Міністерство освіти і науки України

Одеський національний політехнічний університет

Навчально-науковий інститут комп’ютерних систем

Кафедра системного програмного забезпечення

Шевчук Ігор Олександрович,

студент групи АС-151

**КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА**

Інформаційна система планування сімейного бюджета.

Підтема 2. Клієнтська частина

Спеціальність:

121 – Інженерія програмного забезпечення

Спеціалізація:

Інженерія програмного забезпечення

Керівник:

Писаренко Катерина Олександрівна,

к.т.н., доцент

Одеса – 2020

ЗМІСТ

[ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ 4](#_Toc58348836)

[ЗАВДАННЯ НА РОЗРОБКУ РОЗДІЛУ «ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ» 6](#_Toc58348837)

[АНОТАЦІЯ 7](#_Toc58348838)

[ВСТУП 8](#_Toc58348839)

[1. ПРОБЛЕМИ ІСНУЮЧИХ РІШЕНЬ 10](#_Toc58348840)

[1.1 Аналіз потреб користувачів 10](#_Toc58348841)

[1.2 Аналіз існуючих рішень 12](#_Toc58348842)

[2. РОЗРОБКА ІНФОРМАЦІЙНОЇ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ДЛЯ ПЛАНУВАННЯ БЮДЖЕТУ 15](#_Toc58348843)

[2.1 Підстава для розробки 15](#_Toc58348844)

[2.2 Призначення розробки 15](#_Toc58348845)

[2.3 Аналіз *serverless* архітектури 15](#_Toc58348846)

[3. СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ 20](#_Toc58348847)

[3.1 Визначення бізнес-вимог 20](#_Toc58348848)

[3.2 Визначення вимог користувачів 20](#_Toc58348849)

[3.3 Функціональні вимоги 36](#_Toc58348850)

[3.4 Нефункціональні вимоги 36](#_Toc58348851)

[3.5 Оцінка тривалості розробки 36](#_Toc58348852)

[3.6 Планування розробки 41](#_Toc58348853)

[3.7 Ідентифікація ризиків 41](#_Toc58348854)

[4 ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ 48](#_Toc58348855)

[4.1 Набір інструментальних засобів для розробки програмної системи 48](#_Toc58348856)

[4.2 Опис концептуальних класів 48](#_Toc58348857)

[4.3 Проектування архітектури на основі *serverless* та *serverfull*. 49](#_Toc58348858)

[5 ПРОГРАМНА РЕАЛІЗАЦІЯ І ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ 50](#_Toc58348859)

[5.1 Опис програмних класів 50](#_Toc58348860)

[5.2 Тестування взаємодії *serverless* та *serverfull* елементів 56](#_Toc58348861)

[5.3 Інструкція користувача 61](#_Toc58348862)

[6. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ 68](#_Toc58348863)

[ВИСНОВКИ 70](#_Toc58348864)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 71](#_Toc58348865)

[ДОДАТОК А. ЛІСТИНГ 73](#_Toc58348866)

[ДОДАТОК Б. ПИТАННЯ З ОХОРОНИ ПРАЦІ ТА БЕЗПЕКИ У НАДЗВИЧАЙНИХ СИТУАЦІЯХ 86](#_Toc58348867)

Міністерство освіти і науки України

Одеський національний політехнічний університет

Навчально-науковий інститут комп’ютерних систем

Кафедра системного програмного забезпечення

Рівень вищої освіти: другий (магістерський)

Спеціальність: 121 – Інженерія програмного забезпечення

Спеціалізація: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри СПЗ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Крісілов В. А.

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 року

# **ЗАВДАННЯ**

# **НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Шевчука Ігоря Олександровича, група АС-151

1. Тема роботи: «Інформаційна система планування сімейного бюджета. Підтема 2. Клієнтська частина.»

Керівник роботи: Писаренко Катерина Олександрівна, к.т.н., доцент.

затверджені наказом ректора ОНПУ від « » 20 р. № \_\_\_

2. Зміст роботи: Проблеми існуючих рішень, розробка інформаційної клієнт-серверної системи для планування бюджету, аналіз вимого до програмної системи, конструювання програмного продукту, розділ охорона праці, тестування програмної системи, розгортання програмного продукту.

3. Перелік ілюстративного матеріалу: У відповідності до слайдів електронної презентації.

4. Консультанти розділів роботи

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
| завдання видав | завдання прийняв |
| Охорона праці та безпека в надзвичайних ситуаціях | Москалюк А.Ю. |  |  |

5. Дата видачі завдання: «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів дипломної роботи | Строк виконання етапів роботи | Примітка |
| 1 | Проблеми існуючих рішень |  | Виконано |
| 2 | Розробка інформаційної клієнт-серверної системи для планування бюджету |  | Виконано |
| 3 | Аналіз вимог до програмної системи |  | Виконано |
| 4 | Проектування програмного продукту |  | Виконано |
| 5 | Програмна реалізація та тестування програмної системи |  | Виконано |
| 6 | Розгортання програмного продукту |  | Виконано |
| 7 | Охорона праці та безпека в надзвичайних ситуаціях |  | Виконано |

**Здобувач вищої освіти** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ І. О. Шевчук

**Керівник роботи** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К. О. Писаренко

# **ЗАВДАННЯ**

# на розробку розділу «охорона праці та безпека в надзвичайних ситуаціях»

Шевчука Ігоря Олександровича, група АС-151

Навчально-науковий інститут комп’ютерних систем

Кафедра системного програмного забезпечення

Тема роботи: «Інформаційна система планування сімейного бюджета. Підтема 2. Клієнтська частина.»

Зміст розділу:

1. Аналіз умов праці і вибір заходів і засобів захисту від небезпечних шкідливих виробничих факторів.
2. Аналіз техногенних небезпек і вибір заходів і засобів забезпечення безпеки у надзвичайних ситуаціях.

3. Індивідуальне завдання

Керівник роботи Консультант з охорони праці та БНС

   Писаренко К.О.    Москалюк А.Ю.

« » \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_р. « » \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_р.

АНОТАЦІЯ

Метою роботи є зниження затрат на підтримку програмної системи за рахунок поєднання серверної архітектури з архітектурою безсерверних обчислень. Як результат створено мобільний застосунок для планування бюджету, який для обчислень використовує гібрину архітектуру (поєднання серверної та безсерверної архітектури). Інструментами розробки були мова програмування *Java*, середовище розробки *Android Studio*, інструмент тестування застосунку *Android Virtual Device.*

Ключові слова: *Android*, мобільний застосунок, *serverless*, *serverfull*.

**ABSTARCT**

The aim of the work is to reduce the cost of maintaining the software system by combining the server architecture with the architecture of serverless computing. As a result, a mobile budget planning application has been created that uses a hybrid architecture (a combination of server and serverless architecture) for calculations. The development tools were the Java programming language, the Android Studio development environment, and the Android Virtual Device testing tool.

Keywords: Android, mobile application, serverless, serverfull.

# **ВСТУП**

В наш час, коли відбувається дуже швидкий розвиток технологій, неможливо уявити життя без використання смартфону – кишенькового персонального комп’ютера, який доповнюється функціональністю мобільного телефону. В XXI сторіччі смартфон це річ першої необхідності, адже в багатьох людей кожен день починається з перегляду стрічки в соціальних мережах, перегляду новин або публікації постів. Цей пристрій дозволяє користувачам спілкуватися між собою, заробляти гроші, цікаво проводити час тощо. Наразі існує дві найпоширеніших операційних системи для смартфонів – це *Android* і *iOS.* За статистикою 2019 року кількість активних пристроїв на операційній системі становить більше ніж 2,5 мільярдів пристроїв, а в період з 2019 по 2020 рік в процентному співвідношенні операційна система *Android* має 74.43% користувачів, а операційна система *iOS* – 24.99%, 0,58% – решта мобільних операційних систем.

В цій роботі був розроблений застосунок для планування бюджету. Основою для застосунку була використана архітектура, яка поєднує можливості серверних та безсерверних обчислень, що в свою чергу дозволяє зменшити навантаження на пристрій, а також зменшити витрати на підтримку застосунку.

Актуальністьроботи полягає в розробці архітектури, яка дозволяє знизити витрати на експлуатацію застосунку, яка використовується на клієнтському застосунку.

Метою роботи є зниження затрат на підтримку програмної системи за рахунок поєднання серверної архітектури з архітектурою безсерверних обчислень.

Для досягнення мети були вирішені наступні задачі:

* проаналізувати існуючі архітектури;
* спроектувати архітектуру програмної системи;
* провести тестування розробленої  архітектуру програмної системи;
* розробити клієнтску частину системи.

Об’єктом роботи є дослідження застосування серверної та безсерверної програмної архітектури.

Предметом роботи є застосунок для планування бюджету, який базується на використанні серверної та безсервеної архітектур.

Подальшого розвиткуотримала серверна архітектура, яка розширюється використанням безсерверної архітектури, що дозволяє зменшити витрати на її підтримку. На основі такої архітектури було розроблено інформаційну систему для планування бюджету.

Публікація:

* «Тестування та аналіз результатів використання безсервеної та класичної архітектур в інформаційній системі для планування бюджету» *THEORY, SCIENCE AND PRACTICE Abstracts of III International Scientific and Practical Conference Tokyo, Japan* 2020.
* «Алгоритм вибору методу для прогнозування витрат користувача» *Abstracts of VIII International Scientific and Practical Conference. Prague, Czech Republic 2020.*

Перший розділ присвячений аналізу проблем існуючих рішень. У другому розділі виконано розробку інформаційної клієнт-серверною системи для планування бюджету. У третьому розділі описується аналіз вимог до програмної системи. Четвертий розділ присвячений проектуванню програмного продукту. У п’ятому розділі зроблено програмну реалізацю та проведено тестування системи. У шостому розділі наведено виконання завдання з охорони праці та безпеки у надзвичайних ситуаціях.

**1. ПРОБЛЕМИ ІСНУЮЧИХ РІШЕНЬ**

**1.1 Аналіз потреб користувачів**

Для створення програмного продукту, який буде корисний для споживача, необхідно проаналізувати проблеми користувачів та можливі варіанти вирішення цих проблем. Для цього доцільно використовувати канву ціннісної пропозиції.

Канва ціннісної пропозиції (***Value Proposition Canvas - VPC***) – це шаблон, який складається з двох важливих блоків:

* сегмент користувачів, на які орієнтуємося при розробці продукту;
* ціннісні пропозиції, які надає продукт.

Метою *VPC* є спрощення в пошуку співвідношення між ціннісною пропозицією, яку надає продукт, та потребами користувача, шляхом систематизації та візуалізації основних блоків.

Кожен блок складається з трьох частин, які будуть описані в рамках цієї роботи.

Сегмент користувачів має наступні частини:

1. Завдання користувача (*Customer Jobs*). Тут наводяться задачі, які необхідно виконати користувачу, проблеми які необхідно вирішити або потреби, які необхідно задовольнити. В рамках цієї роботи користувачу необхідно спланувати бюджет та мати інформацію про свої витрати.
2. Болі користувача (*Pains*). Тут наводяться негативні емоції, небажані витрати або ризики, які має користувач під час або після виконання завдання. В нашому випадку до болі користувача відносимо те що йому важко вести статистику бюджету, а також неможливо розпланувати свої витрати.
3. Вигоди користувача (*Gains*). В цій частині наводяться цінності, які користувач бажає отримати від використання застосунку. В нашому випадку сюди відноситься економія часу на планування бюджету та можливість дізнатися стан бюджету в будь-який час.

Після опису блока сегменту користувача необхідно описати блок ціннісної пропозиції, який має наступні частини:

1. Продукти та послуги (*Products & Services*). В цій частині наводяться продукти та послуги з яких сформована ціннісна пропозиція. В нашому випадку це мобільний застосунок, який дозволяє планувати бюджет.
2. Знеболюючі (*Pain Relievers*). Тут наводяться приклади того, як ціннісна пропозиція полегшує болі користувачів. В рамках цієї роботи сюди відноситься автоматичне формування статистики витрат.
3. Створення вигоди (*Gain Creators*). Тут наводяться приклади того, що дозволяє користувачам отримати вигоду від продукту. В нашому випадку сюди відноситься можливість розподілу бюджету на категорії, що значно спрощує його планування та будь-яку іншу взаємодію з ним.

На рисунку 1.1 представлено візуалізацію канви ціннісної пропозиції.

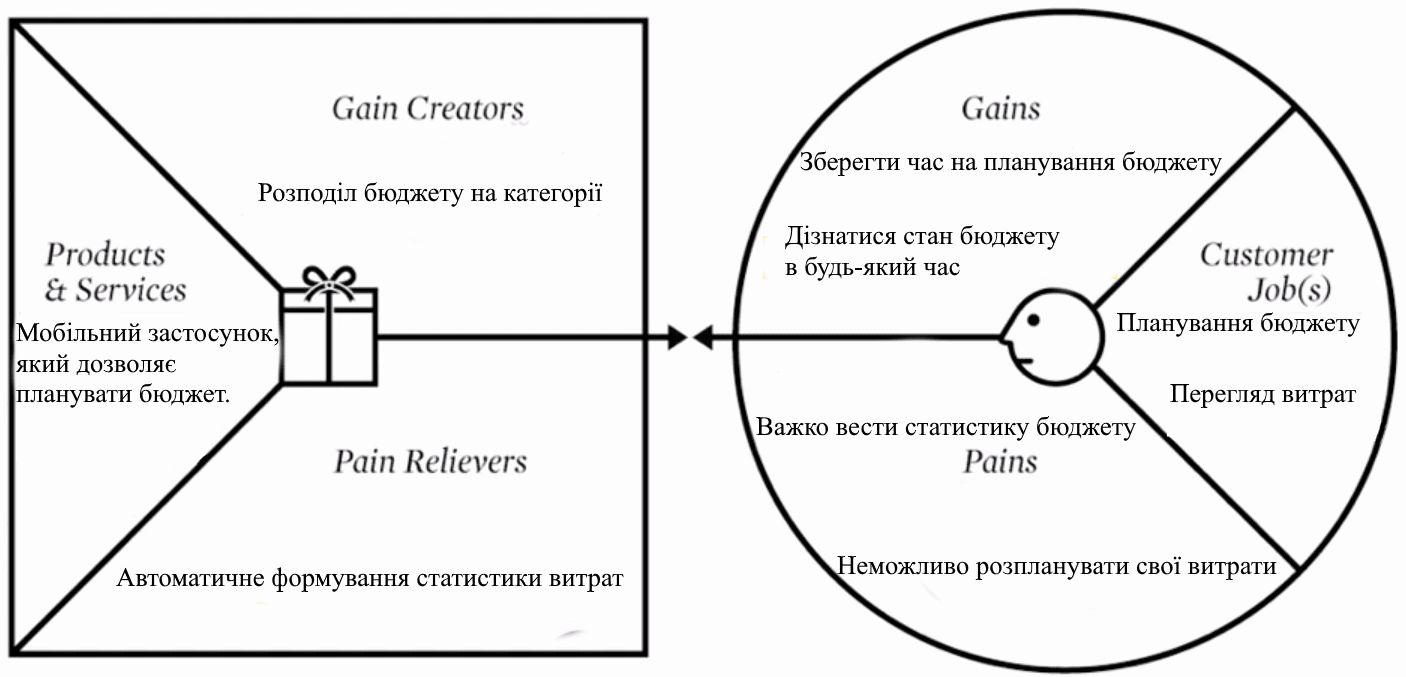
****

Рисунок 1.1 – Канва ціннісної пропозиції

Проаналізувавши отриману канву можна зробити висновок, що користувачу потрібен продукт, який дозволить йому зекономити час на планувати бюджету та ведення статистики своїх витрат. Разом з тим, ми пропонуємо мобільний застосунок, який дозволить розпланувати бюджет та буде надавати статистку витрат, що дозволяє користувачу економити час.

**1.2 Аналіз існуючих рішень**

Проаналізувавши канву ціннісної пропозиції було виявлено декілька існуючих варіантів вирішення проблеми, їх порівняння наводиться в таблиці 1.1.

Таблиця 1.1 Аналіз існуючих рішень

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Паперові облікові журнали | Excel-таблиці | Pocket Guard | YNAB | Spendless | Наша система |
| Ведення групового бюджету | + | + | - | + | + | + |
| Прив'язка до банківського аккаунту | - | - | + | + | - | - |
| Використання інтерактивної колонки Amazon Alexa | - | - | - | - | - | + |
| Безкоштовне розповсюдження та використання | + | - | - | - | + | + |

Як можна побачити з таблиці 1.1, наш застосунок має як прямих так і непрямих конкурентів. Однак, наш програмний застосунок може використовувати інтерактивну колонку Amazon Alexa, що дає змогу застосовувати альтернативний метод використання - голосові команди. Для подальшого аналізу в цій роботі сфокусуємося на тих конкурентах, які представлені мобільними програмними застосунками.

*Pocket Guard* – це мобільний застосунок, який дозволяє відслідковувати свої витрати. До основних можливостей застосунку можна віднести формування статистики витрат та прив’язка до банківського рахунку.

*YNAB* – застосунок, який доступний на різних операційних системах. Він дозволяє планувати бюджет, має можливість прив’язати банківську картку та формувати звіти про витрати.

*Spendless* – мобільний застосунок, який дозволяє вести бюджет як власний так і декількох учасників, додавати геолокацію до даних транзакцій та формувати звіти.

*Pocket Guard* та *YNAB* для своєї роботи потребують версію операційної системи *Android 6.0* або вище, в той час як *Spendless* потребує ОС *Android 4.1* або вище. Усі застосунки в той чи інший спосіб стягують плату за використання.

Враховуючи фактори, які були отримані в процесі аналізу конкурентів, необхідно розробити конкурентну стратегію за допомогою методу стратегії «блакитного океану».

Канва стратегії блакитного океану являє собою лінійний графік, на якому зображаться функції/фактори, які важливі для компанії або організації, а потім на це накладаються конкуренти або галузеві показники. Такими чином, отримаємо інформацію яка допоможе сформулювати конкурентну стратегію.

На горизонтальній осі графіку вказуються фактори, за які ведеться конкуренція та в які необхідно інвестувати, а вертикальна вісь вказує рівень пропозиції, який отримують споживачі за вказаними факторами. Отриманий графік являє собою графічну зображення відносної діяльності компанії за чинниками конкуренції в її галузі.

Таким чином ця канва дозволяє зібрати в одній картинці наступне:

* всі фактори в яких ведеться конкуренція,
* що отримують споживачі,
* стратегічні профілі основних гравців.

Це дає змогу побачити та зрозуміти наскільки схожими є стратегії основних гравців обраної галузі для споживачів та як вони рухають галузь до «червоного океану».

Для створення канви стратегії продукту використовуються наступні фактори:

* кількість людей в бюджетній групі,
* кількість гаманців,
* кількість інтерактивних засобів вводу/виводу,
* вартість використання.

Розмістивши ці фактори на графіку (рис. 1.2) отримаємо візуалізацію стратегій розвитку зазначених конкурентів в порівнянні з нашим застосунком та зможемо сформувати власну стратегію розвитку продукту.

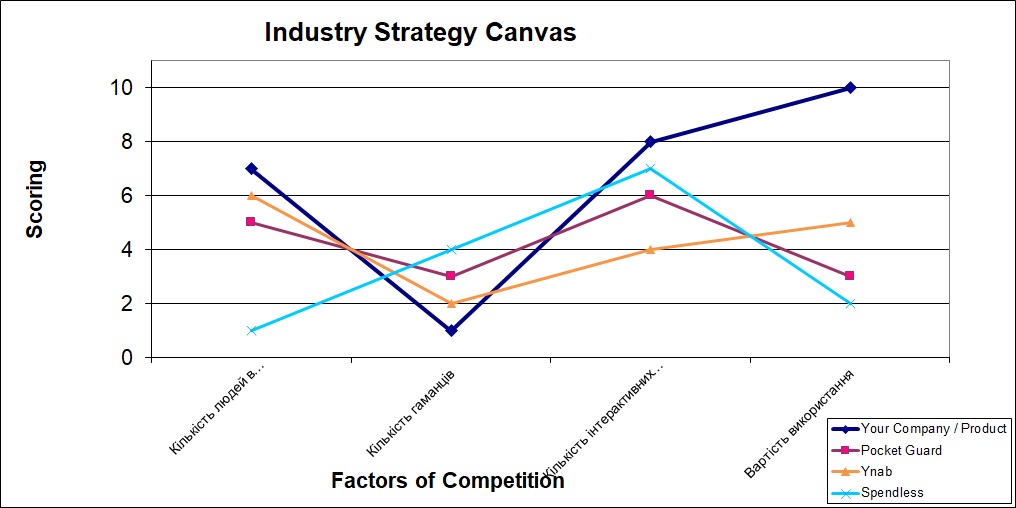
****

Рисунок 1.2 – Графік канви стратегії продукту

Як можна бачити з графіку (рис. 1.2), наш продукт має значні переваги в кількості інтерактивних засобів вводу/виводу, а також у вартості використання.

**2. РОЗРОБКА ІНФОРМАЦІЙНОЇ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ДЛЯ ПЛАНУВАННЯ БЮДЖЕТУ**

**2.1 Підстава для розробки**

Підставою розробки є завдання на дипломне проектування. Темою роботи є «Інформаційна система для планування сімейного бюджету. Підтема №2: Клієнтська частина».

**2.2 Призначення розробки**

Задачею розробки є створення інформаційної системи, яка дозволяє планувати бюджет користувача. Система призначена для планування бюджету, створення звітів про витрати та надходження.

Система повинна надавати користувачу можливість створювати категорії витрат бюджету, створювати бюджет, який доступний групі користувачів, формувати звіти про витрати та надходження, використовувати механізм інтерактивної колонки *Amazon Alexa* для взаємодії з системою. Також система повинна буди спроектована з метою мінімізації витрат на подальше обслуговування, що можна досягти за допомогою поєднання архітектурних стилів, які використовують серверні (*serverfull*) та безсерверні (*serverless*) обчислення.

Для реалізації такої системи необхідно проаналізувати існуючі аналоги рішень, для визначення повного спектру потреб користувачів, проаналізувати архітектури *serverless* та *serverfull* і визначити їх основні переваги та недоліки, а також поєднати архітектури s*erverless* та *serverfull* для досягнення мінімізації витрат на підтримку продукту.

**2.3 Аналіз *serverless* архітектури**

Для того, щоби визначити, як можна поєднати *serverless* та *serverfull* для мінімізації витрат на підтримку продукту, необхідно визначити що являє собою *serverless* архітектура, які можливості вона надає на прикладі конкретного середовища виконання, недоліки та переваги з традиційною *serverfull* архітектурою та можливість їх поєднань.

**2.3.1 Загальний опис *serverless* архітектури.** Збільшення обсягів даних та швидкість їх надходження вимагають створення нових підходів до їх обробки. Так монолітні програми можна розділити на декілька дрібних компонентів (мікросервіси), які можуть виконуватися окремо один від одного. Можливість перетворити запрограмовані функції або методи на хмарні служби, які одразу готові до використання спрощує створення нових функцій та робить робочі процеси швидшими та зручнішими для постачальників послуг застосунків, оскільки зникає необхідність попередньо розподіляти ресурси. Такі переваги надали поштовх тенденції в галузі розробки програмного забезпечення, яка отримала назву безсерверні обчислення, що базується на безсерверній архітектурі.

*Serverless* архітектура – це така архітектура програмних застосунків, що включають сторонні послуги «Бекенд як послуга» (*Backend as a Service* - *BaaS*) та / або які включають власний код, що виконується в керованих контейнерах на платформі «Функції як послуга» (*Function as a service - FaaS*). Використання такого підходу дозволяє позбавити застосунок в потребі мати постійно працюючий серверний компонент. Це дозволяє отримати вигоду від знижених експлуатаційних витрат на підтримання серверного компоненту за рахунок збільшення залежності від сторонніх постачальників послуг. На рівні *FaaS* команди виконуються синхронно або асинхронно за допомогою тригерів. Екземпляри функцій надаються на вимогу через холодний старт або теплий старт за кілька мілісекунд, масштабування відбувається за необхідністю, і плата стягується за виклик та період часу використання ресурсів, що дозволяє отримати практично ідеальну модель ціноутворення на оплату послуг за необхідністю.

Підхід до розробки програмного забезпечення з використанням *serverless* архітектури має свої переваги і недоліки. До переваг такого підхожу можна віднести наступі:

1. Вартість. *Serverless* може бути економічно вигіднішим, ніж придбання або оренда фіксованої кількості серверів, використання яких передбачає значні періоди недостатнього використання їх ресурсів або простою. Це можна описати як «оплата за обчислення», тобто сплачується лише той час та пам’ять, які виділяються для запуску і виконання коду та відсутні витрати на час простою. Безпосередньо отримані вигоди пов’язані з відсутністю витрат на операційні системи, що включає в себе відсутність витрат на ліцензію, встановлення, технічне обслуговування та підтримку.

2. Масштабування. *Serverless* архітектура дозволяє розробникам та операторам системи не витрачати час на налаштування та корегування політики автоматичного масштабування системи. За всі дії, які пов’язані з масштабуванням системи відповідає постачальник хмарних послуг.

3. Продуктивність. Програмний код, який виконується на платформі *FaaS*, являє собою функції, які керуються визначеними подіями. Це дозволяє розробнику не турбуватися про багатопотоковість або безпосередньо обробку *HTTP*-запитів у своєму коді, що дозволяє спростити процес розробки програмного забезпечення.

Однак *serverless* архітектура має і свої недоліки, до яких можна віднести:

1. Час виконання. *Serverless*-код, який рідко використовується має більшу затримку відгуку, ніж код, який постійно працює на виділеному сервері, віртуальній машині або контейнері. Це пов’язано з тим, що на відміну від звичайного автомасштабування, хмарний постачальник зазвичай зменшує або взагалі зупиняє потужності, коли *serverless-*код не використовується, а отже час виконання вимагає певного часу для запуску, шо створює додаткову затримку.

2. Моніторинг та налагодження. Діагностика продуктивності або надмірного використання ресурсів із *serverless-*кодом є більш складно, ніж з кодом, який виконується на традиційних серверах. Незважаючи на те, що *serverless*-функції можуть бути синхронізовані, зазвичай не має можливості отримати більше інформації через під’єднання різноманітних профайлерів, дебаггерів або засобів управління продуктивністю додатків (Application performance management - APM). До того ж, середовище, в якому працює *serverless*-код не являє собою програмне забезпеченням з відкритим кодом, а отже неможливо точно визначити робочі характеристики та відтворити їх в локальному середовищі.

3. Безпека. *Serverless* архітектура є менш захищеною ніж традиційна архітектура. Хоча за безпеку відповідає постачальник хмарних послуг, однак через те, що кожен компонент є точкою входу в *servrless*-застосунок це значно збільшує кількість місць, які можуть стати ціллю атаки. Крім того, рішення безпеки, які використовували клієнти для захисту власних хмарних робочих навантажень стають неактуальними, оскільки клієнти не можуть контролювати та встановлювати будь-які засоби на кінцевих точках (*endpoint*) та на рівні мережі, такі як система виявлення/запобігання вторгненню (intrusion detection/prevention system - IDS/IPS).

4. Блокування постачальника. *Serverless*-обчислення надаються як сторонні послуги. Програми та програмне забезпечення, які працюють в *serverless* середовищі, за замовчуванням заблоковані для певних постачальників хмарних послуг, що може створювати певні проблеми під час міграції.

**2.3.2 *Azure Functions.*** Це сервіс *serverless*-обчислень, який дозволяє виконувати програмний код за допомогою подій-тригерів, без необхідності явного надання або керування інфраструктурою.

*Azure Functions* дозволяє виконувати невеликі частини коду, так звані функції, не турбуючись про інфраструктуру застосунку. Завдяки цій технології хмарна інфраструктура надає сучасні сервери, які необхідні для забезпечення масштабної роботи програми. Всі функції запускаються за певним типом подій-тригерів, які включають в себе відповідь на зміну даних, відповідь на повідомлення, запуск за розкладом або як результат на *HTTP*-запит. До ключових особливостей *Azure Functions* можна віднести такі:

- *Serverless*-застосунки: функції дозволяють розробляти *serverless-*застосунки на платформі *Microsoft Azure*.

- Вибір мови програмування: дозволяє писати функції на одній з мов програмування таких як *C#, Java, JavaScript, Python, and PowerShell.*

- Модель ціноутворення з оплатою за використання: оплачувати потрібно лише час, коли виконується програмний код.

Щодо оплати використання, то *Azure Functions* має три цінових плани, які дозволяють обрати той, що найбільш відповідає потребам:

- План споживання: *Azure* надає необхідні обчислювальні ресурси. Вам не потрібно турбуватися про управління ресурсами, а платити лише за час, який працює код.

- Преміум план: дозволяє вказати скільки розігрітих екземплярів повинні завжди знаходитись в мережі та бути готовими до негайної роботи. Коли функція запускається, *Azure* надає будь-які додаткові обчислювальні ресурси, які потрібні. Оплата стягується за попередньо розігріті екземпляри, які працюють безперервно, а також за всі додаткові екземпляри, які використовуються, коли *Azure* масштабує програму.

- План Служби програм: дозволяє запускати функції як власні веб-програми. Якщо використовується Служба програм для інших застосунків, тоді функції можуть працювати за тим самим планом без додаткових витрат.

**3. СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ**

**3.1 Визначення бізнес-вимог**

Гроші – це невід’ємна частина життя кожної людини, адже вони дозволяють нам отримувати різноманітні товари і послуги. Багато людей хочуть розуміти на що вони витрачають власні кошти. Таке розуміння дозволить планувати свої прибутки та витрати, а також дозволить накопити достатньо грошей на довгоочікувану покупку.

Основні проблеми, які стоять перед користувачем:

* необхідність вести облік власних доходів та витрат,
* аналізувати свої витрати,
* планування свого бюджету.

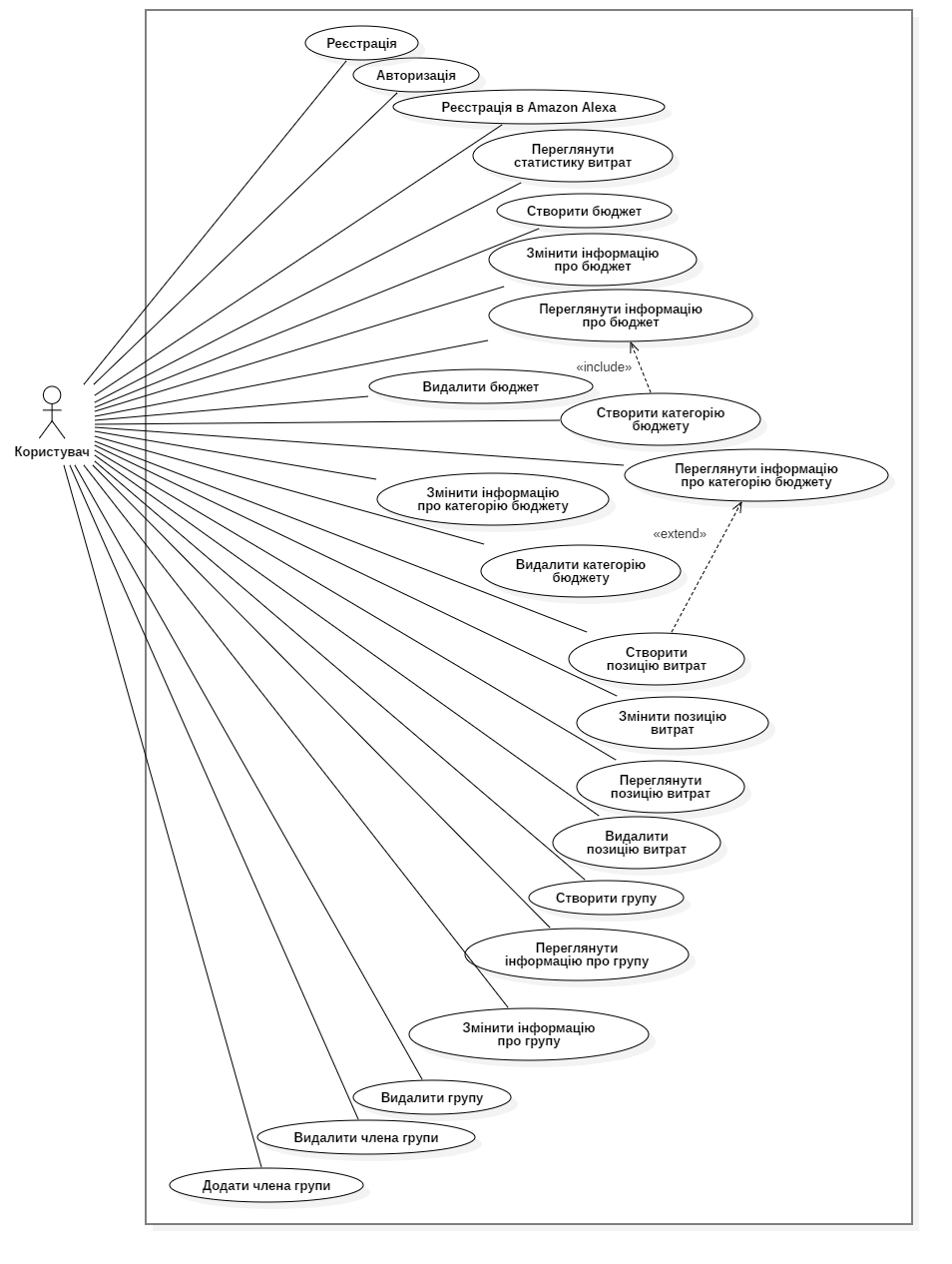
Проаналізувавши основні проблеми, які виникають у користувачів, можна сформулювати наступну ціль для реалізації: створення програмного застосунку, який дозволяє користувачу вести облік власного бюджету та бюджету групи, створювати та переглядати категорії бюджетів, а також переглядати статистику витрат за обраний період часу.

**3.2 Визначення вимог користувачів**

Для опису вимог користувачів використовуються діаграми варіантів використання, а також словесний опис варіантів використання.

**3.2.1 Опис діаграми варіантів використання.** Діаграма варіантів використання наведена на рисунку 3.1. Ця діаграма містить актора Користувач та варіанти використання, які йому доступні.

Користувач – людина, яка використовує наш застосунок для керування бюджетом.

Рисунок 3.1 – Діаграма варіантів використання

Як можна бачити з діаграми варіантів використання, користувачу доступні 22 варіанти використання.

**3.2.2 Опис сценаріїв варіантів використання.** В даному пункті наводиться опис сценаріїв, які були зображені на діаграмі використання.

Варіант використання «Реєстрація»

Основна діюча особа: користувач.

Передумова: користувач запустив застосунок.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: користувач зареєстрований в системі.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить форму для реєстрації нового користувача.
3. Користувач заповнює поля: логін, пароль, підтвердження паролю.
4. Система підтверджує.
5. Дані відправляються на серверну частину.
6. Система отримує дані від серверної частини.
7. Кінець прецеденту.

Альтернативний сценарій:

4а. Введені некоректні дані. Система не підтверджує:

4а1. Система повідомляє, що введені некоректні дані. Перехід до П.3.

5а. Сервер не відповідає:

5а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Авторизація»

Основна діюча особа: користувач.

Передумова: користувач запустив застосування.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: користувач увійшов у систему.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить форму авторизації.
3. Користувач вводить логін. Система підтверджує.
4. Користувач вводить пароль. Система підтверджує.
5. Користувач авторизувався в системі.
6. Кінець прецеденту.

Альтернативний сценарій:

3а. Логін пустий:

3а1. Система повідомляє, що поле логіну пусте. Перехід до П.3.

4а. Пароль пустий:

4а1. Система повідомляє, що поле паролю пусте. Перехід до П.4.

5а. Такої комбінації логіна та пароля не існує:

5а1. Система повідомляє, що дані невірні. Перехід до П.3.

5б. Сервер не відповідає:

5б1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Реєстрація Amazon Alexa»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: користувач зареєструє інтерактивну колонку Amazon Alexa в системі.

Основний сценарій:

1. Користувач звертається до системи.
2. Система запитує про надання коду для реєстрації. Користувач підтверджує.
3. Система відображає код для реєстрації інтерактивної колонки Amazon Alexa.
4. Користувач називає отриманий код інтерактивній колонці Amazon Alexa.
5. Дані відправляються на серверну частину.
6. Система отримує дані від серверної частини. Колонка зареєстрована в системі.
7. Кінець прецеденту.

Альтернативний сценарій:

4а. Названий код невірний. Amazon Alexa не підтверджує:

4а1. Amazon Alexa повідомляє, що наданий код некоректний. Перехід до П.1.

5а. Сервер не відповідає:

5а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту

Варіант використання «Додати новий бюджет»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: додано новий бюджет.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить форму для додавання нового бюджету.
3. Користувач заповнює поля: назва, опис, ціна, дата початку, дата кінця.
4. Система підтверджує.
5. Кінець прецеденту.

Альтернативний сценарій:

4а. Введені некоректні дані. Система не підтверджує:

4а1. Система повідомляє, що введені некоректні дані. Перехід до П.3.

Варіант використання «Змінити інформацію про бюджет»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: інформація про бюджет змінена.

Основний сценарій:

1. Користувач звертається до системи.
2. Система відображає форму з інформацією про бюджет.
3. Користувач змінює дані.
4. Система підтверджує.
5. Дані відправляються на серверну частину.
6. Кінець прецеденту.

Альтернативний сценарій:

4а. Введені некоректні дані. Система не підтверджує:

4а1. Система повідомляє, що введені некоректні дані. Перехід до П.5.

5а. Сервер не відповідає:

5а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Переглянути інформацію про бюджет»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: надано інформацію про бюджет.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список бюджетів.
3. Користувач обирає бюджет.
4. Система виводить інформацію про витрати обраний бюджету.
5. Кінець прецеденту.

Альтернативний сценарій:

2а. Сервер не відповідає:

2а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Видалити бюджет»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: бюджет видалений.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список бюджетів.
3. Користувач обирає бюджет.
4. Система запитує підтвердження. Користувач підтверджує.
5. Дані передаються на сервер. Бюджет видалено.
6. Кінець прецеденту.

Альтернативний сценарій:

4а. Користувач не підтверджує:

4а1. Кінець прецеденту.

5а. Сервер не відповідає:

5а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Додати нову категорію бюджету»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: додано нову категорію бюджету.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить форму для додавання нової категорії бюджету.
3. Користувач заповнює поля: назва, опис.
4. Система підтверджує.
5. Дані відправляються на серверну частину.
6. Система отримує дані від серверної частини.
7. Система виводить нову категорію на екран.
8. Кінець прецеденту.

Альтернативний сценарій:

4а. Введені некоректні дані. Система не підтверджує:

4а1. Система повідомляє, що введені некоректні дані. Перехід до П.3.

6а. Сервер не відповідає:

6а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Змінити інформацію про категорію бюджету»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: інформація про категорію бюджету змінена.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список категорій бюджету.
3. Користувач обирає категорію бюджету.
4. Система відображає форму з інформацією про категорію бюджету.
5. Користувач змінює дані.
6. Система підтверджує.
7. Дані відправляються на серверну частину.
8. Кінець прецеденту.

Альтернативний сценарій:

6а. Введені некоректні дані. Система не підтверджує:

6а1. Система повідомляє, що введені некоректні дані. Перехід до П.5.

7а. Сервер не відповідає:

7а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Переглянути інформацію про категорію бюджету»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: надано інформацію про категорію бюджету.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список категорій бюджету.
3. Користувач обирає категорію бюджету.
4. Система виводить інформацію про витрати обраної категорії бюджету.
5. Кінець прецеденту.

Альтернативний сценарій:

2а. Сервер не відповідає:

2а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Видалити категорію бюджету»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: категорія бюджету видалена.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список категорій бюджету.
3. Користувач обирає сторінку категорію бюджету.
4. Система запитує підтвердження. Користувач підтверджує.
5. Дані передаються на сервер. Категорію бюджету видалено.
6. Кінець прецеденту.

Альтернативний сценарій:

4а. Користувач не підтверджує:

4а1. Кінець прецеденту.

5а. Сервер не відповідає:

5а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Додати нову позицію витрат»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: додано нову позицію витрат.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список категорій бюджету.
3. Користувач обирає категорію бюджету в яку необхідно додати позицію витрат.
4. Система виводить форму для додавання нової позиції витрат.
5. Користувач заповнює поля: ціну, назву, опис.
6. Система підтверджує.
7. Дані відправляються на серверну частину.
8. Система отримує дані від серверної частини.
9. Кінець прецеденту.

Альтернативний сценарій:

6а. Введені некоректні дані. Система не підтверджує:

6а1. Система повідомляє, що введені некоректні дані. Перехід до П.4.

8а. Сервер не відповідає:

8а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Переглянути інформацію про позицію витрат»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: надано інформацію про позицію витрат.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список категорій бюджету.
3. Користувач обирає категорію бюджету.
4. Система виводить інформацію про витрати обраної категорії бюджету.
5. Користувач обирає позицію витрат.
6. Система виводить інформацію про обрану позицію витрат.
7. Кінець прецеденту.

Альтернативний сценарій:

2а. Сервер не відповідає:

2а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Видалити позицію витрат»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху позиція витрат видалена.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список категорій бюджету.
3. Користувач обирає категорію бюджету.
4. Система виводить список позицій витрат.
5. Користувач обирає позицію витрат для видалення.
6. Система запитує підтвердження. Користувач підтверджує.
7. Дані передаються на сервер. Позицію витрат видалено.
8. Кінець прецеденту.

Альтернативний сценарій:

5а. Користувач не підтверджує:

5а1. Кінець прецеденту.

6а. Сервер не відповідає:

6а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Змінити інформацію про позицію витрат»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: інформація про позицію витрат змінена.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список категорій бюджету.
3. Користувач обирає категорію бюджету.
4. Система виводить список позицій витрат.
5. Користувач обирає позицію витрат.
6. Система відображає форму з інформацією про позицію витрат.
7. Користувач змінює дані.
8. Система підтверджує.
9. Дані відправляються на серверну частину.
10. Кінець прецеденту.

Альтернативний сценарій:

7а. Введені некоректні дані. Система не підтверджує:

7а1. Система повідомляє, що введені некоректні дані. Перехід до П.5.

9а. Сервер не відповідає:

9a1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Додати нову групу»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: додано нову групу.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить форму для створення нової групи.
3. Користувач заповнює поля: назва, опис.
4. Система підтверджує.
5. Дані відправляються на серверну частину.
6. Система отримує дані від серверної частини.
7. Кінець прецеденту.

Альтернативний сценарій:

6а. Сервер не відповідає:

6а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Переглянути інформацію про групу»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: надано інформацію про групу.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список груп.
3. Користувач обирає групу.
4. Система виводить інформацію про обрану групу.
5. Кінець прецеденту.

Альтернативний сценарій:

2а. Сервер не відповідає:

2а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Видалити групу»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: група видалена.

Основний сценарій:

Користувач звертається до системи.

Система виводить список категорій бюджету.

1. Користувач звертається до системи.
2. Система виводить список груп.
3. Користувач обирає групу для видалення.
4. Система запитує підтвердження. Користувач підтверджує.
5. Дані передаються на сервер. Групу видалено.
6. Кінець прецеденту.

Альтернативний сценарій:

4а. Користувач не підтверджує:

5а1. Кінець прецеденту.

5а. Сервер не відповідає:

5а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Змінити інформацію про групу»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: інформація про групу змінена.

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список груп.
3. Користувач обирає групу.
4. Система відображає форму з інформацією про групу.
5. Користувач змінює дані.
6. Система підтверджує.
7. Дані відправляються на серверну частину.
8. Кінець прецеденту.

Альтернативний сценарій:

6а. Введені некоректні дані. Система не підтверджує:

6а1. Система повідомляє, що введені некоректні дані. Перехід до П.5.

7а. Сервер не відповідає:

7a1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Додати члена групи»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: додано члена групи

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список груп.
3. Користувач обирає групу.
4. Система відображає форму з інформацією про групу.
5. Користувач вводить інформацію про нового члена групи.
6. Система підтверджує.
7. Дані відправляються на серверну частину.
8. Кінець прецеденту.

Альтернативний сценарій:

6а. Введені некоректні дані. Система не підтверджує:

6а1. Система повідомляє, що введені некоректні дані. Перехід до П.5.

7а. Сервер не відповідає:

7a1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Видалити члена групи»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: видалений член групи

Основний сценарій:

1. Користувач звертається до системи.
2. Система виводить список груп.
3. Користувач обирає групу.
4. Система відображає форму з інформацією про групу.
5. Користувач обирає члена групи для видалення.
6. Система запитує підтвердження. Користувач підтверджує.
7. Дані відправляються на серверну частину.
8. Кінець прецеденту.

Альтернативний сценарій:

6а. Введені некоректні дані. Система не підтверджує:

6а1. Система повідомляє, що введені некоректні дані. Перехід до П.5.

7а. Сервер не відповідає:

7a1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

Варіант використання «Переглянути статистику витрат»

Основна діюча особа: користувач.

Передумова: користувач увійшов в систему.

Мінімальні гарантії: нічого не відбувається.

Гарантії успіху: надано інформацію про витрати.

Основний сценарій:

1. Користувач звертається до системи.
2. Система звертається до сервера.
3. З сервера надходять дані про витрати.
4. Система виводить інформацію статистику витрат.
5. Кінець прецеденту.

Альтернативний сценарій:

2а. Сервер не відповідає:

2а1. Система повідомляє, що немає зв’язку з сервером. Кінець прецеденту.

## **3.3** **Функціональні вимоги**

Система повинна задовольняти наступним функціональним вимогам:

* Користувач повинен мати можливість реєструватися в системі.
* Користувач повинен мати можливість авторизуватися в системі.
* Користувач повинен мати можливість керувати бюджетом.
* Користувач повинен мати можливість керувати категоріями бюджету.
* Користувач повинен мати можливість керувати позиціями витрат.
* Користувач повинен мати можливість керувати групами.
* Користувач повинен мати можливість додавати членів групи.
* Користувач повинен мати можливість видаляти членів групи.

**3.4 Нефункціональні вимоги**

Система повинна задовольняти наступним нефункціональним вимогам:

* Програмне рішення має працювати на ОС *Android 4.4*.
* Час відгуку клієнта повинен бути не більше 2 с.

**3.5 Оцінка тривалості розробки**

Оцінка тривалості розробки виконується за методом *UCP* (*Use Case Points*). Цей метод дозволяє оцінити тривалість проекту, на основі варіантів використання.

В таблиці 3.1 представлено оцінку складності інтерфейсів системі на основі акторів.

Таблиця 3.1 – Оцінка складності інтерфейсів на основі акторів

|  |  |  |  |
| --- | --- | --- | --- |
| Актор | Примітка | Складність | Вага |
| Користувач | Взаємодія за допомогою *GUI*-інтерфейсу | Складний | 3 |

В таблиці 3.2 представлено оцінка масштабів системи за допомогою визначення типу варіантів використання. Тип варіантів використання визначається відповідно до кількості транзакцій.

Таблиця 3.2 – Оцінка типів варіантів використання

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Варіант використання | Складність | Вага |
| 1 | Авторизація | Простий | 5 |
| 2 | Реєстрація | Простий | 5 |
| 3 | Реєстрація в Amazon Alexa | Простий | 5 |
| 4 | Переглянути інформацію про бюджет | Простий | 5 |
| 5 | Додати нову категорію бюджету | Простий | 5 |
| 6 | Змінити інформацію про категорію бюджету | Простий | 5 |
| 7 | Переглянути інформацію про категорію бюджету | Простий | 5 |
| 8 | Додати нову позицію витрат | Простий | 5 |
| 9 | Змінити інформацію про позицію витрат | Простий | 5 |
| 10 | Переглянути інформацію про позицію витрат | Простий | 5 |
| 11 | Змінити баланс бюджету | Простий | 5 |
| 12 | Видалити категорію бюджету | Простий | 5 |
| 13 | Видалити позицію витрат | Простий | 5 |
| 14 | Створити групу | Простий | 5 |
| 15 | Змінити інформацію про групу | Простий | 5 |
| 16 | Переглянути інформацію про групу | Простий | 5 |
| 17 | Видалити групу | Простий | 5 |
| 18 | Переглянути статистику витрат | Простий | 5 |
| 19 | Змінити інформацію про бюджет | Простий | 5 |
| 20 | Переглянути інформацію про бюджет | Простий | 5 |
| 21 | Додати члена групи | Простий | 5 |
| 22 | Видалити члена групи | Простий | 5 |

Оцінка акторів (*UAW*) обчислюється за формулою (3.1).

(3.1)

де *SA* – кількість «простих» акторів (*Simple actors*);

*AA* – кількість «середніх» акторів (*Average actors*);

*CA* – кількість «складних» акторів (*Complex actors*).

Нескоректована оцінка варіантів використання (*UUCW*) обчислюється за формулою (2.2).

(3.2)

де *SUC* – кількість «простих» варіантів використання (*Simple Use Cases*);

*AUC* – кількість «середніх» варіантів використання (*Average Use Case*);

*CUC* – кількість «складних» варіантів використання (*Complex Use Cases*).

*UCP* розраховується за формулою (3.3).

(3.3)

де *UAW* – оцінка акторів;

*UUCW* – оцінка варіантів використання.

Після розрахунку *UCP* необхідно оцінити технічні фактори (таблиця 3.3) та складність архітектури системи. Оцінка виконується за шкалою від 0 до 5, де 0 означає відсутність впливу, 5 – сильний вплив на розробку.

Таблиця 3.3 – Оцінка технічних факторів системи

|  |  |  |  |
| --- | --- | --- | --- |
| Фактор | Опис | Вага | Оцінка |
| T1 | Розподіленість системи | 2 | 1 |
| T2 | Час відгуку | 1 | 3 |
| T3 | Ефективність кінцевого користувача | 1 | 5 |
| T4 | Складність обробки | 1 | 4 |
| T5 | Фокус на повторному використанні коду | 1 | 2 |
| T6 | Простота інсталяції | 0,5 | 2 |
| T7 | Простота використання | 0,5 | 5 |
| T8 | Портативність | 2 | 4 |
| T9 | Простота змін | 1 | 5 |
| T10 | Паралельні обчислення | 1 | 0 |
| T11 | Засоби захисту | 1 | 2 |
| T12 | Доступ до третьої сторони | 1 | 1 |
| T13 | Потреби в спеціальному навчанні | 1 | 3 |

Сума добутків вагових коефіцієнтів та оцінок для кожного з технічних факторів визначає показник *TFactor*. Отже, *TFactor =* 38.5 Оцінка значення *TCF* обчислюється за формулою (3.4).

(3.4)

де *TFactor* – cума добутків вагових коефіцієнтів та оцінок для кожного з технічних факторів;

Після оцінки технічних факторів необхідно оцінити зовнішні фактори (табл.3.4).

Таблиця 3.4 – Оцінка зовнішніх факторів

|  |  |  |  |
| --- | --- | --- | --- |
| Фактор | Опис | Вага | Оцінка |
| F1 | Знайомство з процесом розробки | 1,5 | 5 |
| F2 | Досвід подібних проектів | 0,5 | 4 |
| F3 | Досвід об’єктно-орієнтованої розробки | 1 | 3 |
| F4 | Досвідченість провідного аналітика | 0,5 | 4 |
| F5 | Мотивація | 1 | 4 |
| F6 | Стабільність вимог | 2 | 4 |
| F7 | Часткова зайнятість працівників | -1 | 4 |
| F8 | Складність мови програмування | -1 | 5 |

Сума добутків вагових коефіцієнтів та оцінок для кожного з зовнішніх факторів визначає показник *EFactor*. Отже, *EFactor* = 17,5 Оцінка значення *ECF* обчислюється за формулою (3.5).

(3.5)

де *EFactor* – сума добутків вагових коефіцієнтів та оцінок для кожного з зовнішніх факторів;

Скоректовані UCP обчислюються за формулою (3.6).

(3.6)

де *UUCW –* нескоректована оцінка варіантів використання;

*UAW* – оцінка акторів;

*TCF* – фактор технічної складності;

*ECF* – фактор зовнішньої складності.

Так як, одному *UCP* відповідає 36 годин, то кількість годин на розробку обчислюється за формулою (3.7):

(3.7)

де *UCP* – оцінка варіантів використання

Отже, в результаті оцінки тривалості розробки ми отримали 2454,12 год.

**3.6 Планування розробки**

Для створення графіку робіт використовується діаграма Ганта, яка зображена на рис. 3.2.

## **3.7 Ідентифікація ризиків**

Для того, щоб розробка програмного продукту завершилася вчасно необхідно виявити ризики, які можуть вплинути на проект та визначити як на ці ризики реагувати. Після аналізу завдань проекту, будо ідентифіковано наступні ризики:

1. Неякісне проектування
2. Часті зміни вимог
3. Переоцінка сил
4. Неякісне тестування
5. Відсутність мотивації
6. Прокрастинація

7. Брак досвіду

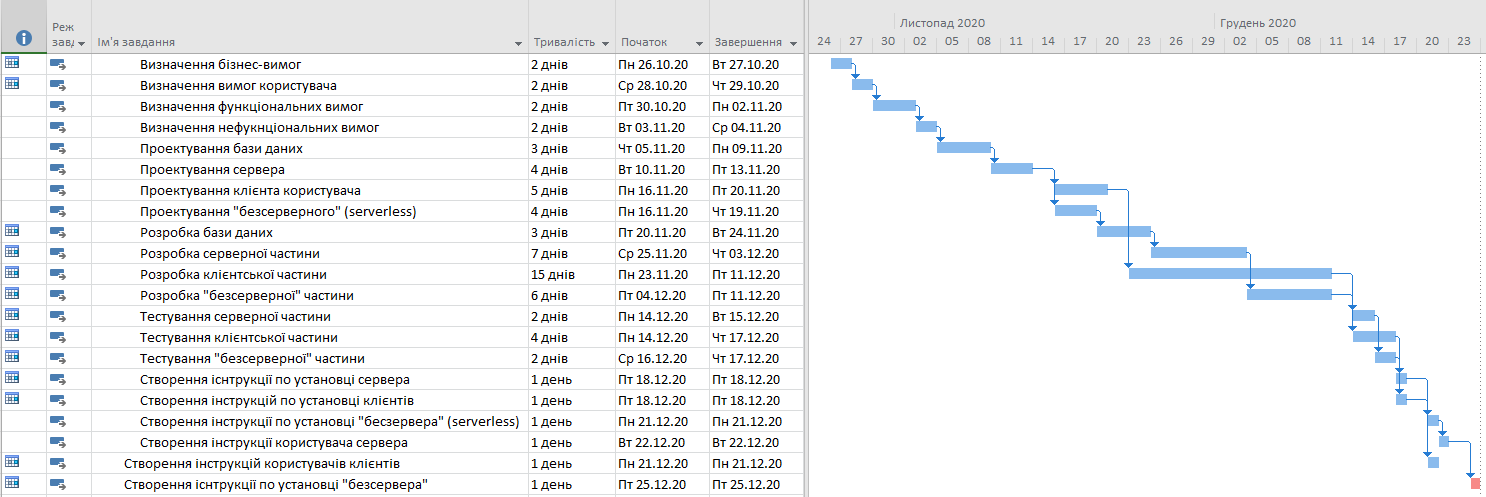


Рисунок 3.2 – Діаграма Ганта

Після того, як ризики булу ідентифіковані, необхідно дати їм детальну характеристику (табл. 3.5 - 3.11) та визначити ті з них, які мають найбільший вплив на проект.

Таблиця 3.5 – Ризик «Неякісне проектування»

|  |  |
| --- | --- |
| Назва: Неякісне проектування | Категорія: Технологічний |
| Причина: Нестача кваліфікованих кадрів | Умови: Архітектор проекту не проектував подібні системи |
| Наслідки: Низька продуктивність розробки | Вплив: Збільшення термінів і трудомісткості розробки |
| Імовірність: Висока | Ступінь впливу: Критичний |
| Близькість: Дуже скоро | Ранг: 9 |
| Вихідні дані: Архітектура проекту | |

Таблиця 3.6 – Ризик «Часті зміни вимог»

|  |  |
| --- | --- |
| Назва: Часті зміни вимог | Категорія: Соціальний |
| Причина: Відсутність документації вимог | Умови: Низький рівень взаємодії з замовником |
| Наслідки: Низька продуктивність розробки | Вплив: Збільшення термінів розробки і трудомісткості розробки |
| Імовірність: Висока | Ступінь впливу: Критичний |
| Близькість: Дуже скоро | Ранг: 9 |
| Вихідні дані: "Вимоги проекту" | |

Таблиця 3.7 – Ризик «Відсутність мотивації»

|  |  |
| --- | --- |
| Назва: Відсутність мотивації | Категорія: Соціальний |
| Причина: Розробка нецікавого проекту | Умови: Не ефективний менеджмент |
| Наслідки: Низька продуктивність розробки | Вплив: Збільшення термінів розробки |
| Імовірність: Низька | Ступінь впливу: Середній |
| Близькість: Не скоро | Ранг: 1 |
| Вихідні дані: "План проекту" | |

Таблиця 3.8 – Ризик «Переоцінка сил»

|  |  |
| --- | --- |
| Назва: Переоцінка сил | Категорія: Соціальний |
| Причина: Нестача кваліфікованих кадрів | Умови: Команда проекту не розробляла подібні системи |
| Наслідки: Низька продуктивність розробки | Вплив: Збільшення термінів розробки і трудомісткості розробки |
| Імовірність: Висока | Ступінь впливу: Критичний |
| Близькість: Скоро | Ранг: 9 |
| Вихідні дані: "План проекту" | |

Таблиця 3.9 – Ризик «Неякісне тестування»

|  |  |
| --- | --- |
| Назва: Неякісне тестування | Категорія: Технологічний |
| Причина: Нестача кваліфікованих кадрів | Умови: не достатньо досвіду в тестуванні подібних систем |
| Наслідки: Збільшення вартості виправлення багів | Вплив: Велика кількість не виявлених багів |
| Імовірність: Висока | Ступінь впливу: Критична |
| Близькість: Не скоро | Ранг: 9 |
| Вихідні дані: "Тест план", "Тест кейси", "Чек-листи" | |

Таблиця 3.10 – Ризик «Неякісне тестування»

|  |  |
| --- | --- |
| Назва: Брак досвіду | Категорія: Соціальний |
| Причина: Нестача кваліфікованих кадрів | Умови: Команда не розробляла подібні системи |
| Наслідки: Низька продуктивніть розробки | Вплив: Збільшення термінів розробки і трудомісткості розробки |
| Імовірність: Висока | Ступінь впливу: Критичний |
| Близькість: Скоро | Ранг: 9 |
| Вихідні дані: "План проекту" | |

Таблиця 3.11 – Ризик «Прокрастинація»

|  |  |
| --- | --- |
| Назва: Прокрастинація | Категорія: Технологічний |
| Причина: Відсутність мотивації | Умови: Не ефективний менеджмент |
| Наслідки: Низька продуктивність розробки | Вплив: Збільшення термінів розробки і трудомісткості розробки |
| Імовірність: Низька | Ступінь впливу: Середній |
| Близькість: Не скоро | Ранг: 1 |
| Вихідні дані: "План проекту" | |

Для ризику «Неякісне проектування» розробляється діаграма Ішикави (рис. 3.3) та рекомендації для попередження ризику.



Рисунок 3.3 – Неякісне проектування

Реагування: Ухилення від ризику

Для захисту проекту від цього ризику необхідно сформувати чіткі вимоги перед початком проектування, а також поліпшити якість документації.

# **4 ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ**

## **4.1 Набір інструментальних засобів для розробки програмної системи**

Для розробки програмного продукту використовувалась інтегроване середовище розробки *Android Studio*, яке надає можливість розробляти мобільні застосунки, які працюють на операційній системи *Android,* використовуючи мову програмування *Java*.

## **4.2 Опис концептуальних класів**

Спираючись на існуючі варіанти використання було розроблено діаграму концептуальних класів, що зображено на рисунку 4.1.

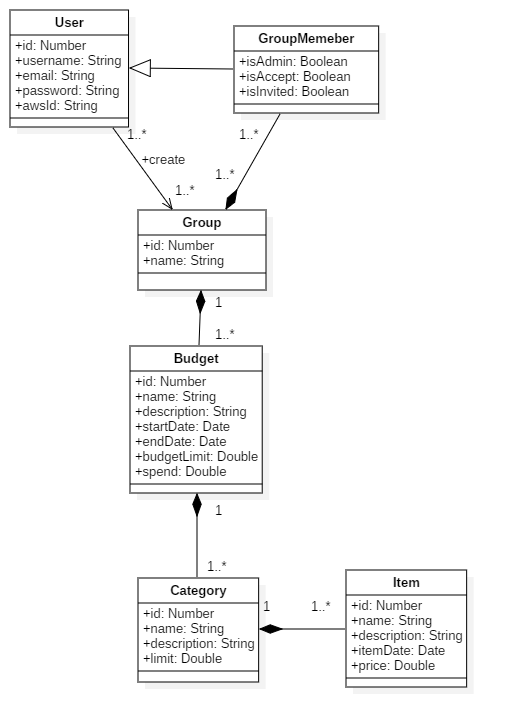


Рисунок 4.1 – Діаграма концептуальних класів

На цій діаграмі зображений клас користувача (*User*), від якого успадковується клас учасника групи (*GroupMember*). Користувач може створювати групу (*Group*), яка має бюджет (*Budget*). В бюджеті можна створювати категорії (*Category*) в які додаються витрати користувача (*Item*).

## **4.3 Проектування архітектури на основі *serverless* та *serverfull*.**

Для створення застосунку, який буде дешевий в подальшій підтримці, але в той же час буде задовольняти користувачів своєю швидкодією да іншими параметрами, необхідно під час розробки спиратися на переваги *serverless* архітектури, в той же час перекривати недоліки можливостями *serverfull* архітектури.

Для зменшення вартості експлуатації та підтримки застосунку можливості *serverless* будуть використовуватися для виконання частини коду, який повинен виконуватися або за розкладом, або не передбачає частий виклик.

Для операцій, які передбачають частий виклик, або які повинні виконувати довготривалі обчислення буде використаний *serverfull*, тобто традиційний сервер.

Враховуючи той факт, що в цій роботі клієнтським застосунком є мобільний застосунок, то такий підхід дозволить з одного боку перенести обчислення з використанням великого обсягу даних або довготривалі обчислення частково на постачальника хмарних послуг, а частково на традиційну серверну частину. Обробка частини даних на мобільному пристрої дозволить не перевантажувати мережу, а також дозволить не витрачати ресурси сервера на такі обчислення, що в свою чергу зменшить необхідну кількість традиційних серверів для експлуатації та підтримці системи.

Отже, можна припустити, що використання такого підходу для розробки програмного забезпечення дозволяє знизити витрати за рахунок використання *serverless*.

# **5 ПРОГРАМНА РЕАЛІЗАЦІЯ І ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ**

## **5.1 Опис програмних класів**

Програмний продукт передбачає взаємодію з користувачем за допомогою графічного інтерфейсу, тому для розробки мобільного застосунку доцільно буде спиратися на шаблон проектування «Модель–Представлення–Пред'явник» (*Model-View-Presenter – MVP*). Такий підхід дозволяє відділити візуальне відображення даних та поведінку застосунку у різні класи, що полегшує тестування та підтримку застосунка.

Модель (*Model*) – клас, який призначений для визначення даних з якими буде відбуватися взаємодія користувача, або які будуть відображатися в інтерфейсі користувача. Тобто тут реалізується робота с даними, наприклад така як запит даних з серверу, читання файлу, збереження даних в базу даних, тощо.

Представлення (*View*) – це клас, який відповідає за відображення даних на екран та взаємодію з користувачем. Усі дії які робить користувач передаються до Пред’явника, який вже вирішує що робити в залежності від тієї чи іншої дії.

Пред’явник (*Presenter*) – клас, який зв’язує Модель та Представлення, оскільки вони не повинні взаємодіяти один з одним напряму. Пред’явник отримує від Представлення інформацію про дії користувача та вирішує що потрібно робити далі. Якщо необхідно отримати якісь дані, для цього Пред’явник використовує Модель, а якщо треба щось відобразити тоді він повідомляє про це Представленню.

В своїй роботі Пред’явник керує Моделлю та Представленням через відповідні інтерфейси, а не конкретну реалізацію. Це дозволяє полегшити розуміння та підтримку коду, а також спрощує процес тестування.

На рисунку 5.1 зображені пакети класів, які складають загальну схему взаємодії компонентів в *MVP*-шаблоні. Як можна бачити з цієї схеми, компоненти з пакету *View* приймають дії користувача та сповіщають про них компоненти з пакету *Presenter*. *Presenter* отримає дані про дії користувача та вирішує, що необхідно робити далі. Якщо від користувача були отримані якісь дані, тоді *Presenter* надсилає команду на зміну моделі та отримає її оновлений стан, після чого оновлює компонент *View*.

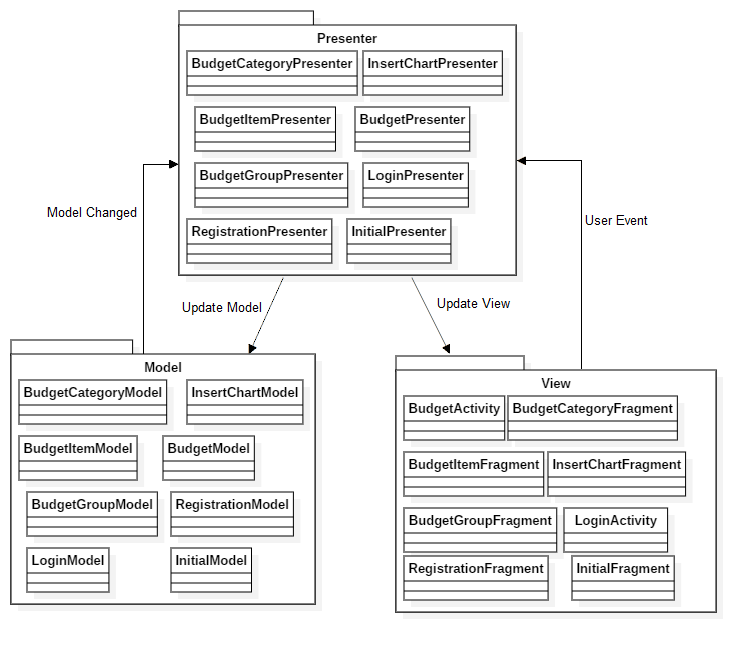


Рисунок 5.1 – Діаграма пакетів класів на основі шаблону *MVP*

На рисунку 5.2 представлена діаграма класів, яка демонструє взаємодію між *MVP-*компонентами, на основі класів, які відповідають за роботу з позиціями витрат бюджету (*items*).

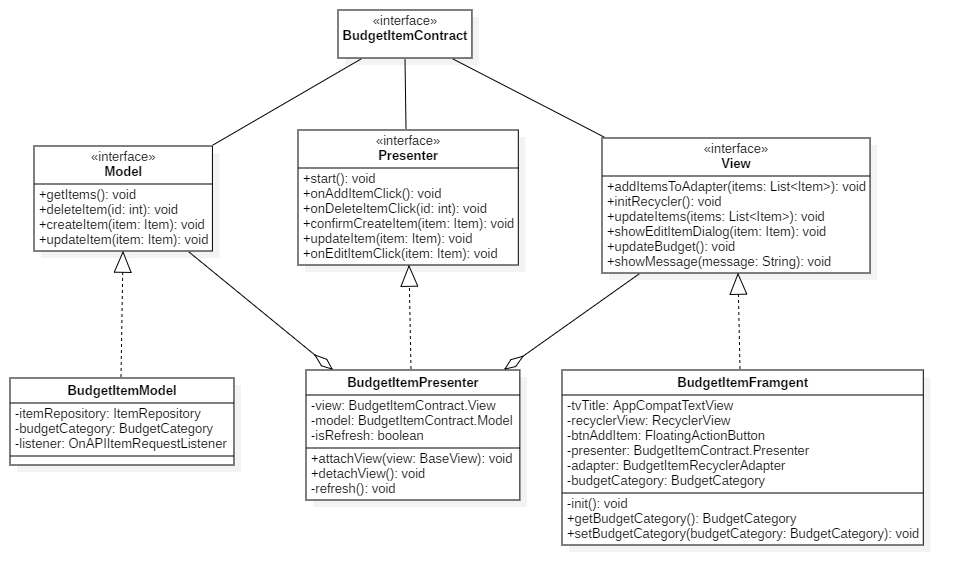


Рисунок 5.2 – Діаграма класів взаємодії *item*-компонентів

Контракт (*Contract)* – інтерфейс, в якому описується взаємодія між компонентами *MVP*.

Як можна бачити з діаграми (рис. 5.2) в нас є Контракт *BudgetItemContract,* який описує взаємодію трьох інтерфейсів *Model*, *View* і *Presenter*, які є компонентами шаблону *MVP*. Кожен з цих інтерфейсів має перелік методів, які необхідні для роботи, та мають відповідні реалізації у вигляді класів *BudgetItemModel*, *BudgetItemPresenter* та *BudgetItemFragment*.

*BudgetItemModel –* клас, який відповідає за дані та реалізує інтерфейс *Model.* В цьому класі відбувається звернення до сервера для отримання даних та передача цих даних до який реалізує інтерфейс *Presenter*. Методи, які реалізує клас:

* *getItems()* – отримання списку об’єктів *item* від сервера;
* *deleteItem(id: int)* – видалення вказаного *item* за його *id*;
* *createItem(item: Item)* – запис в базу даних сервера нового *item*;
* *updateItem(item:Item) –* оновлення інформації про обраний *item.*

Атрибути, які має клас:

* *itemRepository* – екземпляр класу репозиторія, який відповідає за взаємодію між застосунком та сервером і обробкою *item-*об’єктів;
* *budgetCategory* – екземпляр класу *BudgetCategory,* який містить інформацію про поточну категорію, яку обрав користувач;
* *listener* – реалізація інтерфейсу *OnAPIItemRequestListener*, який відповідає потрібен для обробки результатів асинхронних методів.

*BudgetItemPresenter –* клас, який відповідає за взаємодію Моделі та Представлення і реалізує інтерфейс *Presenter*. Методи, які реалізує клас:

* *start()* – метод, який викликається при створенні Представлення та виконує завантаження початкових даних на екран;
* *onAddItemClick()* – повідомляє, що користувач натиснув на кнопку додавання нового об’єкта;
* *onDeleteItemClick(id: int)* – повідомляє, що користувач натиснув на кнопку видалення певного об’єкта;
* *confirmCreateItem(item:Item) –* повідомляє, що користувач сформував параметри нового об’єкта та натиснув на кнопку підтвердження його створення*;*
* *updateItem(item: Item) –* повідомляє, що користувач змінив параметри об’єкта та натиснув на кнопку підтвердження його зміни*;*
* *onEditItemClick(item: Item) –* повідомляє, що користувач натиснув на кнопку зміни певного об’єкта*.*
* *attachView(view: BaseView) –* метод, який прикріплює до Пред’явника певний об’єкт Представлення*;*
* *detachView() –* метод, який відкріплює поточний об’єкт Представлення від Пред’явника*;*
* *refresh()* – метод, який надсилає команду для оновлення поточного Представлення*.*

Атрибути, які має клас:

* *view* – екземпляр класу BudgetItemContract.View;
* *model* – екземпляр класу *BudgetItemContract.Model*;
* *isRefresh* – змінна, яка вказує чи потрібно оновлювати поточні дані або завантажувати ті, які є в Моделі.

*BudgetItemFramgent –* клас, який відповідає за представлення даних на екрані та реалізує інтерфейс *View*. Методи, які реалізує клас:

* *addItemsToAdapter(items: List<Item>)* – метод, який додає список об’єктів *item* до адаптера, який відображає цей список на екрані у встановленому форматі;
* *initRecycler()* – метод, який створює візуальний елемент для відображення списку об’єктів *item* з вказаними налаштуваннями;
* *updateItems(items: List<Item>)* – оновлює список елементів *item*, які відображаються на еркані;
* *showEditItemDialog(item: Item) –* створює та відображає діалогове вікно, яке дозволяє створити новий елемент або змінити існуючий, якщо він вибраний*;*
* *updateBudget() –* метод, який оновлює інформацію про поточний бюджет на екрані;
* *showMessage(message: String) –* метод, який відповідає за вивід інформаційних повідомлень на екран*;*
* *init() –* метод, в якому відбувається початкова ініціалізація елементів екрана*;*
* *getBudgetCategory()* – метод, повертає поточну обрану категорію*;*
* *setBudgetCategory(budgetCategory: BudgetCategory)* – метод, який дозволяє встановити категорію як поточну.

Атрибути, які має клас:

* *tvTitle –* візуальний елемент *AppCompatTextView,* в якому відображається заголовок екрана;
* *recyclerView* – візуальний елемент *RecyclerView*, який відповідає за відображення списку елементів;
* *btnAddItem* – візуальний елемент *FloatingActionButton*, який відповідає за кнопку додавання нового елементу;
* *presenter* – екземпляр класу *BudgetItemContract.Presenter*;
* *adapter* – екземпляр класу *BudgetItemRecyclerAdapter*, який відповідає за формат відображення списку елементів на екрані;
* *budgetCategory* – екземпляр класу *BudgetCategory*.

Таким чином, можна побачити використання *MVP* шаблону та переваги від цього, а саме те, що між собою пов’язані лише інтерфейси, а не конкретні реалізації, що дозволяє за необхідності змінювати реалізацію того чи іншого компонента.

В цій роботі всі інші компоненти застосунку пов’язані між собою за таким самим принципом. Такий підхід дозволяє полегшити розуміння коду та структури програми, а відсутність монолітною системи дозволяє легко замінювати компоненти та розширяти систему у разі необхідності. Для передачі об’єктів між клієнтом та сервером використовується *JSON* формат.

## **5.2 Тестування взаємодії *serverless* та *serverfull* елементів**

Під час використання застосунку користувач має в середньому 6000 пунктів надходжень та витрат, які являються об’єктами з яких необхідно сформувати щомісячний звіт. На рисунку 5.3 зображується графік відповідності кількості таких об’єктів до часу їх обробки.

Як можна бачити з графіку при використанні *Azure Function* на обробку 9000 даних витрачається 1376 мс, а при використанні *serverfull* – 682 мс, що на 50.43% менше. Така різниця відбувається через те, що *Azure Function* потребує певний час для старту функції.

Припустимо, що застосунок буде використовуватися в середньому 150 000 користувачів одночасно, які мають 30 активних бюджетів, які повинні бути оброблені 4 рази в місяць, таким чином для їх обробки буде здійснюватися 18 000 000 викликів в місяць

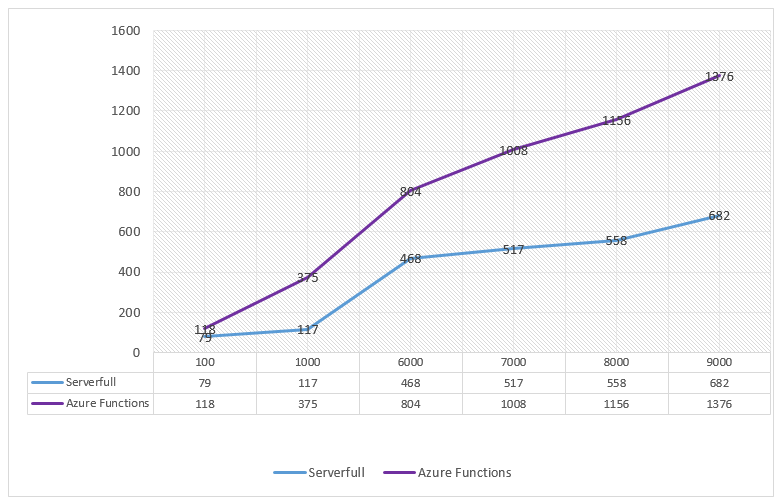


Рисунок 5.3 – Графік залежності часу обробки від кількості об’єктів

Оскільки, сервер окрім цих операцій повинен обробляти іншу інформацію від користувачів, тоді нам потрібен сервер з такими характеристиками: *RAM* 64*GB*, *vCPUs* 16*vCPUs*, *SSD* 200*GB*. Сервіс *digitalocean* пропонує такий сервер за щомісячну платню в $480.

Для того аби зменшити навантаження сервер, а відповідно і витрати на нього, делегуємо обробку щомісячного звіту на *serverless*-функції *Azure Function*. Таким чином, достатньо буде серверу з такими характеристиками: 32*GB*, vCPUs 8*vCPUs*, *SSD* 100*GB*. За даними *digitalocean* це коштує $240/місяць.

Для розрахунку звіту для одного бюджету користувача за тиждень нам необхідно *serverless* з *RAM* 512*MB*, яки буде викликатися 18000000 разів (рис. 5.4).

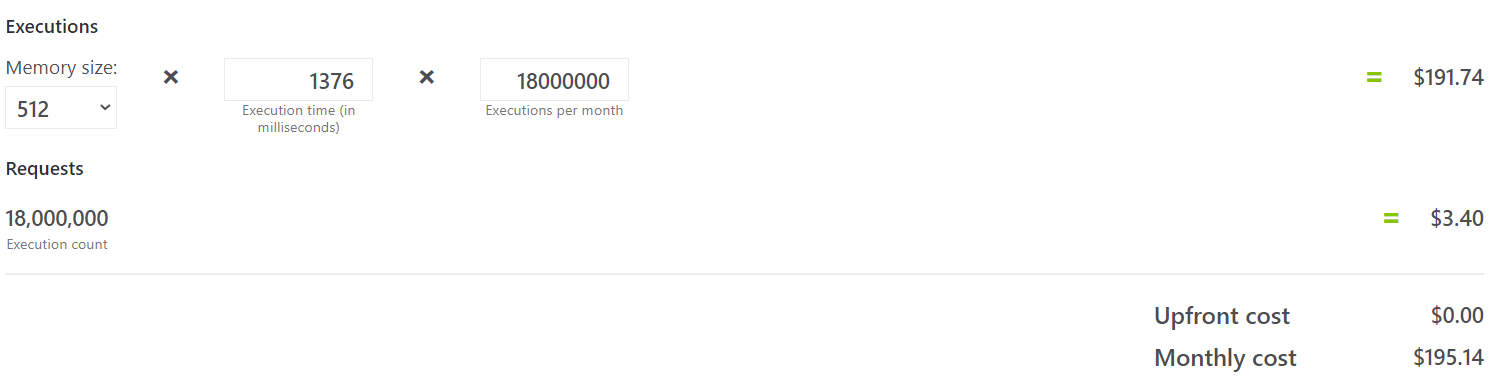


Рисунок 5.4 – Розрахунок вартості *Azure Function*

Згідно з розрахунками (рис. 5.4) така функція буде коштувати $195.14/місяць. Замість використання одного потужного серверу за $480/місяць можна використовувати сервер за $240/місяць та *serverless-*функцію за $195.14/місяць.

Таким чином, при використанні змішаної архітектури вартість використання серверу знижується з $480 до $435.14, що становить 9.34% економії.

**5.2 Тестування системи**

Тестування системи проводилося методом «чорного ящика» за допомогою тест-кейсів. Такий підхід дозволяє протестувати роботу системи та перевірити реакцію систему на дії які буде робити користувач під час користування програмним продуктом.

В таблиці 5.1 наводиться тест-кейс варіанту використання «реєстрація користувача».

Назва теста: Реєстрація нового користувача

Передумова: Користувач запустив застосунок

Таблиця 5.1 – Тест-кейс реєстрація нового користувача

|  |  |  |  |
| --- | --- | --- | --- |
| **Крок** | **Опис** | **Очікуваний результат** | **Статус** |
| Крок 1 | Запустити ярлик BudgetManager | Відкрилась форма авторизації | Успішно |
| Крок 2 | Натиснути кнопку «*Create new Account*» | Відкрилась форма реєстрації | Успішно |
| Крок 3 | В поле «*Username*» ввести «*Alex*» | *Username* введено | Успішно |
| Крок 4 | В поле «*Password*» ввести «nxb!U56wF$» | Пароль введено | Успішно |
| Крок 5 | В поле «*Confirm Password*»ввести «nxb!U56wF$» | Паролі співпадають | Успішно |
| Крок 6 | Натиснути кнопку «*Registration*» | Користувача зареєстровано. Відкрилась форма авторизації | Успішно |

В таблиці 5.2 наводиться тест-кейс варіанту використання «додавання нової позиції витрат».

Назва теста: Додавання нової позиції витрат

Передумова: Користувач зареєстрований в системі

Таблиця 5.2 – Тест-кейс додавання нової позиції витрат

|  |  |  |  |
| --- | --- | --- | --- |
| **Крок** | **Опис** | **Очікуваний результат** | **Статус** |
| Крок 1 | Запустити ярлик BudgetManager | Відкрилась форма авторизації | Успішно |
| Крок 2 | В поле «*Username*» ввести «*Alex*» | *Username* введено | Успішно |
| Крок 3 | В поле «*Password*» ввести «nxb!U56wF$» | Пароль введено | Успішно |
| Крок 4 | Натиснути кнопку «*Sign In*» | Відкрилась форма головного меню |  |
| Крок 5 | Обрати бюджет для додавання нової позиції витрат | Вивід списку категорій бюджету | Успішно |
| Крок 6 | Обрати категорію «*Food*» | Вивід списку з позиціями витрат обраної категорії | Успішно |
| Крок 7 | Натиснути кнопку «*Add*» | Відкрилась форма для вводу параметрів позиції витрат | Успішно |
| Крок 8 | У поле «*Price*» ввести «10», у поле «*Name*» ввести «*Icecream*» | Обрані поля заповнені. Поле «*Description*» залишилось пустим | Успішно |
| Крок 9 | Натиснути кнопку «*OK*» | Позиція витрат записана до обраної категорії. Дата створення позиції встановлюється на поточну дату. Поточний показник бюджету зменшується на 10 одиниць. | Успішно |

В таблиці 5.3 наводиться тест-кейс варіанту використання «перегляд звіту витрат за категоріями».

Назва теста: Перегляд звіту витрат за категоріями

Передумова: Користувач зареєстрований в системі

Таблиця 5.3 – Тест-кейс перегляд звіту витрат за категоріями

|  |  |  |  |
| --- | --- | --- | --- |
| **Крок** | **Опис** | **Очікуваний результат** | **Статус** |
| Крок 1 | Запустити ярлик BudgetManager | Відкрилась форма авторизації | Успішно |
| Крок 2 | В поле «*Username*» ввести «*Alex*» | *Username* введено | Успішно |
| Крок 3 | В поле «*Password*» ввести «nxb!U56wF$» | Пароль введено | Успішно |
| Крок 4 | Натиснути кнопку «*Sign In*» | Відкрилась форма головного меню | Успішно |
| Крок 5 | Обрати бюджет для перегляду звіту витрат | Вивід списку категорій бюджету | Успішно |
| Крок 6 | Натиснути кнопку «*Statistic*» | Відкрилась форма відображення звітів бюджету | Успішно |
| Крок 7 | Обрати пункти «*Categories*» та «*Spending*» | Відобразилась кругова діаграма, на якій зображуються назви категорій та значення витрат в цих категоріях | Успішно |

Отже, було розроблено та виконано три тест-кейса для тестування інтерфейсу користувача. Усі тести були успішно пройдено і це вказує на працездатність клієнтської частини та всієї системи загалом.

## **5.3 Інструкція користувача**

Для встановлення клієнтського застосунку, користувачу необхідно запустити файл *BudgetManager.apk* на своєму пристрої, після чого, запуск відбувається через ярлик *BudgetManager*.

Після запуску застосунку, відкривається форма авторизації (рис. 5.5), де користувачу пропонується або увійти в систему, або зареєструватися. Якщо під час авторизації відмітити пункт «*Remember me»*, тоді при повторному запуску застосунку, користувач одразу увійде в систему.

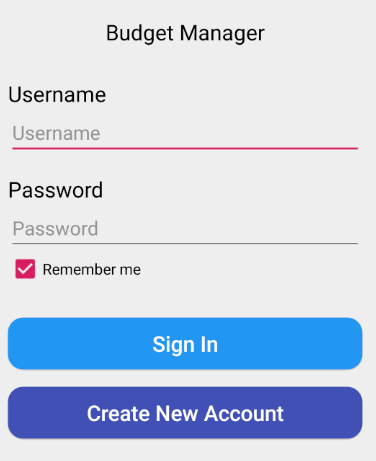


Рисунок 5.5 – Форма авторизації

Якщо на формі авторизацію натиснути кнопку «*Registration*», тоді відкриється форма реєстрації (рис. 5.6). Під час реєстрації користувачу запропонується одразу підключити до свого профіля інтерактивну колонку *Amazon Alexa*. Якщо при створені нового облікового запису обрати пункт «*Enable Alexa*», тоді після реєстрації користувач отримає код, який дозволить приєднати інтерактивну колонку *Amazon Alexa* до застосунку, інакше, це можна зробити пізніше в застосунку.

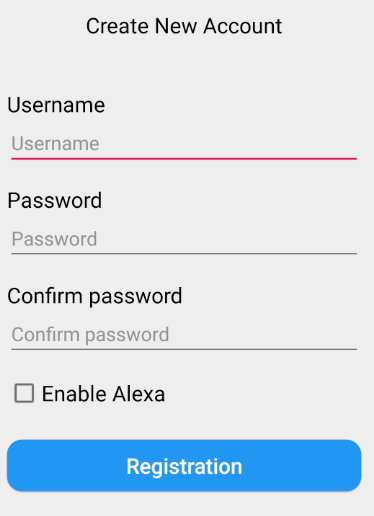


Рисунок 5.6 – Форма реєстрації

Після авторизації в застосунку та внесені інформації про бюджет, в ньому можна створювати категорії (рис. 5.7) бюджету, для збереження інформації про витрати. Для кожної категорії вказується її назва, опис, а також надається можливість переглянути вміст цієї категорії, додати нову категорію, або додати позицію витрат до категорії, не переглядаючи її вміст.

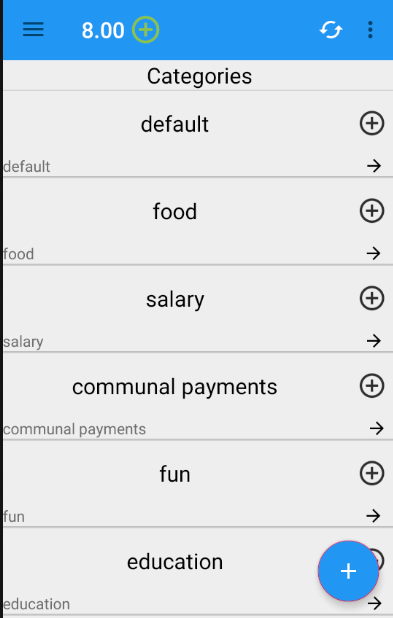


Рисунок 5.7 – Форма виведення списку категорій

При перегляді конкретної категорії (рис. 5.8), користувачу виводиться список з усіх її елементів, які мають назву, ціну, опис та дату створення.



Рисунок 5.8. – Форма відображення позицій витрат в категорії

Якщо користувач бажає переглянути звіт бюджету, то йому пропонується декілька варіантів звіту. Якщо користувач вибере параметри «*Categories*» та «*General*», тоді йому виведеться загальний звіт про витрати та надходження за категоріями (рис. 5.9).

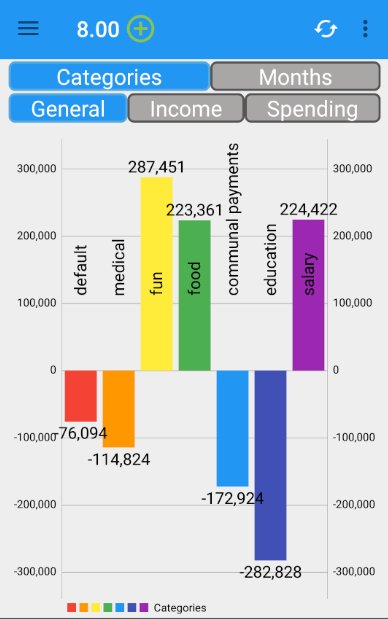


Рисунок 5.9 – Звіт витрат та доходів за категоріями бюджету

Якщо в формі отримання звіту обрати параметри «*Categories*» та «*Spending*», тоді користувач зможе побачити звіт витрат за категоріями (рис. 5.10). При виборі параметру «*Months*» буде виведена інформація про витрати за місяцями.

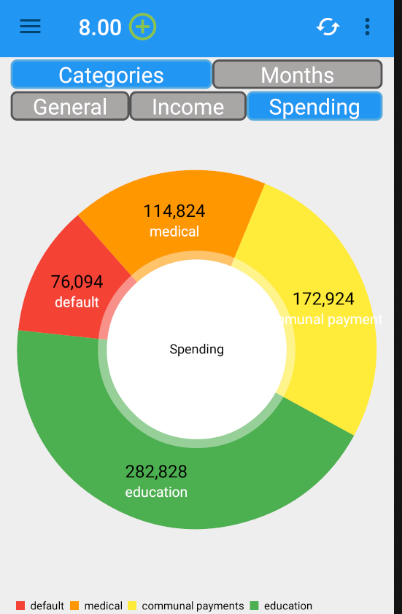


Рисунок 5.10 – Звіт витрат за категоріями

При встановленні параметрів «*Months*» та «*Income*», тоді користувач зможе побачити звіт надходжень за місяцями (5.11).

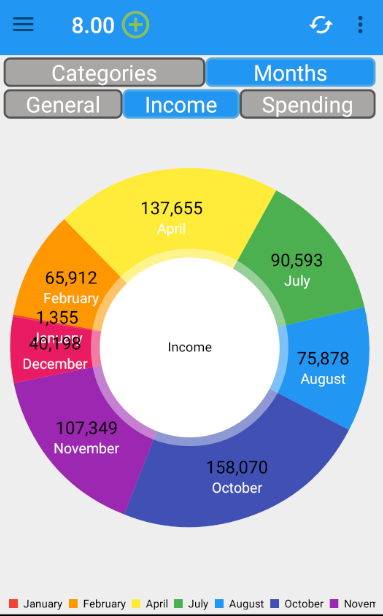


Рисунок 5.11 – Звіт доходів по місяцям

Таким чином, при різних комбінаціях параметрів «*Categories*», «*Months*» та «*General*», «*Income*», «*Spending*» можна отримати різні види звітів.

# **6. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ**

Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів і засобів, які спрямовані на збереження життя, здоров’я та працездатності співробітника в процесі трудової діяльності.

У дипломному проекту розглянуті питання розробки мобільного застосунку, вирішення яких здійснюється розробниками програмного забезпечення, а саме їх здійснює інженер-програміст.

Отже розглянемо питання охорони праці та безпеку у надзвичайних ситуаціях на робочому місці інженера-програміста, яке розташовано в компанії *R-Navigator* в офісній будівлі, на 6 поверсі.

Норми та заходи, які встановлюють та регулюють безпечні умови праці та поведінку під час виникнення надзвичайної ситуації викладені в наступних документах:

* НПАОП 0.00-4.12-05. Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці.
* ДБН В.2.5–28:2018. Природне і штучне освітлення.
* ДБН В.2.5-67:2013. Опалення, вентиляція та кондиціонування.
* ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень.
* ДСН 239-96. Державні санітарні норми і правила захисту населення від впливу електромагнітних випромінювань.
* ДНАОП 1.1.30-5.31-96. Інструкція з облаштування, огляду і вимірювання опору штатних заземлень.
* НАПБ Б.03.002-2007. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою.

Вивчення найважливіших заходів з охорони праці та безпеки у надзвичайних ситуаціях для працівників галузі інженерія програмного забезпечення наведено у Додатку Б та висвітлює наступні питання:

* організація та управління охороною праці на підприємстві
* обґрунтування заходів з покращення умов охорони праці
* розрахунок освітлення робочої зони
* надзвичайні ситуації та шляхи їх запобігання

Проведено аналіз умов праці на робочому місці програміста, а саме освітлення робочого місця, повітря робочої зони, мікрокліматичні умови, іонізуюче випромінювання, електробезпека та пожежна безпека. Було розраховано система освітлення робочого місця інженера-програміста, визначено дози опромінення та часу перебування людей на радіоактивно зараженій місцевості при встановленій дозі опромінення та запропоновано наступні заходи з охорони праці та безпеки в надзвичайних ситуаціях:

- Збільшення контрастності між фоном та об’єктом розрізнення,

- Збільшення розміру об’єкта розрізнення,

- Розробка заходів з евакуації персоналу у разі виникнення загрози радіоактивного характеру.

# **ВИСНОВКИ**

В цій роботі було розроблено клієнтську частину для системи планування бюджету. В результаті виконання дипломної роботи були виконані наступні задачі:

1. Проаналізовано *serverless* та *serverfull* архітектури.
2. Спроектувано архітектуру програмної системи.
3. Проведено тестування  архітектури програмної системи та виявлено, що при поєднанні *serverless* архітектури у вигляді *Azure Functions* з серверною архітектурю, економія на підтримку системи становить 9.34%. Програш у часі при використанні *serverfull* архітектури становить 50.43%.
4. Розроблено клієнтську частину для системи планування бюджету. Клієнтська частина являє собою *Andriod* застосунок, який дозволяє користувачу виконувати такі дії як:

- створення та керування обліковим записом користувача;

- створення та керування бюджетом;

- створення та керування категоріями бюджету;

- додавання витрат бюджету;

- керування групами користувачів;

- синхронізація системи з інтерактивною колонкою *Amazon Alexa*;

- створення звіту витрат користувачів.

В майбутньому ця програмна система може бути розширена можливістю взаємодіяти з банківською системою. Клієнтська частина може бути перенесена у веб-браузер, що дозволить збільшити кількість платформ та операційних систем, на яких буде працювати ця система.

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Brandom R.: There are now 2.5 billion active Android devices. [Електронний ресурс]. – Режим доступу: URL:

<https://www.theverge.com/2019/5/7/18528297/google-io-2019-android-devices-play-store-total-number-statistic-keynote> – назва з екрану.

1. Statcounter GlobalStats. [Електронний ресурс]. – Режим доступу: URL: <https://gs.statcounter.com/os-market-share/mobile/worldwide> – назва з екрану.
2. Spillner J., Mateos C., Monge D.: FaaSter, Better, Cheaper: The Prospect of Serverless Scientific Computing and HPC.
3. Jamieson, Frazer (4 September 2017). "Losing the server? Everybody is talking about serverless architecture". [Електронний ресурс]. – Режим доступу: URL: <https://www.bcs.org/content-hub/losing-the-server/> – назва з екрану.
4. Miller, Ron. "AWS Lambda Makes Serverless Applications A Reality". TechCrunch. [Електронний ресурс]. – Режим доступу: URL: <https://techcrunch.com/2015/11/24/aws-lamda-makes-serverless-applications-a-reality/> – назва з екрану.
5. Miller, Ron. "Microsoft answers AWS Lambda's event-triggered serverless apps with Azure Functions". TechCrunch. [Електронний ресурс]. – Режим доступу: URL: <https://techcrunch.com/2016/03/31/microsoft-answers-aws-lambdas-event-triggered-serverless-apps-with-azure-functions/> – назва з екрану.
6. MSV, Janakiram. "PaaS Vendors, Watch Out! Amazon Is All Set To Disrupt the Market". [Електронний ресурс]. – Режим доступу: URL: <https://www.forbes.com/sites/janakirammsv/2015/07/16/paas-vendors-watch-out-amazon-is-all-set-to-disrupt-the-market/#332a405fe88d> – назва з екрану.
7. The 12 Most Critical Risks for Serverless Applications. [Електронний ресурс]. – Режим доступу: URL: <https://www.puresec.io/serverless-security-top-12-csa-puresec> – назва з екрану.
8. Solow, Hillel. "Serverless Computing Security Risks & Challenges". protego.io. [Електронний ресурс]. – Режим доступу: URL: <https://www.checkpoint.com/products/cloudguard-serverless-security/> – назва з екрану.
9. Bashir, Faizan. "What is Serverless Architecture? What are its Pros and Cons?". Hacker Noon. [Електронний ресурс]. – Режим доступу: URL: <https://hackernoon.com/what-is-serverless-architecture-what-are-its-pros-and-cons-cc4b804022e9> – назва з екрану.
10. "What Is Serverless? Here's a Plain Answer!". Squadex. [Електронний ресурс]. – Режим доступу: URL: <https://squadex.com/insights/what-is-serverless/> – назва з екрану.
11. DigitalOcean. [Електронний ресурс]. – Режим доступу: URL: <https://www.digitalocean.com/pricing/> – назва з екрану.
12. Михалюк М. С., Шевчук І. О. Тестування та аналіз результатів використання безсервеної та класичної архітектур в інформаційній системі для планування бюджету. // Theory, science and practice. Abstracts of III International Scientific and Practical Conference. Tokio, Japan 2020. – pp. 104–106.
13. Писаренко К. О., Оніщенко Т. В., Шевчук І. О. Алгоритм выбора метода для прогнозирования расходов пользователя. // Abstracts of VIII International Scientific and Practical Conference. Prague, Czech Republic 2020. - pp. 239 - 241.