**Національний технічний університет України  
«Київський політехнічний інститут»**

Інститут Прикладного системного аналізу  
Кафедра Системного проектування

Лабораторна робота №1

з дисципліни «Технологія розробки програмного забезпечення»

**«Продаж ЗД квитків. Архітектура та структура додатку.**

**EasyRail App»**

Виконали:

студенти групи ДА-82

факультету «ІПСА»

Чайковський В.В.

Дурда Р.Є

Гапонюк М.О.

Харитонова Д.С.

Київ – 2020

**Вступ**

Життя кожної людини з кожним роком еволюціонує. З’являються новітні технології, які роблять наше існування ще зручнішим та кращим. Майже кожен з нас має мобільний телефон і вже давно з’явилась тенденція про надання послуг через нього, оскільки це зручно і швидко.

Багато сервісів, які належать державі, мають не найкращі мобільні аналоги, оскільки розробка ПЗ на замовлення держави мають високі стандарти, а також велику кількість «нудної» роботи з документами, якої так не люблять розробники.

Одним з таких є сервіси купівлі квитків.

В наш час важко уявити існування людини без її повсякденного пересування. І ми волею чи не волею повинні звертатися до послуг, що нам надає УКРЗАЛІЗНИЦЯ. Проте, у сучасності, люди позбавлені вільного часу на вирішення проблем що пов’язані з замовленням або поверненням квитків. Тому виникає бажання отримати зручну систему, яка дозволить уникнути цих рутинних дій та зекономити і час, і нерви користувача.

**Мета** **роботи**

Метою цього проекту є розробка зручної системи для бронювання та купівлі квитків на потяги, яке має весь життєво необхідний функціонал і може задовільнити рядового користувача так само, як і адміністраторів.

**Ролі**

В нашій команді є чотири особи. Для ефективної роботи над нашим проектом, ми розподілили ролі наступним чином:

* Team lead – має наступні обов’язки:
  + Планування
  + Контроль над виконанням проекту
  + Робота з колективом
* Designer – відповідає за ідейну розробку користувацького інтерфейсу та його зручність.
* Backend – відповідає за бізнес логіку продукту, збереження та організацію даних, взаємодію з користувацьким інтерфейсом.
* Frontend – несе відповідальність за створення функціональної реалізації інтерфейсу продукту. Має наступні обов’язки:
  + Кодування візуальних взаємодій
  + Відтворення ідей дизайнера

Чайковський В.В. (Team lead, mobile frontend + backend)

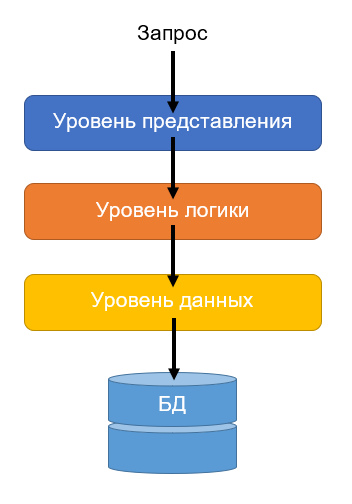
Харитонова Д.С. (Designer, desktop designer + frontend)

Дурда Р.Є. (Backend, mobile frontend + backend)

Гапонюк М.О. (Frontend, desktop designer + backend)

**Архітектура**

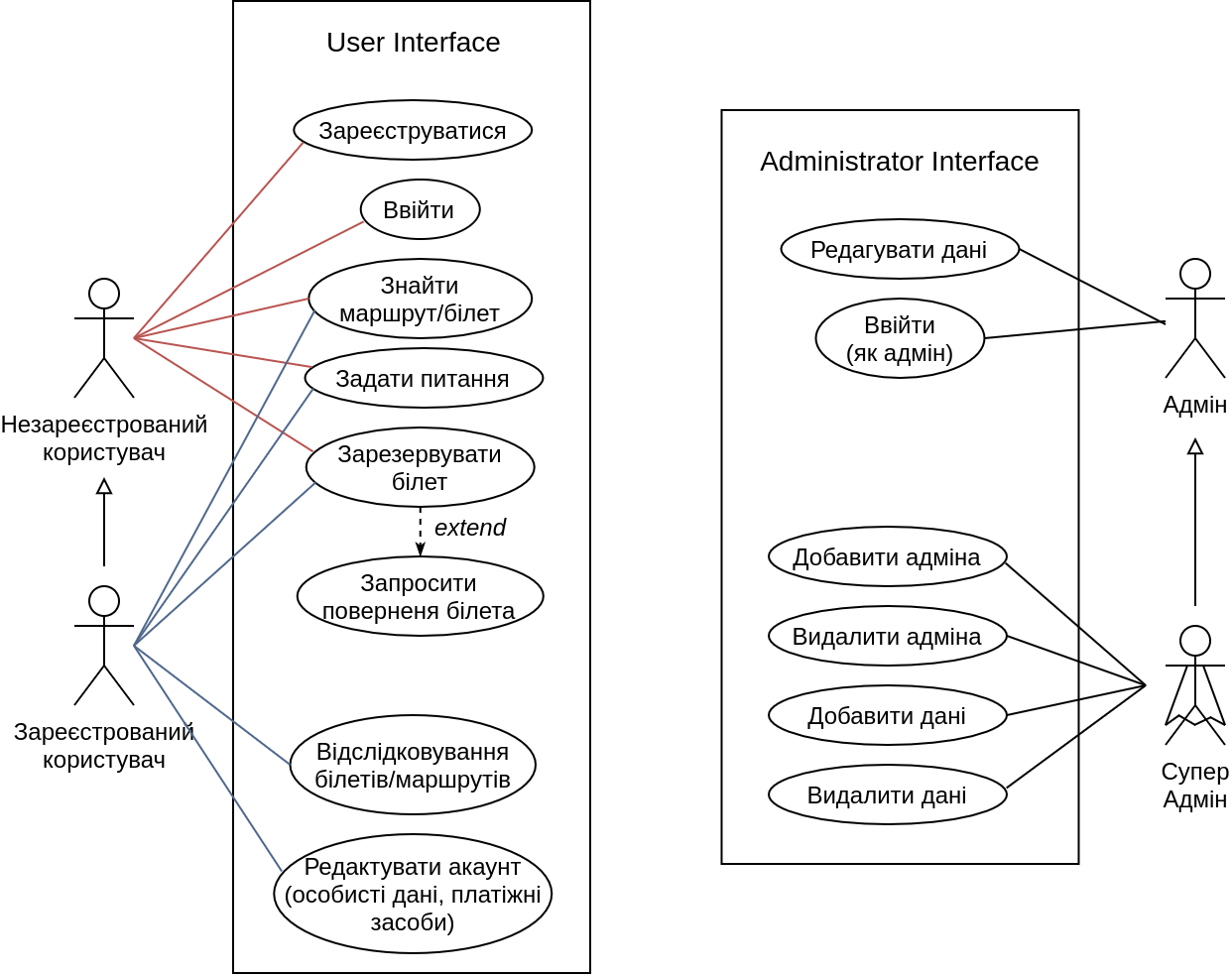
Оскільки ми маємо досить великі обмеження у часі, було вирішено взяти за модель архітектури нашої системи класичну трьохрівневу модель.

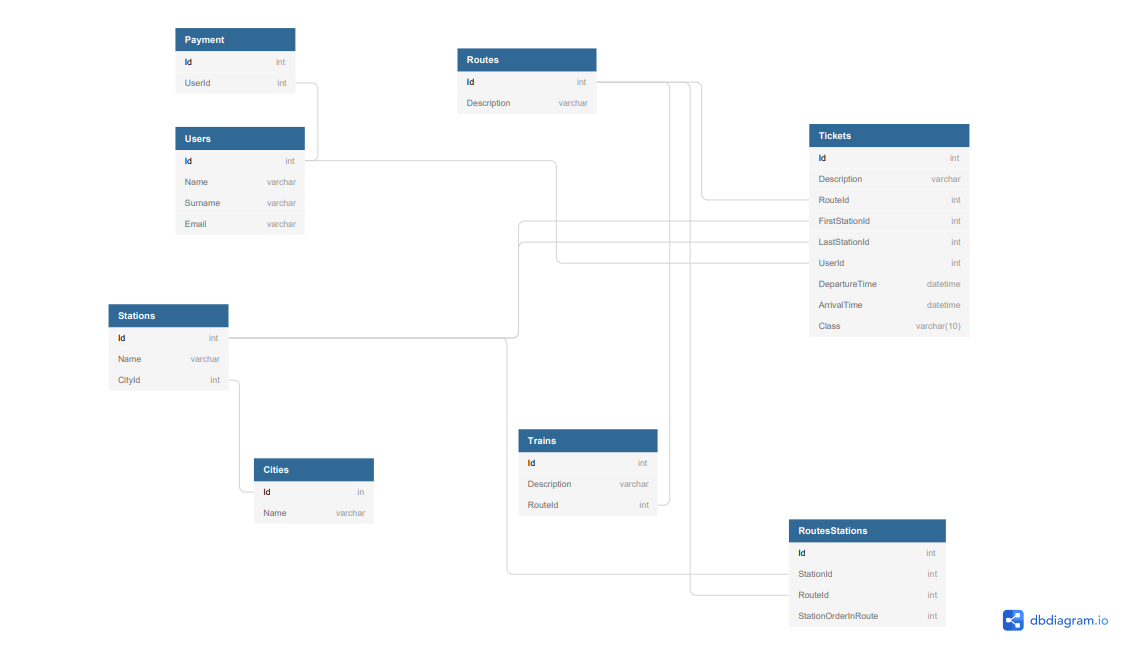
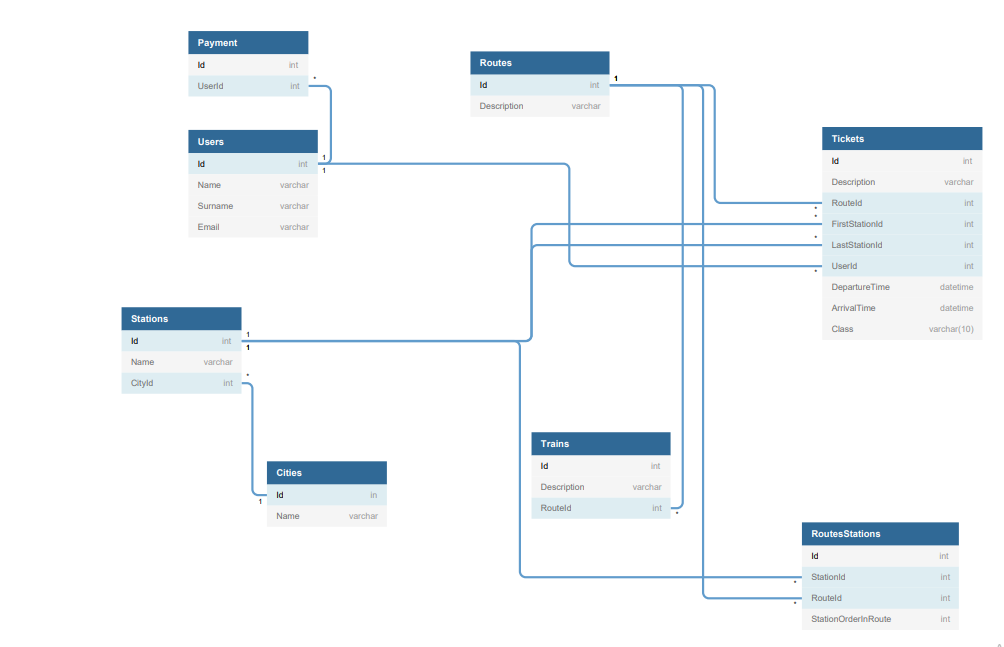


Архітектура була обрана з наступних причин:

1. Кожен рівень архітектури не залежить від сусідніх рівнів. Ця особливість дозволяє зручно розподілити обов’язки кожного з розробників, а також є досить зручним у розширенні та супроводженні.
2. Модель є простою. Таким чином економиться час на аналіз більш складніших архітектур.
3. Трьохрівнева архітектура підходить для розробки додатків, які є аналогічні нашому.

**Use-case діаграма**



**Структура БД**

**Методологія розробки ПЗ**

Було вибрано фреймворк з сімейства методології гнучких фрейморків Agile, а саме Scrum.

Чому саме scrum?

Scrum, як і було сказано вище, належить до сімейства гнучких фрейморків. Саме можливість зміни вектору розробки, корегування планів «on the fly», зробила наш вибір очевидним. Також, scrum добре лягає поверх невеликої команди та одного проекту.

Але наш scrum буде не класичним, оскільки в нас немає замовника. Замовником у нашому випадку є сама команда. Роль scrum-мастера та PO виконується безпосередньо в середині групи розробників.

**Правила мови Java/C++/Python**

1. Виключення: ніколи не захоплюємо та не ігноруємо виключень без пояснення.
   1. Перехоплювати виключення відповідно до рівня абстракції
   2. Якщо є впевненість у тому, що ігнорування виключення має місце – прокоментуйте, чому саме було так рішено.
2. Виключення: не використовувати узагальнені виключення, окрім коду у бібліотеках, у корені стеку.
3. Імпорти: повністю уточнювати імпорти.

**Code Style**

1. Короткі методи, загальні потреби до написання методів
   1. Методи повинні бути невеликими та вирішувати конкретну задачу.
   2. Намагатися розбити великий метод на більш менші.
   3. Метод не повинен змінювати глобальні змінні.
   4. Метод не повинен мати більше ніж 7 параметрів
   5. Метод повинен бути захищений від поганих даних, які можуть порушити його роботу (приклад, передано параметром 0, на яке відбувається ділення)
2. Назви методів – назва методу повинна чітко відображати його сутність
   1. Парні антоніми для методів, які виконують парні дії: open/close, show/hide, add/remove і т.д.
   2. Назва методу повинна бути написана у CamelCas’і, за виключенням першого слова.
   3. Методи, які повертають булеве значення повинні мати назву типу «isClosed()», «isEmpty()» і т.п.
3. Поля: повинні бути зверху файлу, або прямо перед методом, який їх використовує.
4. Локальні змінні: обмежувати область видимості.
5. Відступи: 4 пробіли, без табуляцій.
6. Довжина строки: не більше 100 символів.
7. Назви полів: не public та не static поля починаються з «m»
   1. Не public та не static – починаються з «m»
   2. Static поля починаються з «s»
   3. Інші поля починаються з букви нижнього регістру
   4. Поля константи (const, public static final) пишуться повністю у верхньому регістрі, з використанням підкреслення (ALL\_CAPS\_WITH\_UNDERSCORES)

Приклад:  
public class MyClass {

public static final int SOME\_CONSTANT = 42;

public int publicField;

private static MyClass sSingleton;

int mPackagePrivate;

private int mPrivate;

protected int mProtected;

}

1. Фігурні скобки: відкриваючі фігурні скобки не знаходяться на окремій строчці.
2. Анотації: користування лише стандартних анотацій.
3. Скорочення: використання скорочень як слова в іменах, наприклад XmlHttpRequest, getUrl і т.п.
4. Стиль TODO: «TODO: опис\_тут»
5. Загальні правила до неймінгу змінних:
   1. Уникати абревіатур та неочевидних скорочень.
   2. Уникати імен з близькими значеннями (у рамках конкретної області видимості, приклад, метод)
   3. Уникати змінних, які не мають об’єктивного значення