-function
  private constructor() {
    // empty ctor
```

```
  }

  public static getInstance(): ElectronStore<BudgetBossStore> {
    if (!Store.instance) {
      Store.instance = new ElectronStore<BudgetBossStore>({
        schema,
        name: 'budget-boss-config',
        clearInvalidConfig: true,
        watch: true,
        beforeEachMigration: (store, context) => {
          console.log(`[app] [main-config] migrate from ${context.fromVersion} → ${context.toVersion}`);
        },
        migrations: {
        },
      });
    }

    return Store.instance;
  }
}
```

profiles.channel.ts

```
import { IpcChannelInterface } from '../../../interfaces/ipc/ipc-channel.interface';
import { ChannelType } from '../../../interfaces/ipc/channel-type.enum';
import { IpcRequest } from '../../../interfaces/ipc/ipc-request.interface';
import * as ElectronStore from 'electron-store';
import { BudgetBossStore, Store } from '../../../store/store';
import { Profile } from '../../../interfaces/profile';
import { randomUUID } from 'crypto';
```

```
export class GetProfileChannel implements IpcChannelInterface {
  private store: ElectronStore<BudgetBossStore> = Store.getInstance();

  getName(): string {
    return ChannelType[ChannelType.getProfileAsync];
  }

  handle(event: Electron.CrossProcessExports.IpcMainEvent, request: IpcRequest):
void {
    if (!request.responseChannel) {
      request.responseChannel = `${this.getName()}_response`;
    }

    const profileId = request.params[0];

    const existedProfiles = this.store.get('Profiles').map(value => JSON.parse(value)
as Profile);

    const profile = existedProfiles.find(value => value.id === profileId);

    event.sender.send(request.responseChannel, profile);
  }
}

export class GetProfilesChannel implements IpcChannelInterface {
  private store: ElectronStore<BudgetBossStore> = Store.getInstance();

  getName(): string {
    return ChannelType[ChannelType.getProfilesAsync];
```

```
  }

  handle(event: Electron.CrossProcessExports.IpcMainEvent, request: IpcRequest):
void {
    if (!request.responseChannel) {
      request.responseChannel = `${this.getName()}_response`;
    }

    const existedProfiles = this.store.get('Profiles').map(value => JSON.parse(value)
as Profile);

    event.sender.send(request.responseChannel, existedProfiles);
  }
}

export class AddProfileChannel implements IpcChannelInterface {
  private store: ElectronStore<BudgetBossStore> = Store.getInstance();

  getName(): string {
    return ChannelType[ChannelType.addProfileAsync];
  }

  handle(event: Electron.CrossProcessExports.IpcMainEvent, request: IpcRequest):
void {
    if (!request.responseChannel) {
      request.responseChannel = `${this.getName()}_response`;
    }

    const profile: Partial<Profile> = request.params[0] as Partial<Profile>;
```

```
    const existedProfiles = this.store.get('Profiles').map(value => JSON.parse(value)
as Profile);

    profile.id = randomUUID();
    profile.createdAt = Date.now();

    existedProfiles.push(profile as Profile);

    const mappedProfiles = existedProfiles.map(value => JSON.stringify(value));

    this.store.set('Profiles', mappedProfiles);

    event.sender.send(request.responseChannel, profile);
  }
}

export class DeleteProfileChannel implements IpcChannelInterface {
  private store: ElectronStore<BudgetBossStore> = Store.getInstance();

  getName(): string {
    return ChannelType[ChannelType.deleteProfileAsync];
  }

  handle(event: Electron.CrossProcessExports.IpcMainEvent, request: IpcRequest):
void {
    if (!request.responseChannel) {
      request.responseChannel = `${this.getName()}_response`;
    }

    const profile: Partial<Profile> = request.params[0] as Partial<Profile>;
```

```
    let existedProfiles = this.store.get('Profiles').map(value => JSON.parse(value) as
Profile);

    existedProfiles = existedProfiles.filter(value => value.id !== profile.id);

    const mappedProfiles = existedProfiles.map(value => JSON.stringify(value));

    this.store.set('Profiles', mappedProfiles);

    event.sender.send(request.responseChannel);
  }
}
```

ipc-service.ts

```
// eslint-disable-next-line import/no-extraneous-dependencies
import { IpcRenderer, ipcRenderer } from 'electron';
import { IpcRequest } from '../interfaces/ipc/ipc-request.interface';

export class IpcService {
  private ipcRenderer?: IpcRenderer;

  public send<T>(channel: string, request: IpcRequest = {}): Promise<T> {
    // If the ipcRenderer is not available try to initialize it
    if (!this.ipcRenderer) {
      this.initializeIpcRenderer();
    }
    // If there's no responseChannel let's auto-generate it
    if (!request.responseChannel) {
      request.responseChannel = `${channel}_response_${new Date().getTime()}`;
```

```
    }

    ipcRenderer.send(channel, request);

    // This method returns a promise which will be resolved when the response has arrived.
    return new Promise<T>(resolve => {
      ipcRenderer.once(request.responseChannel, (event, response) => resolve(response));
    });
  }

  private initializeIpcRenderer() {
    this.ipcRenderer = ipcRenderer;
  }
}
```

channel-type.enum.ts

```
export enum ChannelType {
  ping,
  storeGetValue,
  // Updates
  appVersion,
  appActions,
  storeSetValue,
  // Profiles
  getProfileAsync,
  getProfilesAsync,
  addProfileAsync,
  deleteProfileAsync
```

```
}
```

ipc-channel.interface.ts

```
// eslint-disable-next-line import/no-extraneous-dependencies
import { IpcMainEvent } from 'electron';
import { IpcRequest } from './ipc-request.interface';

export interface IpcChannelInterface {
  getName(): string;

  handle(event: IpcMainEvent, request: IpcRequest): void;
}
```

ipc-request.interface.ts

```
export interface IpcRequest {
  responseChannel?: string;

  params?: any[];
}
```