МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

Кафедра комп’ютерних наук

КУРСОВА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

з дисципліни Бази даних

на тему:

«Розробка туристичного вебсайту»

студента ІІ курсу групи КН-21-1
спеціальності 122 «Комп’ютерні науки»
Баковецького Максима Олександровича
(прізвище, ім’я та по-батькові)

Керівник Сугоняк І.І.

Дата захисту: " 10 " червня 2023р.
Національна шкала _____
Кількість балів: _____
Оцінка: ECTS _____

Члени комісії _____ Сугоняк І.І.
(підпис) (прізвище та ініціали)
_____ О.В. Коротун
(підпис) (прізвище та ініціали)
_____ О.Г. Чижмотря
(підпис) (прізвище та ініціали)

Житомир – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерних наук
Освітній рівень: бакалавр
Спеціальність 122 «Комп'ютерні науки»

«ЗАТВЕРДЖУЮ»
Зав. кафедри
_____ Марина ГРАФ
"10"___червня 2023 р.

ЗАВДАННЯ
НА КУРСОВУ РОБОТУ СТУДЕНТУ
Баковецькому Максиму Олександровичу

1. Тема роботи: Розробка туристичного вебсайту,
   керівник курсового проекту: Сугоняк І.І.
2. Строк подання студентом: "__10__" __червня__ 2023р.
3. Вихідні дані до роботи: Розробка туристичного вебсайту.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)
   1. Постановка завдання
   2. Аналіз аналогічних розробок
   3. Алгоритми роботи програми
   4. Опис роботи програми
   5. Програмне дослідження

5. Перелік графічного матеріалу(з точним зазначенням обов'язкових креслень)
   1.Презентація
   2.Посилання на репозиторій: https://gitlab.com/kn211_bmo/tourscape
6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посади консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3	Сугоняк І.І.	01.09.2022	10.01.2023

7. Дата видачі завдання "07" __лютого____ 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів курсової роботи	Строк виконання етапів проекту	Примітки
1	Постановка задачі	07.02.2023	Виконано
2	Пошук, огляд та аналіз аналогічних розробок	12.02.2023	Виконано
3	Формулювання технічного завдання	13.02.2023	Виконано
4	Опрацювання літературних джерел	22.02.2023	Виконано
5	Проектування структури	27.02.2023	Виконано
6	Написання програмного коду	09.03.2023	Виконано
7	Відлагодження	18.05.2023	Виконано
8	Написання пояснювальної записки	31.05.2023	Виконано
9	Захист	02.06.2023	Виконано

Студент _____ Баковецький М.О.
(підпис)
(прізвище та ініціали)

Керівник проекту _____ Сугоняк І.І.
(підпис)
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до курсового проекта на тему «Розробка туристичного веб-сайту» складається з переліку умовних скорочень, вступу, чотирьох розділів, висновків, списку використаної літератури та додатку. Текстова частина викладена на 37 сторінках друкованого тексту. Пояснювальна записка має 44 сторінок додатків. Список використаних джерел займає 1 сторінку. В роботі наведено 25 рисунків. Загальний обсяг роботи – 80 сторінки. У першому розділі було проведено теоретичний аналіз інформаційних потоків та особливостей предметної області дослідження. У другому розділі було проведено проектування та розробка програмного забезпечення. У третьому розділі було проведено опис роботи з програмним додатком та його реалізування. У четвертому розділі налаштовано параметри адміністрування бази даних. Висновок містить в собі результати виконаної роботи при створенні бази даних та веб-додатку на тему «Розробка туристчиного веб-сайту». У додатку представлені технічне завдання та лістинг розробленого програмного продукту. Ключові слова: MongoDB, NodeJS, Mongoose, БД, JavaScript, веб-додаток, MVC.

ЗМІСТ

ВСТУП

Актуальність теми. Туризм став надзвичайно популярним серед масової аудиторії. Люди все більше цінують можливість подорожувати, відкривати нові країни, культури та досвіджувати пригоди. Туристичний веб-сайт може стати важливим джерелом інформації та планування для таких осіб.

Метою курсової роботи є дослідження особливостей проектування та реалізації баз даних за визначеним темою курсової роботи напрямком.

Завданням на курсову роботу є:

− аналіз теоретичних засад проектування та реалізації систем на основі баз даних;

− визначення інформаційних потреб предметної області дослідження;

− аналіз напрямку ризиків інформаційних потоків та їх структури;

− проектування бази даних за визначеною предметною областю;

− розробка математичної та алгоритмічної моделі функціонування системи на основі БД;

− реалізація БД та інтерфейсних засобів інформаційної системи **Об’єктом дослідження є** методи та засоби проектування баз даних за визначеними предметними областями.

Предметом дослідження є можливості застосування концепції БД та СУБД для забезпечення інформаційних потреб предметної області.

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТОКІВ ТА ОСОБЛИВОСТЕЙ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1 Аналіз інформаційних потреб та визначення предметної області достідження

Метою курсового проекту є створення бази даних і веб-додатку для керування та обробки даних у цій базі. Основні завдання включають:

1. Вибір необхідної інформації для функціонування веб-сайту.
2. Визначення потреб звичайних користувачів.
3. Створення простого та зрозумілого інтерфейсу для користувачів.
4. Вибір програмного забезпечення та мови програмування для веб-сайту та бази даних.
5. Створення бази даних та таблиць для зберігання інформації.
6. Реалізація ролей для користувачів з відповідними правами доступу.
7. Налаштування адміністрування веб-сайту.
8. Розробка функціоналу пошуку даних.
9. Відображення вибраної статистики.

Було вирішено створити веб-додаток з використанням патерну MVC (Model View Controller). В ролі бази даних – MongoDB. Адміністрування відбувається за допомогою MongoDBCompass та Atlas.

Обґрунтування вибору засобів реалізації

Порівняємо бази даних MongoDB, MSSQL, MySQL за різними параметрами.

1. Модель даних:

- MongoDB: Є нереляційною, яка використовує документи у форматі JSON для зберігання даних. Дані організовані у колекції документів.
- MSSQL: Реляційна БД, використовує табличну модель для зберігання даних. Дані організовані у вигляді таблиць та зв’язків між ними.
- MySQL: Реляційна БД, дані організовані так само як і у MSSQL.

2. Масштабованість:
    - MongoDB: Має вбудовану підтримку горизонтального масштабування, що дозволяє розширювати кластери бази даних за допомогою реплікацій та шардування.
    - MSSQL: Підтримує вертикальне масштабування, що означає, збільшення обсягу обробки на одному сервері, шляхом покращення апаратного забезпечення або налаштування бази даних.
    - MySQL: Підтримує вертикальне масштабування, але для горизонтального потрібні додаткові рішення.
3. Мови запитів:
    - MongoDB: Використовується мова запитів MongoDB Query Language, яка є потужним і гнучким інструментом для роботи із документами у базі даних.
    - MSSQL: Використовується мова запитів Transact-SQL (T-SQL), яка є стандартною мовою для роботи з реляційними базами даних Microsoft SQL Server.
    - MySQL: Використовується мова запитів SQL, яка є стандартною мовою для роботи з реляційними базами даних.

Отже, після проведеного аналізу було вирішено обрати саме MongoDB. Дана база даних зберігає дані у форматі JSON-подібних об’єктів, що дозволяє легко зберігати структуровані та невстановлені дані. Також MongoDB має швидкодія запису та читання даних завдяки використанню протоколу обміну даними у форматі BSON. Дана база даних є простою і зручною, також існує велика кількість фреймворків та бібліотек, що дозволяє легко та швидко розробляти додатки.

1.2. Аналіз існуючого програмного забезпечення за тематикою курсового проекту

У ході пошуку аналогів було знайдено такі застосунки:

1. Viator

Сайт доступний за посиланням: https://www.viator.com/

Зовнішній вигляд:

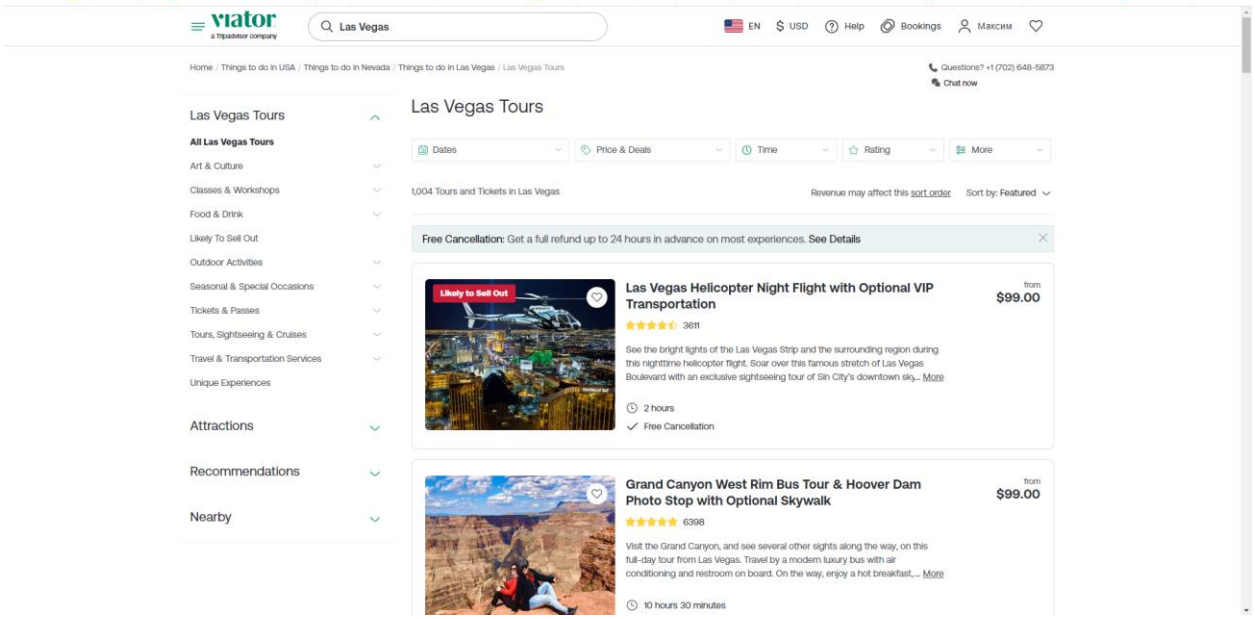

Рис. 1.1 Сайт viator

Переваги:

1. Великий функціонал
2. Приємний інтерфейс

Недоліки:

1. Складнуватий інтерфейс для новачків

2. GetYourGuide

Сайт доступний за посиланням: https://www.getyourguide.com/

Зовнішній вигляд:

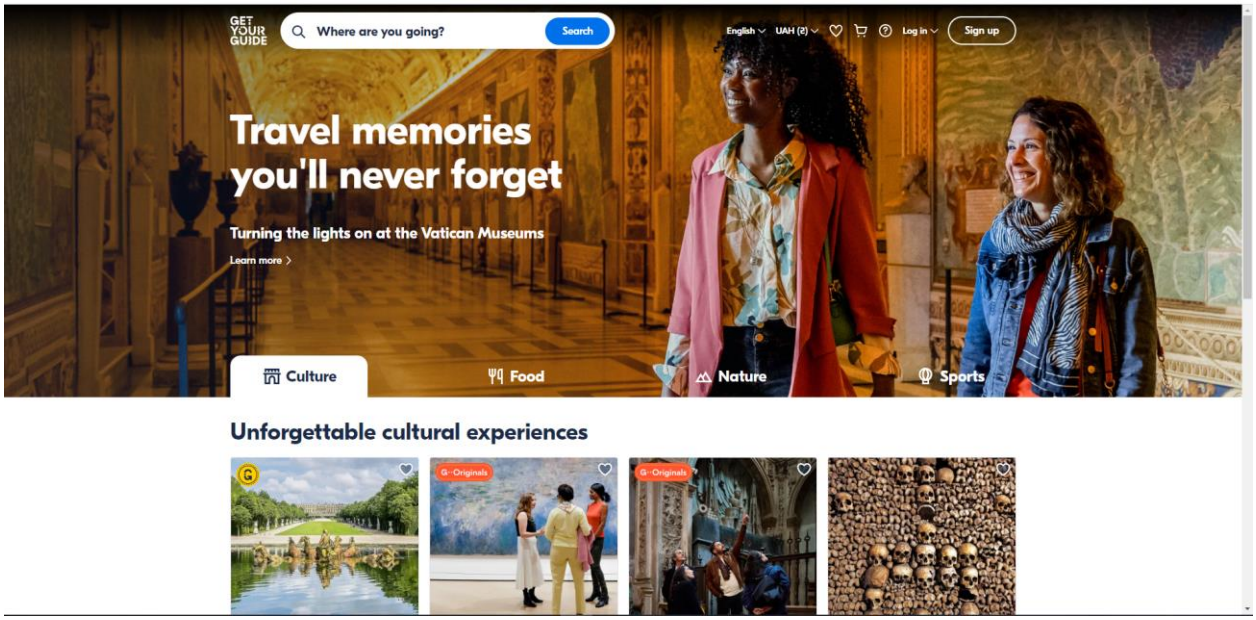

Рис. 1.2 Сайт getyourguide

Переваги:

1. Красивий динамічний інтерфейс
2. Простий у користуванні

Недоліки:

1. Невеликий вибір

Висновки до першого розділу

У процесі виконання першого розділу було встановлено, які можливості доступні користувачам і яка зона функціональності є доступною для роботи. Також була визначена можливість доступу до різних рівнів конфіденційності інформації. Було встановлено необхідні інструменти та засоби для реалізації веб-додатку, а також встановлено вимоги до створення інтерфейсу користувача. Було досліджено аналоги задля визначення вектору ходу розробки програмного додатку.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Проектування загального алгоритму роботи програми

Переходимо до проектування загального алгоритму роботи додатку. На сайті присутня авторизація. При реєстрації, користувач має роль ‘user’, тобто звичайний користувач. Розглянемо інші ролі веб-додатку.

- Admin
- Lead-guide
- Guide
- User
- Неавторизований користувач

Розглянемо можливості адміністратора:

- Додавання, редагування, видалення усього контенту
- Панель адміністратора, де адмін може змінювати ролі користувачам, видаляти їх, а також переглядати загальну статистику по сайту
- Видаляти відгуки користувачів

Що адмін не може робити:

- Видалити або редагувати себе
- Змінювати будь які дані про користувачів, окрім ролі
- Зробити замовлення або залишити відгук(ці функції має лише звичайний користувач)
- Редагувати відгуки

Щодо можливостей lead-guide, то у нього також є доступ до панелі адміна, але переглядати там він може лише статистику сайту. Якщо говоримо про guide, то він має права звичайного користувача, але не може робити

замовлення та залишити відгуки. Тепер щодо звичайного авторизованого користувача:

- може робити замовлення, залишати відгуки
- редагувати дані про себе, видалити свій аккаунт

Неавторизований користувач може лише переглядати сторінки

2.2 Розробка функціональних алгоритмів роботи програми

Розглянемо алгоритм роботу для аутентифікованого користувача:

1. Авторизація або реєстрація, перенаправлення на головну сторінку.
2. Перегляд свого профілю, його редагування або видалення
3. Перегляд усіх пунктів призначень (міст) або окремих міст
4. Перегляд, фільтрація усіх турів
5. Перегляд окремого туру
6. Замовлення туру, написання відгуку
7. Вихід із аккаунту

Загальна схема роботи:

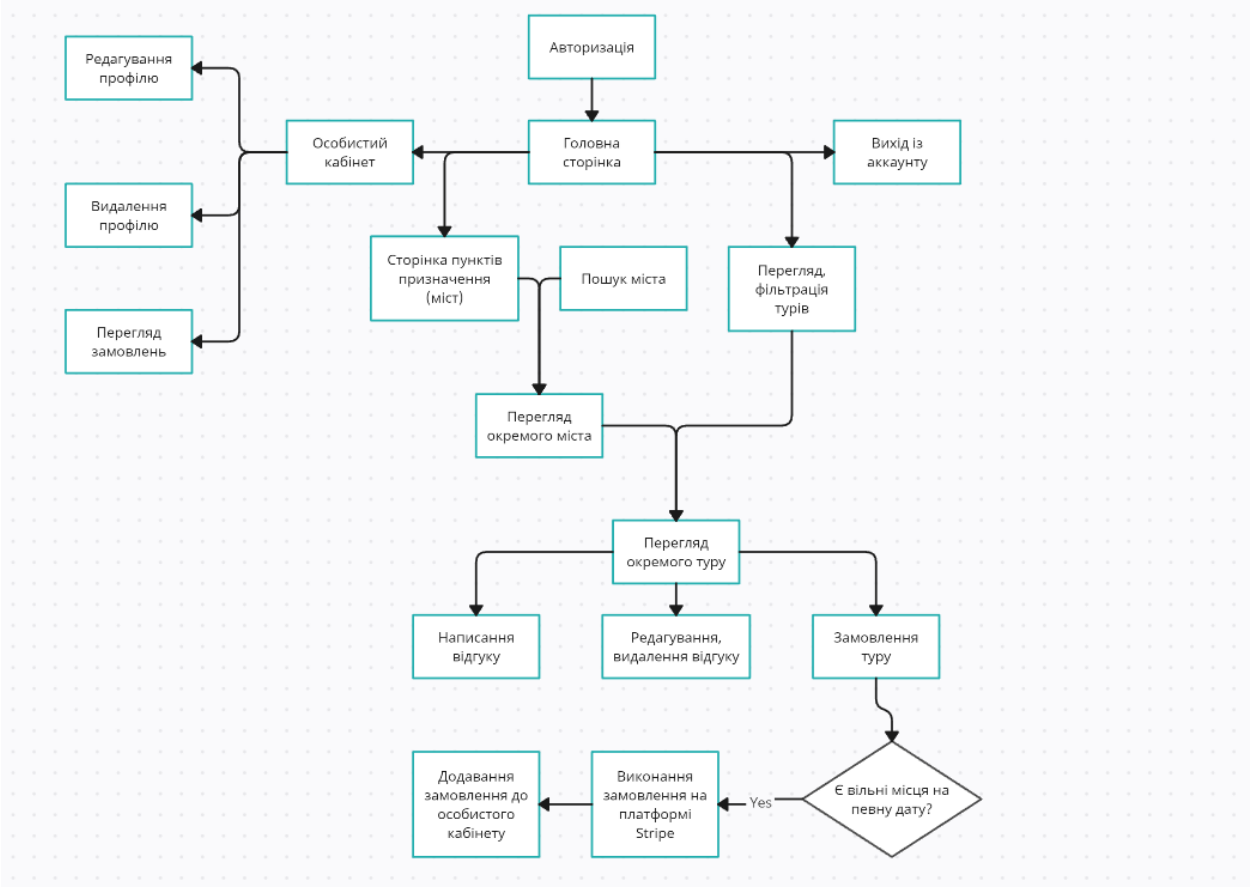

Рис. 2.2.1 Схема алгоритму роботи додатку для авторизованого користувача

Розглянемо алгоритм роботи додатку для адміністратора:

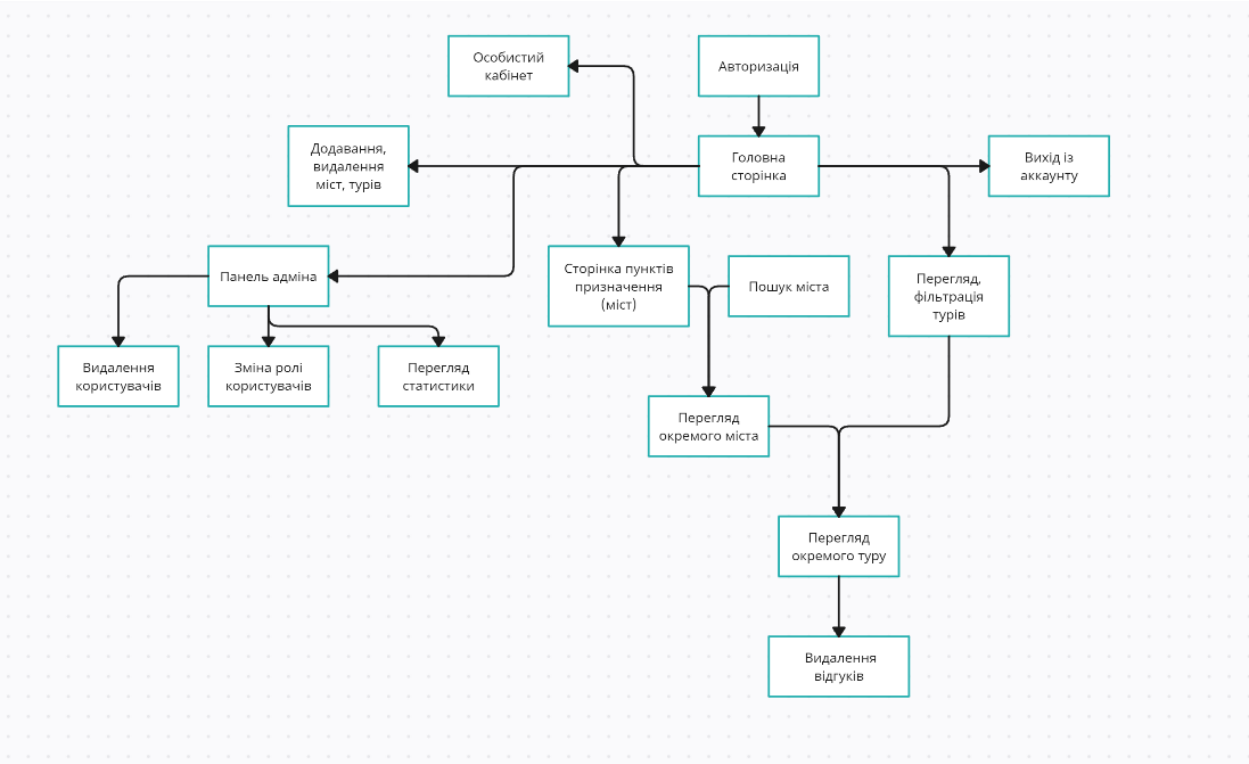

Рис. 2.2.2 Схема алгоритму роботи додатку для адміністратора

Тепер розглянемо схему процесу авторизації та реєстрації:

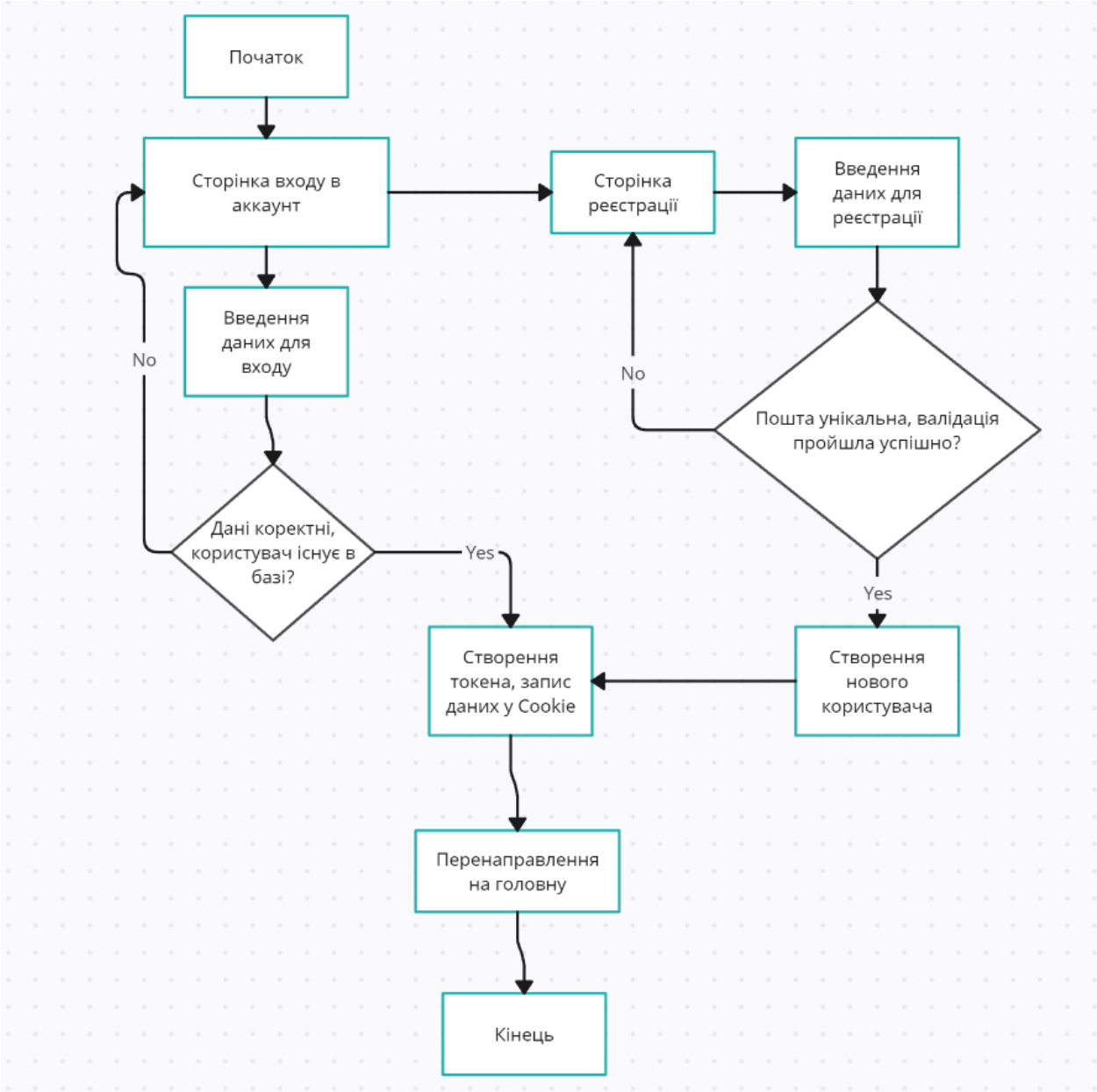

Рис. 2.2.3 Схема алгоритму авторизації та реєстрації

2.3 Розробка веб-додатку

Після завершення проектування алгоритмів додатку було розпочато розробку програмного додатку Було створено проект, який було розділено на дві частини – backend та frontend. Оскільки при розробці використовувався паттерн MVC, у папці із бекендом створюємо такі папки: controllers, models, routes. У папці controllers будуть міститися усі контроллери для кожної моделі бази даних. У папці models – схеми колекцій та моделі. Розглянемо файлову структуру проекту:

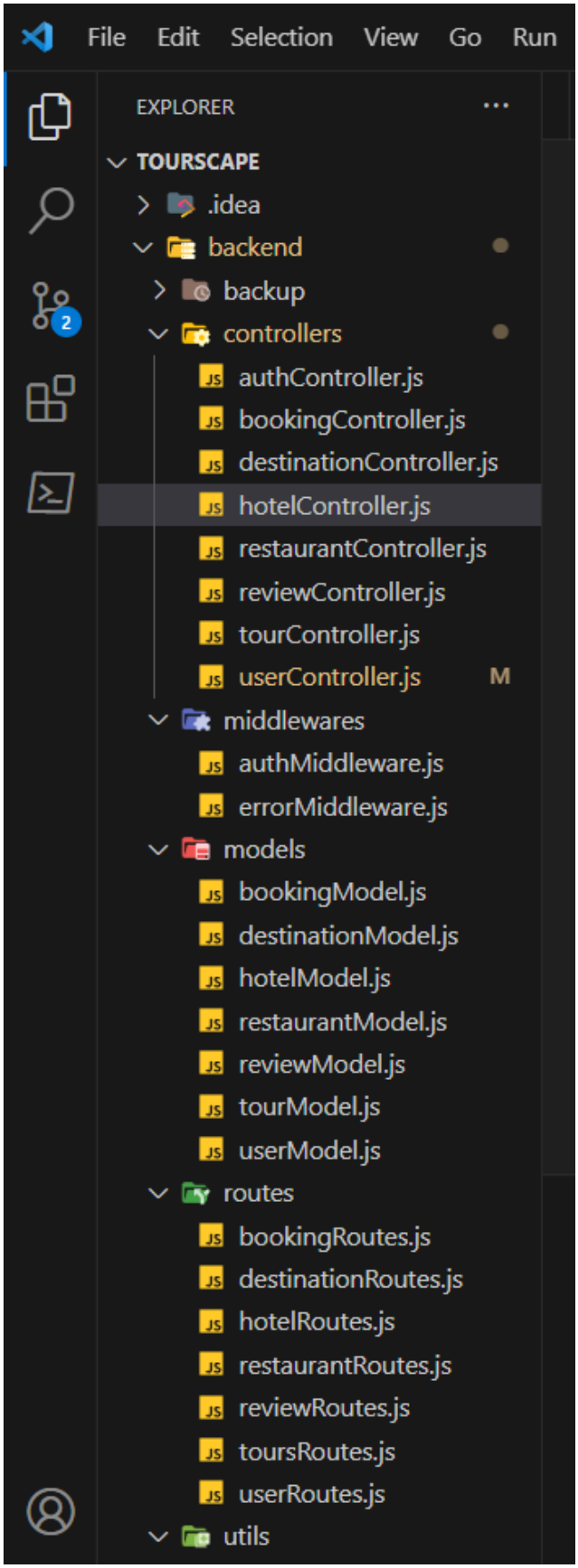

Рис 2.3.1 Структура каталогу backend (неповна)

Контроллери, моделі, та роути готелів та ресторанів можна ігнорувати, оскільки вони не використовувалися у курсовій роботі, а просто створені для особистого покращення проекту.

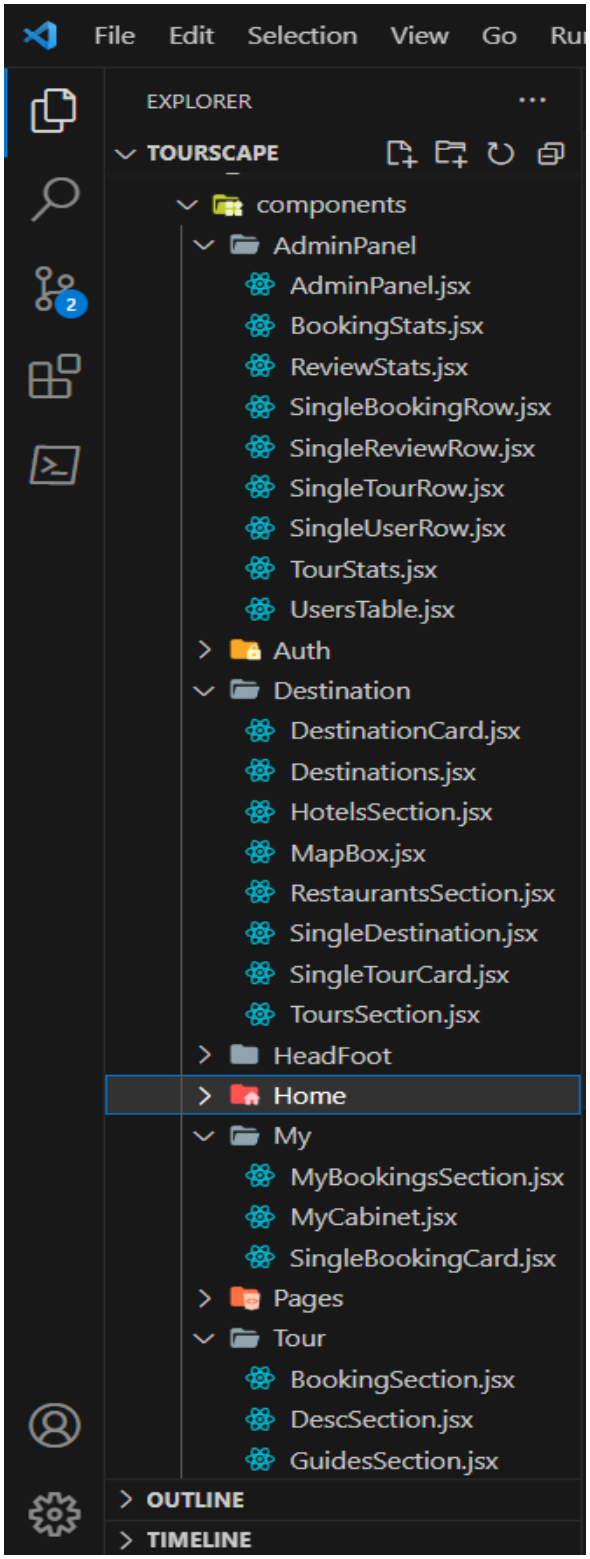

Рис. 2.3.2 Структура каталогу frontend (неповна)

2.4 Проектування структури бази даних за напрямком курсової роботи

Для збереження даних була обрана база даних MongoDB, як уже було сказано вище. Було створено 5 колекцій(ігнорувати hotels & restaurants):

- Users
- Tours
- Destinations
- Reviews
- Bookings

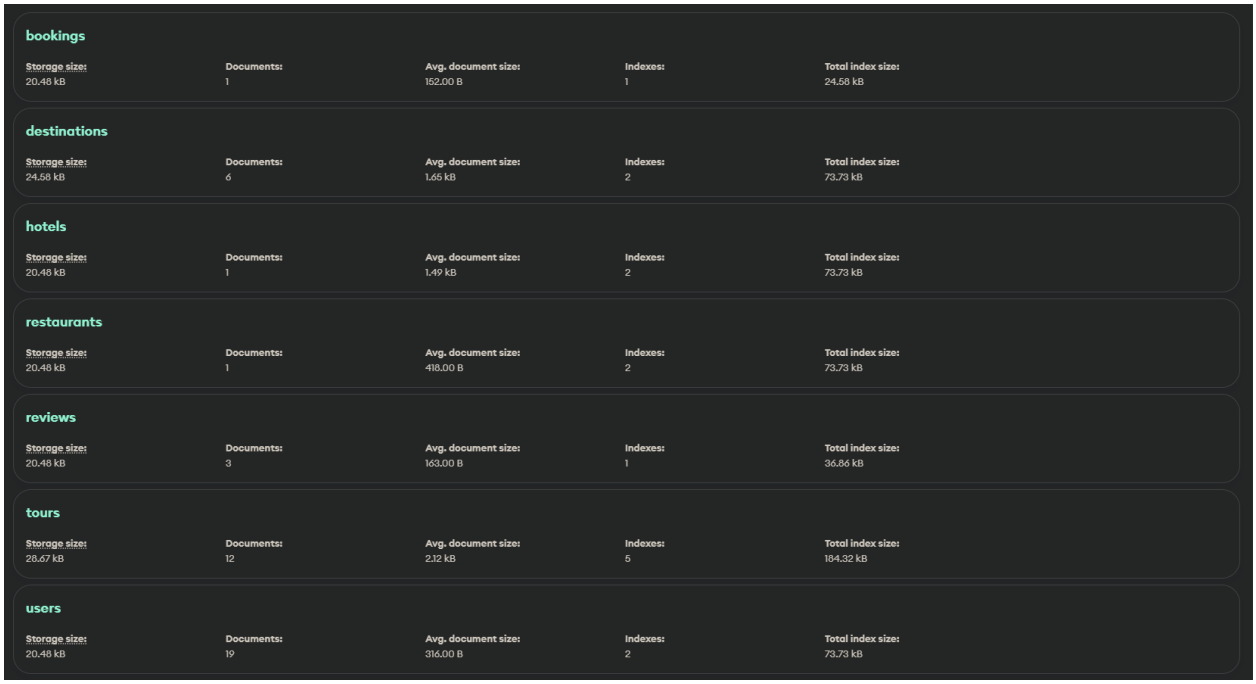

Рис. 2.4.1 Колекції БД

Розберемо структуру кожної колекції окремо (поля createdAt, updatedAt, _id не будуть описані, оскільки вони присутні у кожній колекції):

- Users

Назва поля	Тип поля
name	String
email	String
password	String
photo	String

role	String

Дана таблиця відповідає за зберігання користувачів.

- Tours

Назва поля	**Тип поля**
name	String
duration	Int32
maxGroupSize	Int32
difficulty	String
ratingsAverage	Int32
ratingQuantity	Int32
price	Int32
summary	String
photo	[String]
startDates	[Object]
secretTour	Boolean
startLocation	Object
locations	[Object]
destination	String
guides	[String]
slug	String
description	String

Таблиця відповідає за зберігання турів.

- Destinations

Назва поля	Тип поля
name	String
description	String

location	Object
slug	String
short_desc	String
photo	[String]

Таблиця відповідає за збереження міст.

- Bookings

Назва поля	**Тип поля**
tour	String
user	String
price	Int32
onDate	Date
numberOfPeople	Int32

Таблиця відповідає за збереження замовлень.

- Reviews

Назва поля	**Тип поля**
review	String
rating	Int32
tour	String
user	String

Таблиця відповідає за збереження відгуків.

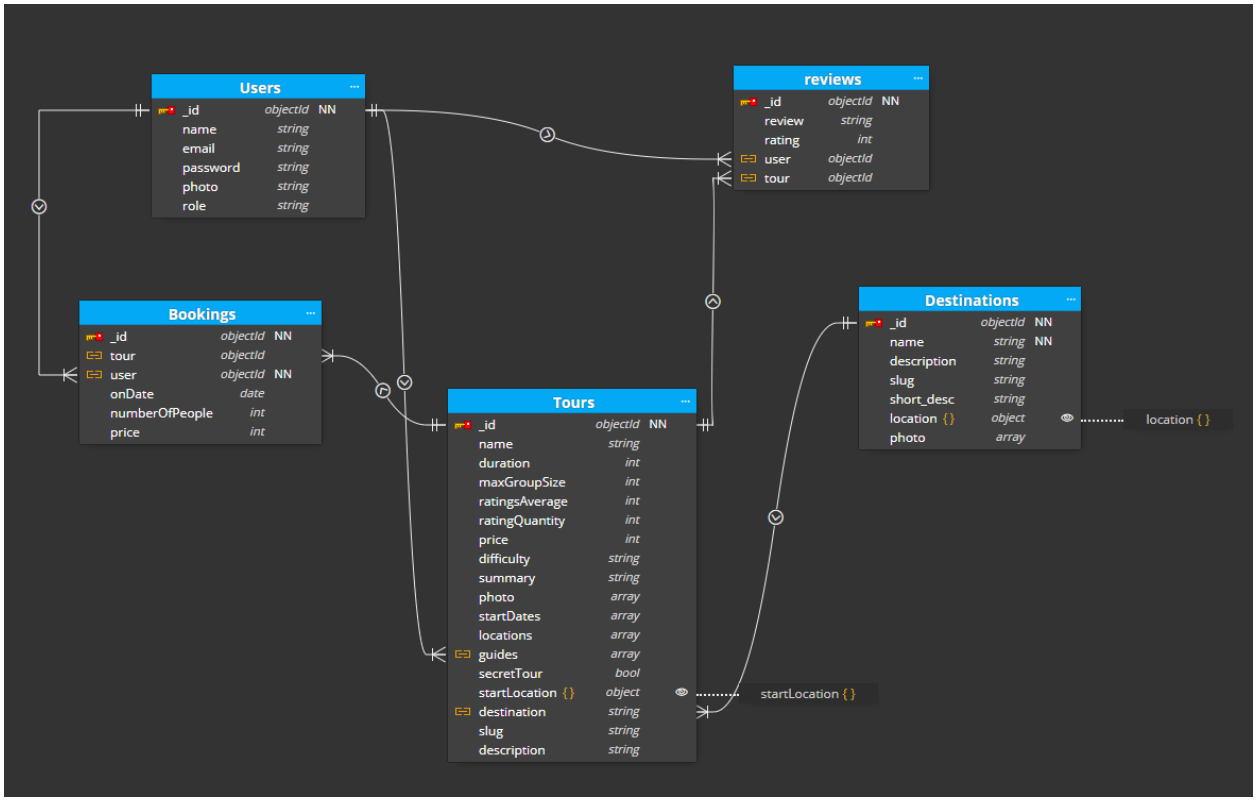

Рис. 2.4.2 Діаграма бази даних

Висновки до другого розділу

Під час виконання даного розділу було розроблено алгоритм роботи додатку та спроектовано діаграму базу даних. Досліджено та описано колекції та їх поля.

РОЗДІЛ 3 ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО РЕАЛІЗУВАННЯ

3.1. Опис роботи з програмним додатком

При переході на веб-сайт ми потрапляємо на головну сторінку.

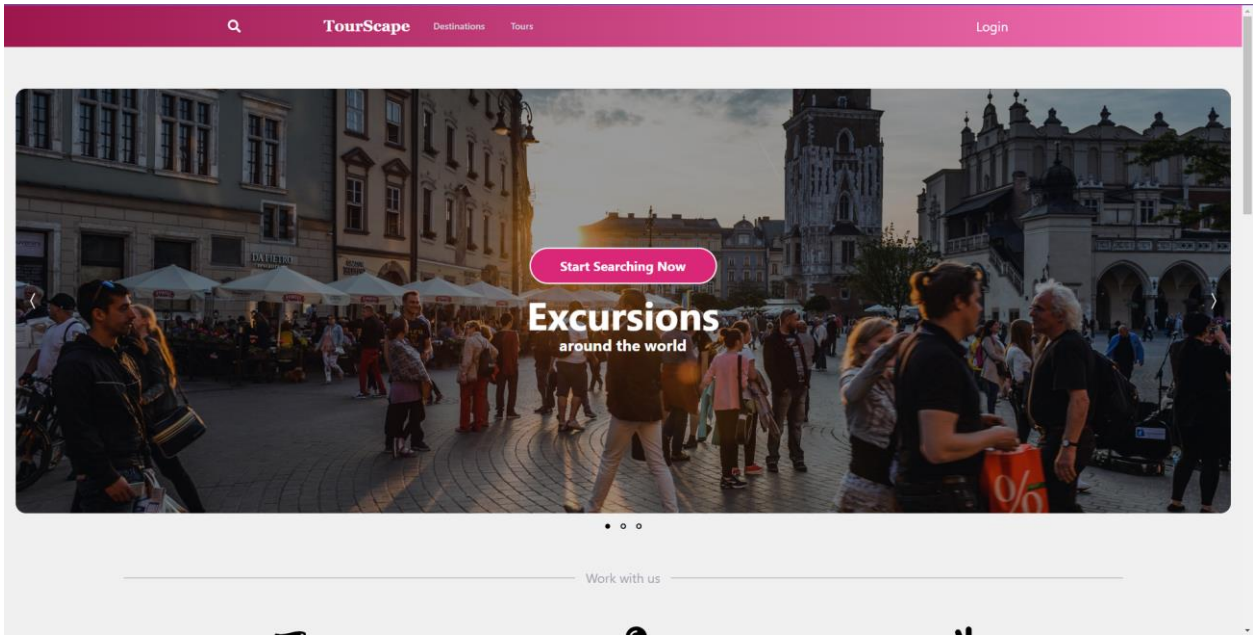

Рис. 3.1.1 Головна сторінка сайту

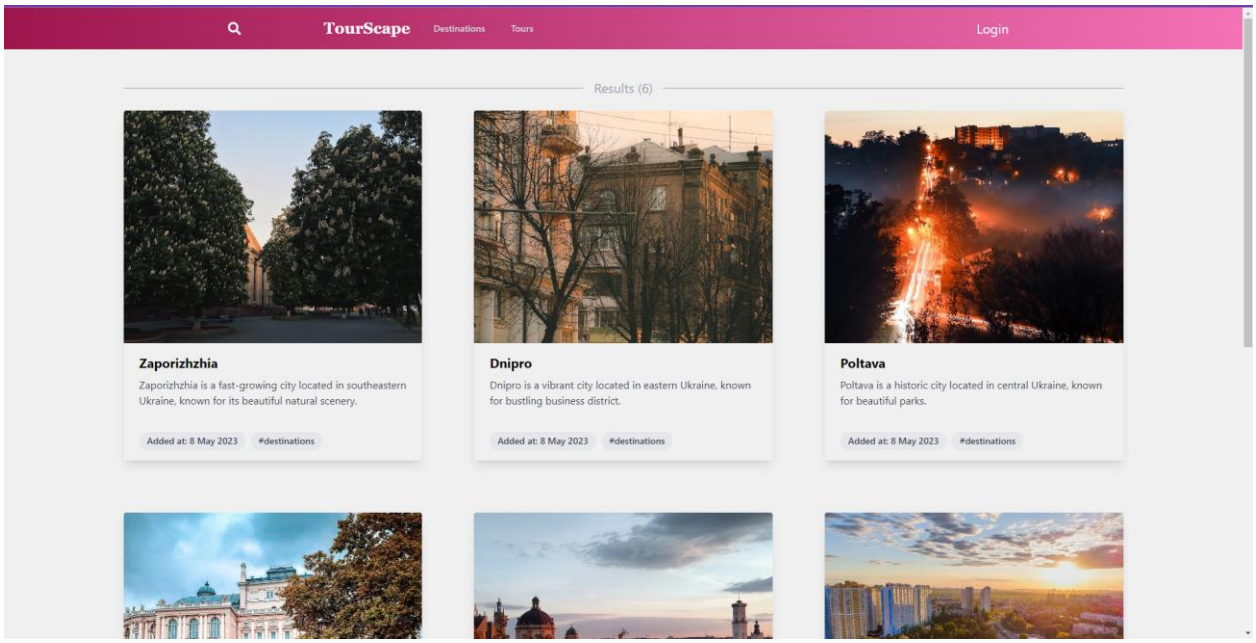

Рис. 3.1.2 Сторінка пунктів призначення (міст)

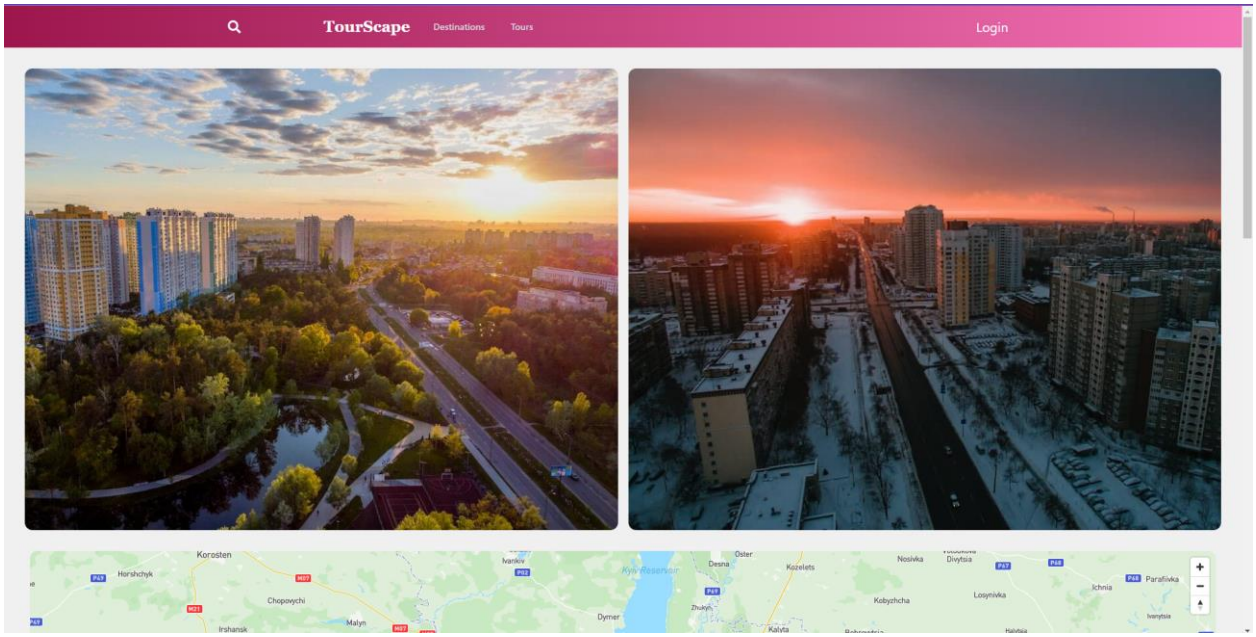

Рис. 3.1.3 Сторінка окремого міста

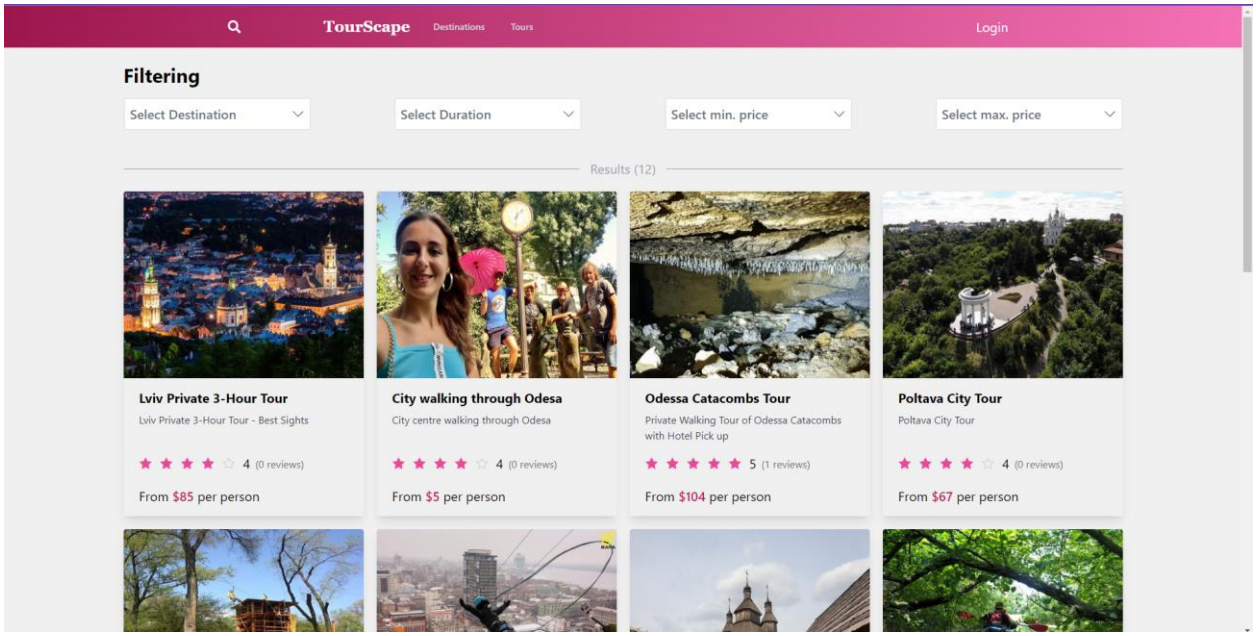

Рис. 3.1.4 Сторінка турів

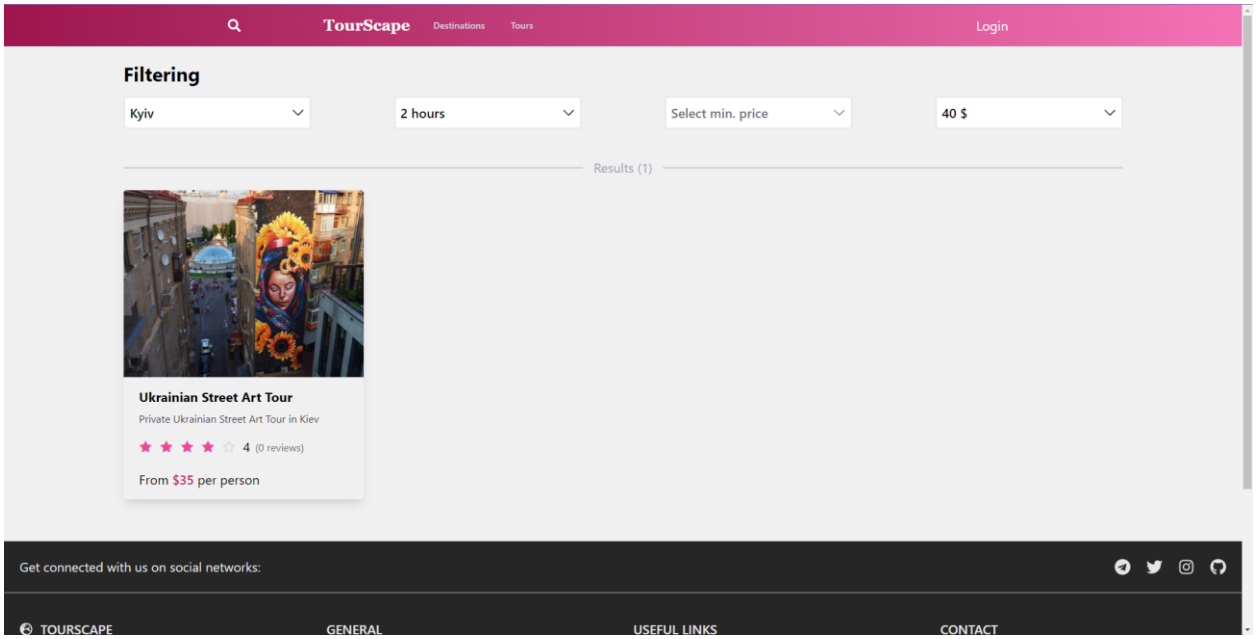

Рис. 3.1.5 Сторінка турів із фільтрами

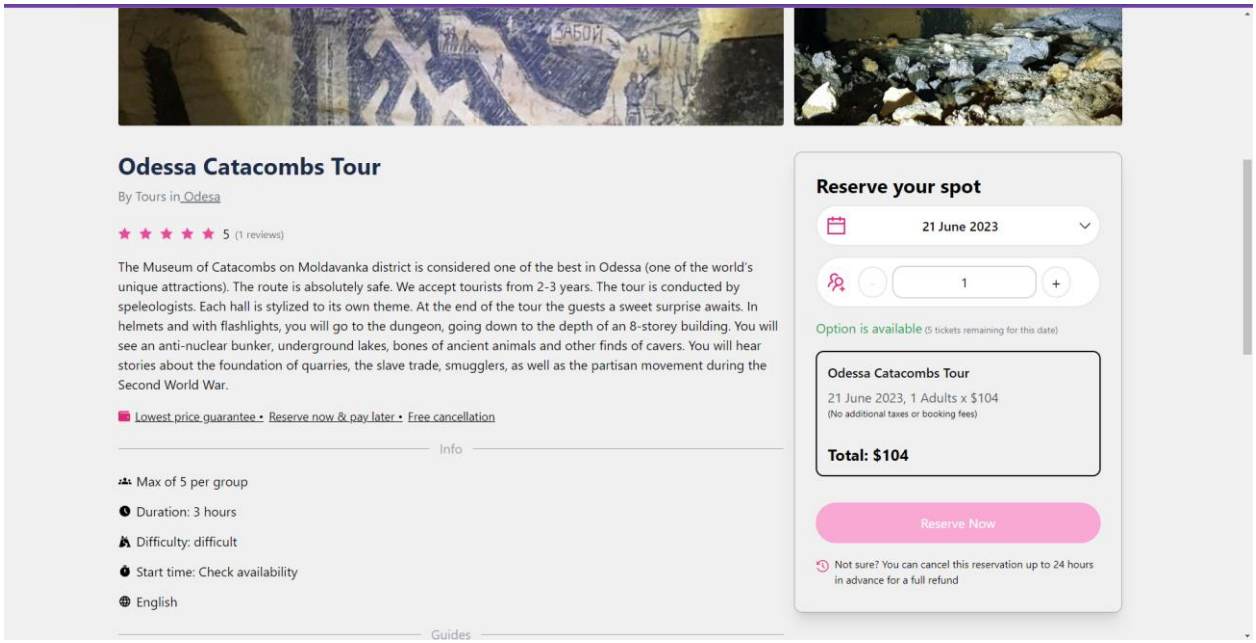

Рис. 3.1.6 Сторінка окремого туру

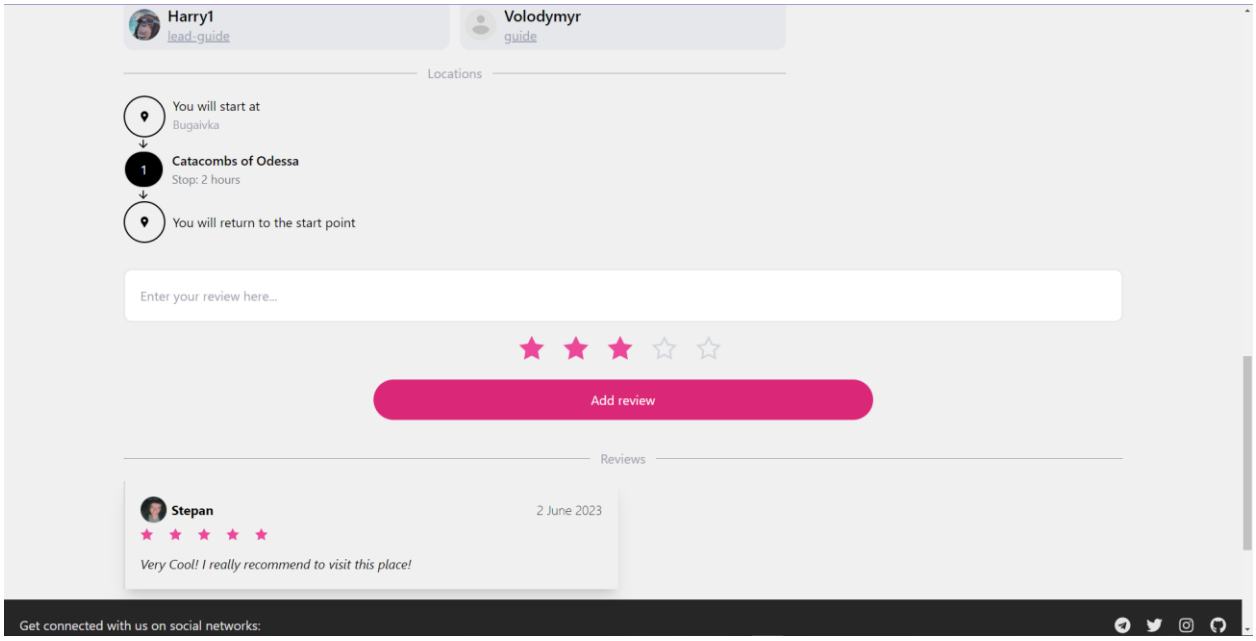

Рис. 3.1.7 Розділ із відгуками

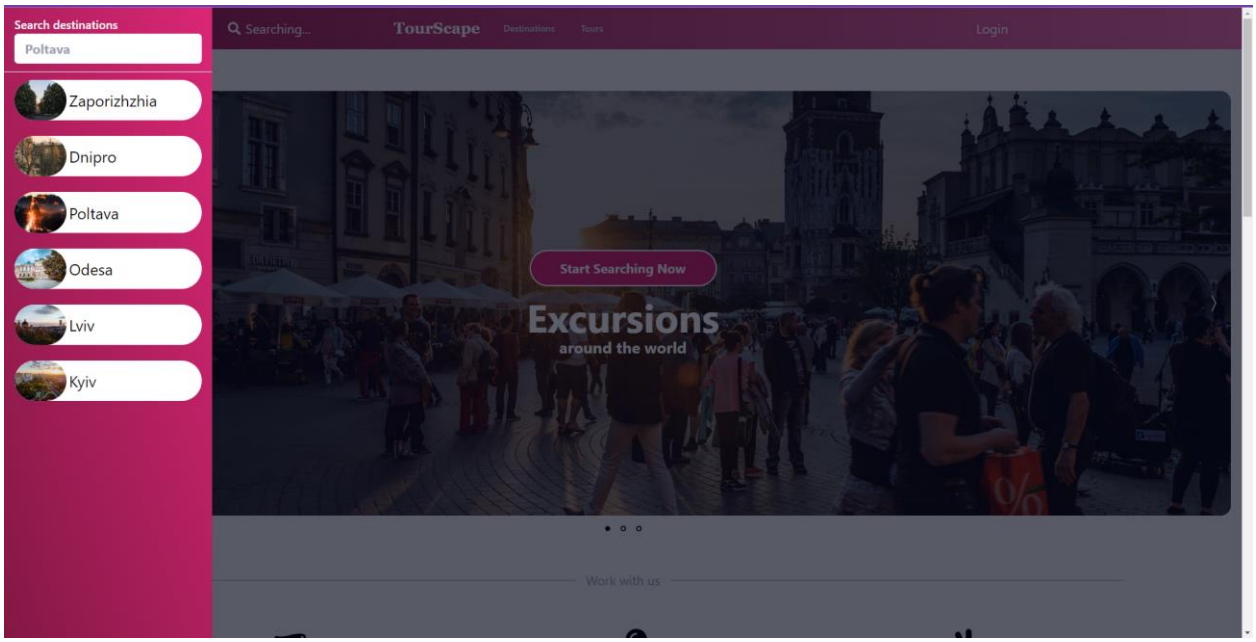

Рис. 3.1.7 Пошук міст

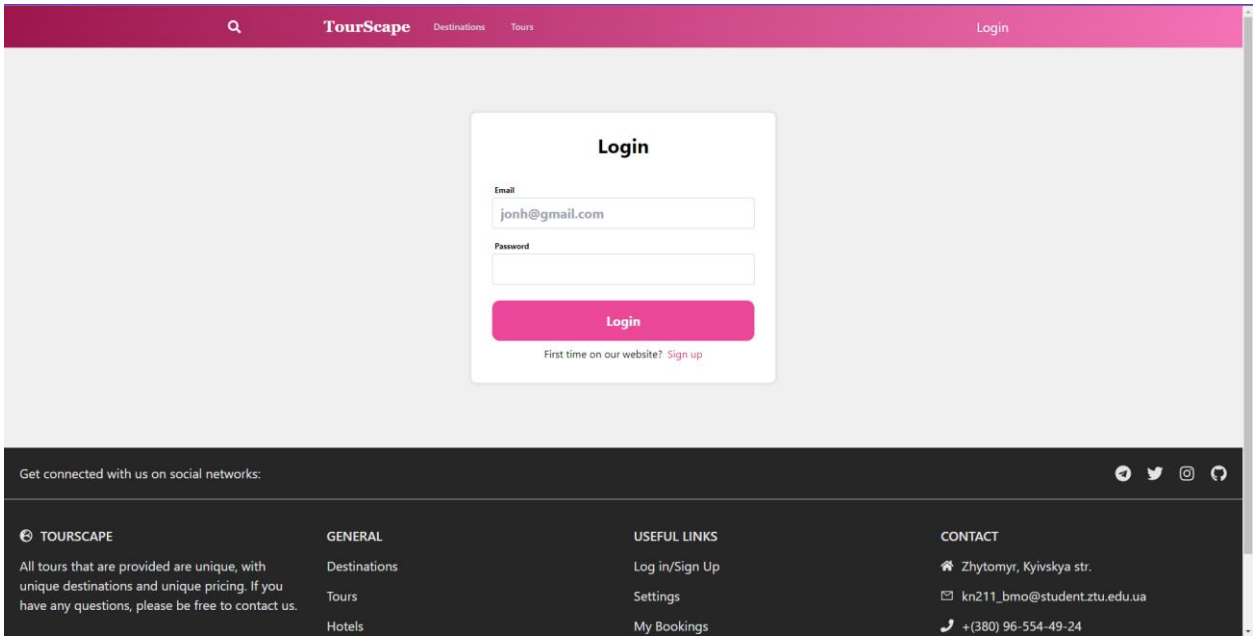

Рис. 3.1.8 Форма авторизації

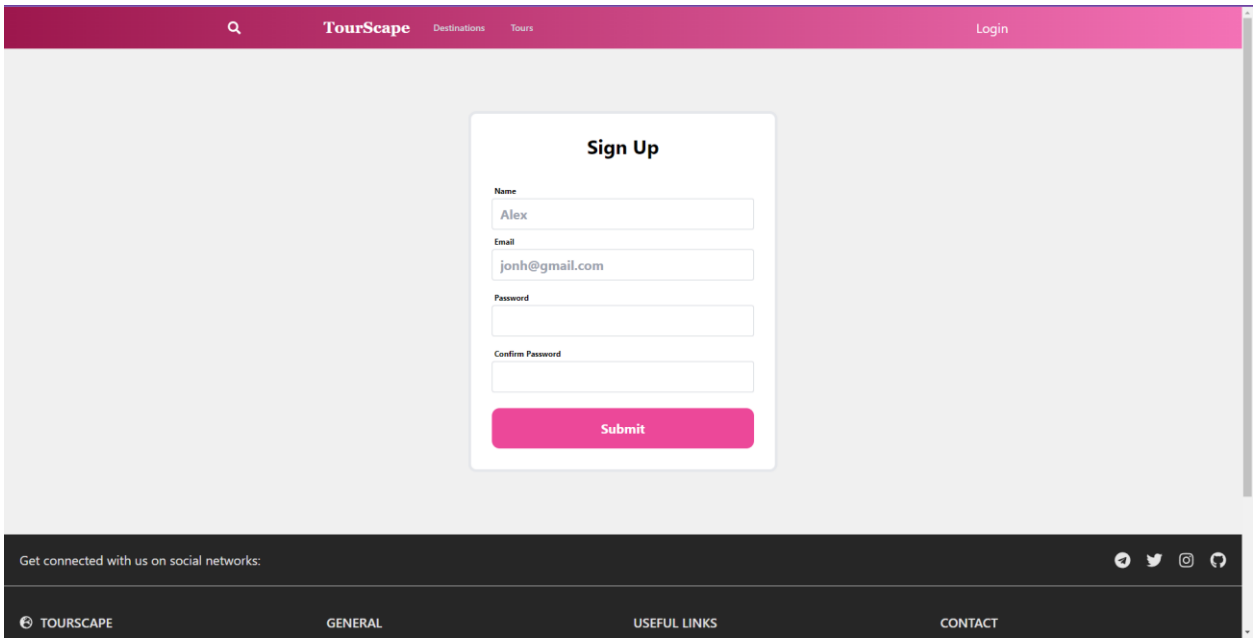

Рис. 3.1.9 Форма реєстрації

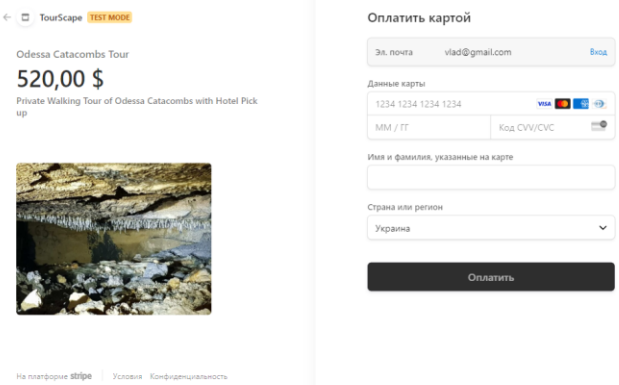

Рис. 3.1.10 Оформлення замовлення

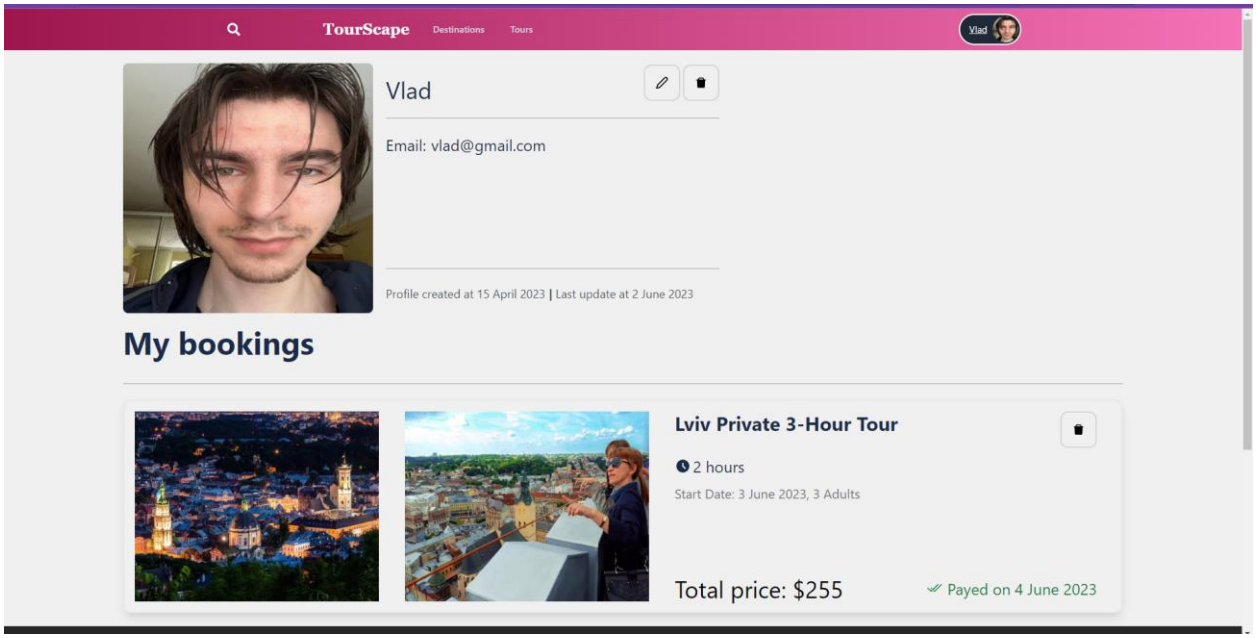

Рис. 3.1.11 Особистий кабінет

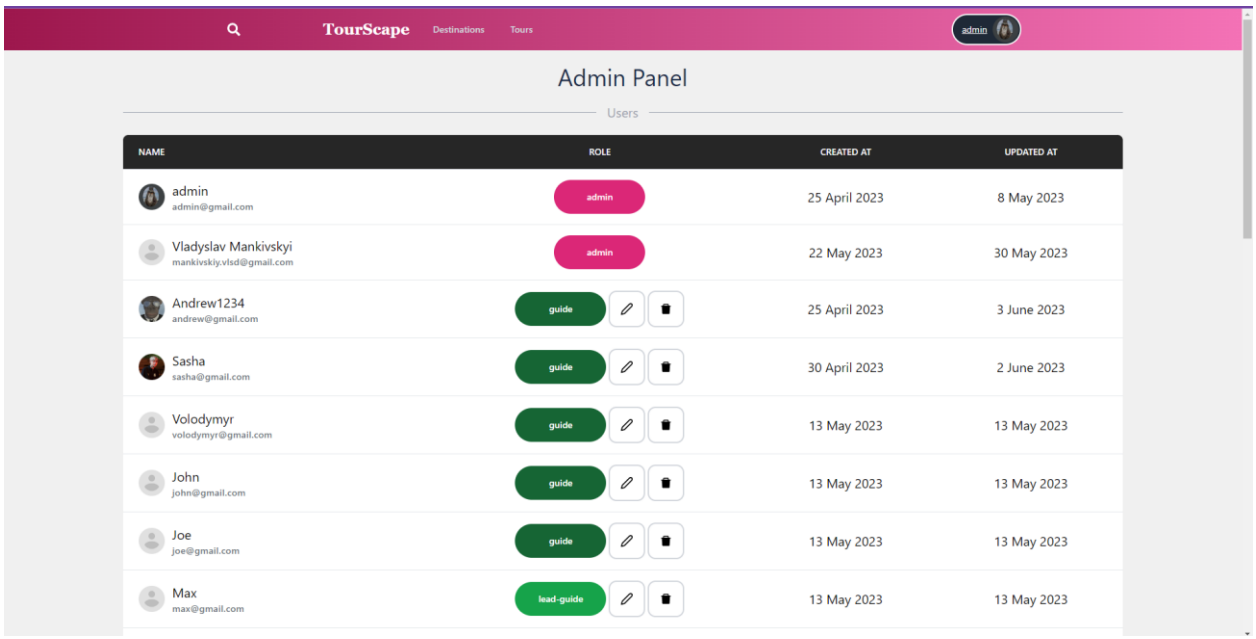

Рис. 3.1.12 Панель адміна (користувачі)

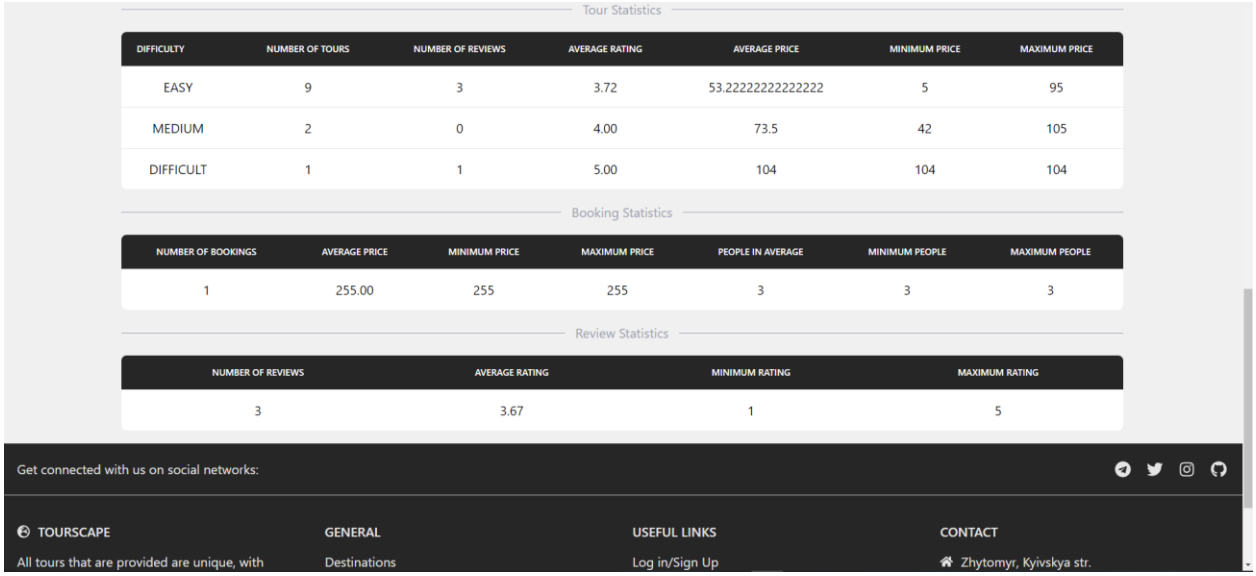

Рис. 3.1.13 Панель адміна (статистика)

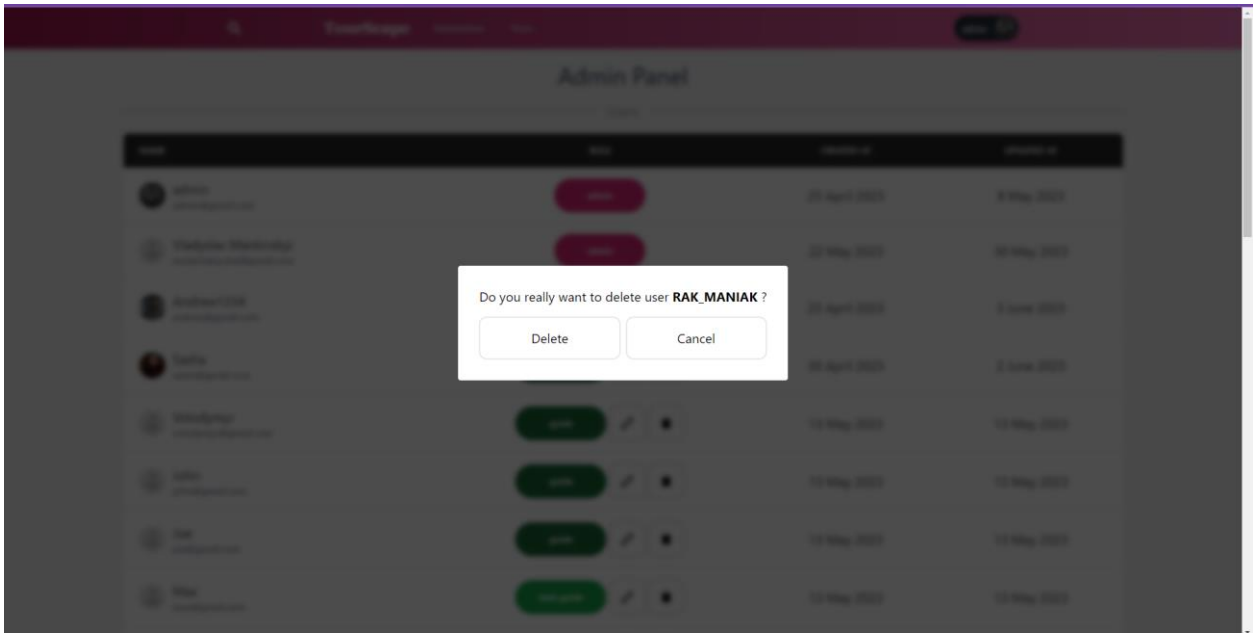

Рис. 3.1.14 Видалення користувача

3.2. Реалізація операцій обробки даних в БД

Розглянемо конфігураційний файл на серверній частині.

Конфігураційний файл містить рядок підключення до бази даних, api ключі до інших сервісів, таких як cloudinary, stripe та інші. Задля безпеки, скріншоту конфігураційного файлу не надається.

Підлючення до бази даних та запуск серверу:

```
const mongoose = require('mongoose')
const dotenv = require('dotenv')
dotenv.config({path:'backend/config.env'})

const DB = process.env.MONGO_URL;

const app = require('./app')


mongoose.connect(DB, {
    useNewUrlParser: true,
    useUnifiedTopology:true
}).then(() => console.log("DB connection established"));

const port = process.env.PORT || 5000;
const server = app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```

```
process.on("unhandledRejection", err => {
    console.log('UNHANDLED REJECTION! Shutting down...')
    console.log(err.name, err.message)
    server.close(()=>{
        process.exit(1)
    })
})
```

API написана як RESTful API, тому розглянемо, як працює обробка http запитів. Для початку у файлі app.js прописано роути, які і відповідають за свою адресу:

```
app.use('/api/users', userRoutes)
app.use('/api/tours', tourRoutes)
app.use('/api/destinations', destinationRoutes)
app.use('/api/reviews', reviewRoutes)
app.use('/api/bookings', bookingRoutes)
```

Розглянемо якийсь із роутів, наприклад userRoutes.

```
const express = require("express");
const userController = require('./../controllers/userController');
const authMiddleware = require('../middlewares/authMiddleware');
const authController = require('../controllers/authController');


const router = express.Router();


router.post('/signup', authController.signup)
router.post('/login', authController.login)
router.get('/logout', authController.logout)
router.post('/forgotPassword', authController.forgotPassword)
router.patch('/resetPassword/:token', authController.resetPassword)

router.use(authMiddleware.protect)

router.get('/me', userController.getMe, userController.getUser)
router.patch('/updateMyPassword', authController.updatePassword)
router.patch(
    '/updateMe',
    userController.uploadUserPhoto,
    userController.updateMe
```

```
);
router.delete('/deleteMe',authMiddleware.restrictTo('user'),
userController.deleteUserPhoto, userController.deleteMe)

router.use(authMiddleware.restrictTo('admin'))

router.get('/', userController.getAllUsers)
router.route('/:id').get(userController.getUser).patch(userController.
uploadUserPhoto,
userController.updateUser).delete(userController.deleteUserPhoto,
userController.deleteUser)


module.exports = router
```

Даний код містить обробники різних http запитів (get,post,patch,delete). Також можна побачити такі middleware функції, як restrictTo та protect. Protect відповідає за те, щоб користувач був авторизованим.

```
exports.protect = catchAsync(async (req, res, next) => {
    let token;
    if (req.headers.authorization &&
req.headers.authorization.startsWith('Bearer')) {
        token = req.headers.authorization.split(' ')[1];
    }  else if (req.cookies._auth) {
        token = req.cookies._auth;
    }


    if (!token) {
        return next(new AppError('You are not logged in. Please log in
to get access', 401))
    }

    const decoded = await promisify(jwt.verify)(token,
process.env.JWT_SECRET);
    const freshUser = await User.findById(decoded.id);
    if (!freshUser) {
        return next(new AppError('The user belonging to this token
does not longer exist', 401))
    }

    if (freshUser.changedPasswordAfter(decoded.iat)) {
```

```
        return next(new AppError('User recently changed password.
Pleaser log in to get access', 401))
    }


    req.user = freshUser;
    // req.locals.user = freshUser;
    next();
})
```

restrictTo дозволяє обмежити функції для певних ролей.

```
exports.restrictTo = (...roles) => {
    return (req, res, next) => {
        if (!roles.includes(req.user.role)) {
            return next(new AppError('You dont have permission to
perform this action', 403))
        }
        next();
    }
}
```

Перейдемо до CRUD операцій. Усі операції виконуються у файлі handlerFactory. Принцип роботи такий: у контроллері є методи, які викликаються як обробники певних url адрес. Далі ці методи викликають потрібний метод у файлі handlerFactory, передаючи туди назву моделі. Як приклад, можна привести декілька методів із цього файлу:

```
exports.deleteOne = Model => catchAsync(async (req, res, next) => {
    const doc = await Model.findByIdAndDelete(req.params.id)
    if (!doc) {
        next(new AppError('No tour found with that id', 404));
        return;
    }
    res.status(204).json({
        status: 'success',
        data: 'Object deleted successfully'
    })
})
```

```
exports.createOne = Model =>
    catchAsync(async (req, res, next) => {
```

```
        const doc = await Model.create(req.body);

        res.status(201).json({
            status: 'success',
            data: {
                data: doc
            }
        });
    });
```

Із методом getAll цікавіше, адже тут доступні такі функції як sort, filter, search, limitFields, paginate. Усі ці методи знаходяться у класі ApiFeatures:

```
const { Query } = require("mongoose");

class APIFeatures {
    constructor(query, queryString) {
        this.query = query;
        this.queryString = queryString;
    }
    filter() {
        const queryObj = { ...this.queryString }
        const excludedFields = ['page', 'sort', 'limit', 'fields',
'search']
        excludedFields.forEach(el => delete queryObj[el])
        let queryStr = JSON.stringify(queryObj);
        queryStr = queryStr.replace(/\b(gte|gt|let|lt)\b/g, match =>
`$${match}`)
        this.query = this.query.find(JSON.parse(queryStr));
        return this;
    }
    search() {
        if (this.queryString.search) {
            const search = this.queryString.search;
            const keyword = {name: {$regex : search, $options:"i"}}
            this.query = this.query.find(keyword);
        }
        return this;
    }
    sort() {
        if (this.queryString.sort) {
            const sortBy = this.queryString.sort.split(',').join(' ');
            this.query = this.query.sort(sortBy)
```

```
        } else {
            this.query = this.query.sort('-createdAt');
        }
        return this;
    }
    limitFields() {
        if (this.queryString.fields) {
            const fields = this.queryString.fields.split(',').join('
');
            this.query = this.query.select(fields);
        } else {
            this.query = this.query.select('-__v');
        }
        return this;
    }
    paginate() {
        const page = this.queryString.page * 1 || 1;
        const limit = this.queryString.limit * 1 || 100;
        const skip = (page - 1) * limit;
        this.query = this.query.skip(skip).limit(limit);
        return this;
    }
}

module.exports = APIFeatures
```

Висновки до третього розділу

В даному розділі було описано процес роботи із програмним додатком для різних ролей. Було продемонстровано можливості веб-сайту та наведено прикладу коду для кращого розуміння його роботи.

РОЗДІЛ 4. АДМІНІСТРУВАННЯ БАЗ ДАНИХ

4.1. Розробка заходів захисту інформації в БД

Було розроблено такі засоби захисту як хешування паролю, розподілення функцій по ролям, використання сторонніх бібліотек задля захисту від атак. Прикладами таких бібліотек є xss-clean, express-mongo-sanitize, helmet,express-rate-limit, hpp:

```
app.use(sanitize());
app.use(xss());
app.use(hpp({
    whitelist: [
        'duration',
        'ratingsQuantity',
        'ratingsAverage',
        'maxGroupSize',
        'difficulty',
        'price'
    ]
}));
```

Хешування паролю відбувається у pre save middleware функції. Використано бібліотеку bcrypt.

```
userSchema.pre('save', async function (next) {
    if (!this.isModified('password')) return next();

    this.password = await bcrypt.hash(this.password, 12);
    this.passwordConfirm = undefined;
    next();
})
```

Розглянемо матрицю доступу для кожної ролі:

Колекції / Ролі	Матриця доступу			
	Незаавторизований	Авторизований	Адміністратор	Лід-гід
tours	1	1	5	1
destinations	1	1	5	1

users	0	0	1,2,4	0
reviews	1	5	1,4	1
bookings	1	5	1	0

4.2. Налаштування параметрів роботи з MongoDB

Адміністрування бази даних буде проводитися на MongoAtlas, а також MongoDBCompass, що є зручнішим варіантом.

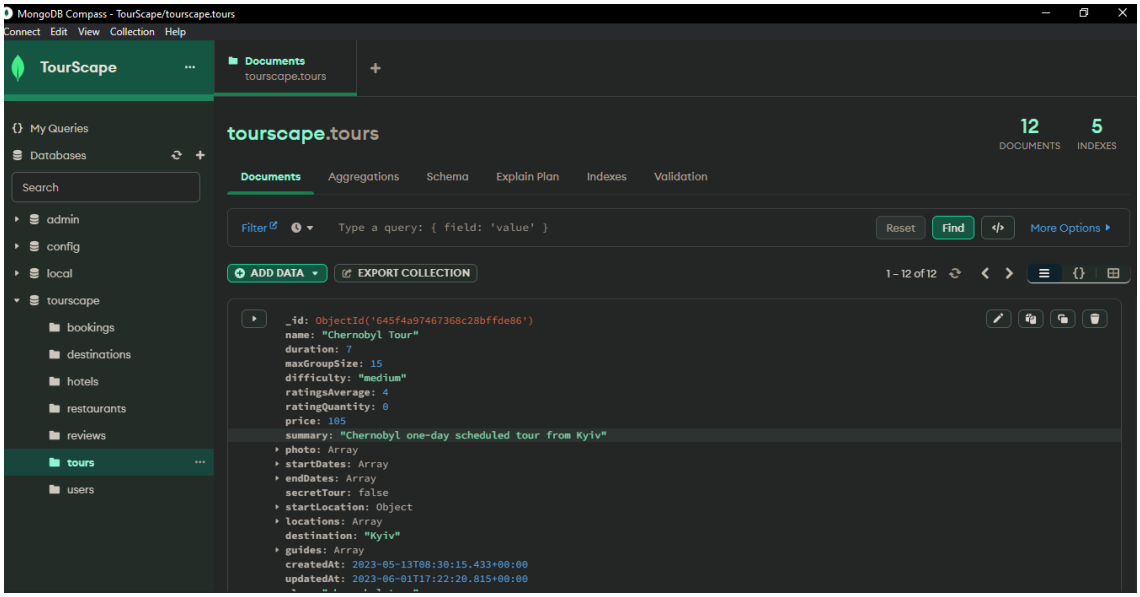

Рис. 4.2.1 Робота із БД у MongoDBCompass

Висновок до четвертого розділу

У даному розділі були наведені певні заходи безпеки при роботі із БД, наведено матрицю доступу для ролей. Продемонстровано роботу із БД у MongoDBCompass.

ВИСНОВКИ

У результаті виконання курсової роботи було отримано досвід у написанні вед-додатку із використанням бази даних. Отримано практичні навички роботи із MongoDB та Mongoose. Були проаналізовані аналоги, що допомогло зрозуміти потреби та вектор руху при розробці сайту. Було розроблено алгоритми роботи та завдяки йому, написано програмний код.

Було продемонстровано процес роботи із веб-додатком та прийняті заходи для безпеки сайту. Таким чином, був створений приємний на вигляд веб-сайт із широким функціоналом із використанням таких технологій, як NodeJS, ReactJS, Express, Mongoose, TailwindCSS, та і MongoDB. Веб сайт відповідає усім поставленим цілям та працює як і задумано розробником.

СПИСОК ЛІТЕРАТУРИ

1. Дмитрий Кирсанов. “Веб-дизайн” [електронний ресурс] - 2006. Режим доступу: http://web-diz.com.ua/skachat/veb-dizayn--dmitriy-kirsanovskachatbesplatno/
2. Mongo DB Docs. Режим доступу: https://www.mongodb.com/docs/
3. "Node.js in Action" - Mike Cantelon, Marc Harter, TJ Holowaychuk, Nathan Rajlich. Режим доступу: https://www.manning.com/books/node-js-in-action
4. "MongoDB: The Definitive Guide" - Kristina Chodorow, Michael Dirolf. Режим доступу: https://www.mongodb.com/mongodb-the-definitive-guide
5. "React Up and Running" - Stoyan Stefanov. Режим доступу: https://www.amazon.com/React-Up-Running-Stoyan-Stefanov/dp/1491931825
6. Сервіс: NodeSchool. Режим доступу: https://nodeschool.io/
7. Сервіс: React.js Documentation. Режим доступу: https://legacy.reactjs.org/docs/getting-started.html
8. Сервіс: Express.js Режим доступу: https://expressjs.com/
9. Сервіс: Mongoose - об'єктно-документний (ODM) модуль для Node.js та MongoDB, що надає простий спосіб взаємодії з базою даних. Режим доступу: https://mongoosejs.com/

ДОДАТКИ

```
const path = require("path");
const express = require('express');
const morgan = require('morgan');
const rateLimit = require('express-rate-limit');
const helmet = require('helmet');
const sanitize = require("express-mongo-sanitize");
const xss = require("xss-clean");
const hpp = require('hpp');
const cookieParser = require('cookie-parser');

const AppError = require('./utils/appError');
const globalErrorHandler = require('./middlewares/errorMiddleware');

const bookingController = require('./controllers/bookingController')

const userRoutes = require("../backend/routes/userRoutes")
const tourRoutes = require("../backend/routes/toursRoutes")
const hotelRoutes = require("./routes/hotelRoutes")
const destinationRoutes =
require("../backend/routes/destinationRoutes")
const restaurantRoutes = require("../backend/routes/restaurantRoutes")
const reviewRoutes = require("../backend/routes/reviewRoutes")
const bookingRoutes = require("../backend/routes/bookingRoutes")



const app = express();


// Set security HTTP headers
app.use(
    helmet({
        crossOriginEmbedderPolicy: false,
        crossOriginResourcePolicy: {
            allowOrigins: [
                'https://res.cloudinary.com',
                'https://icon-library.com',
                'https://media.giphy.com'
            ],
        },
        contentSecurityPolicy: {
            directives: {
                ...helmet.contentSecurityPolicy.getDefaultDirectives()
,
```

```
                'img-src': [
                    "'self'",
                    'data:',
                    'https://res.cloudinary.com',
                    'https://icon-library.com',
                    'https://media.giphy.com',
                    'blob:',
                ],
                'media-src': [
                    "'self'",
                    'data:',
                    'https://res.cloudinary.com'
                ],
                'default-src': [
                    "'self'",
                    'data:',
                    'https://res.cloudinary.com',
                ],
                'script-src': [
                    "'self'",
                    'data:',
                    'https://www.mapbox.com',
                    'https://api.mapbox.com'
                ],
                'style-src': [
                    "'self'",
                    "'unsafe-inline'",
                    'https://api.mapbox.com/'
                ],
                'worker-src': [
                    "'self'",
                    'blob:'
                ],
                'child-src': [
                    'blob:'
                ],
                'connect-src': [
                    "'self'",
                    'https://events.mapbox.com/',
                    'https://api.mapbox.com/',
                    'https://res.cloudinary.com/',
                    'https://*.tiles.mapbox.com/'
                ]
            },
```

```
        },
    })
);

// Development logging
if (process.env.NODE_ENV === 'development') {
    app.use(morgan('dev'));
}


app.post(
    '/webhook-checkout',
    express.raw({ type: 'application/json' }),
    bookingController.checkoutWebhook
);

app.use(express.json({ limit: '100mb' }));
app.use(express.urlencoded({ extended: true, limit: '100mb' }));
app.use(cookieParser());

// Data sanitization against NoSQL query injection
app.use(sanitize());

// Data sanitization against XSS
app.use(xss());

// Prevent parameters pollution
app.use(hpp({
    whitelist: [
        'duration',
        'ratingsQuantity',
        'ratingsAverage',
        'maxGroupSize',
        'difficulty',
        'price'
    ]
}));

app.use('/api/users', userRoutes)
app.use('/api/tours', tourRoutes)
app.use('/api/destinations', destinationRoutes)
app.use('/api/hotels', hotelRoutes)
app.use('/api/restaurants', restaurantRoutes)
app.use('/api/reviews', reviewRoutes)
```

```
app.use('/api/bookings', bookingRoutes)

app.use(express.static(path.join(__dirname, '../frontend/dist')));

app.all('*', (req, res, next) => {
    if (req.originalUrl.includes('api'))
        return next(
            new AppError(`Can't find ${req.originalUrl} on this
server!`, 404)
        );
    res.sendFile(path.join(__dirname, '../frontend/dist/index.html'));
});


// Error handlers
app.use(globalErrorHandler)

module.exports = app;
```

```
process.on('uncaughtException', err => {
    console.log('UNHANDLED Exception! Shutting down...')
    console.log(err)
    process.exit(1)
})


const mongoose = require('mongoose')
const dotenv = require('dotenv')
dotenv.config({path:'backend/config.env'})

const DB = process.env.MONGO_URL;

const app = require('./app')


mongoose.connect(DB, {
    useNewUrlParser: true,
    useUnifiedTopology:true
}).then(() => console.log("DB connection established"));


const port = process.env.PORT || 5000;
const server = app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```

```
process.on("unhandledRejection", err => {
    console.log('UNHANDLED REJECTION! Shutting down...')
    console.log(err.name, err.message)
    server.close(()=>{
        process.exit(1)
    })
})
```

```
const {promisify} = require('util');
const User = require('./../models/userModel');
const catchAsync = require('./../utils/catchAsync');
const AppError = require('./../utils/appError');
const sendEmail = require('./../utils/email');
const jwt = require('jsonwebtoken')
const crypto = require("crypto");
const Email = require('./../utils/email');

const signToken = id => {
    return jwt.sign({id}, process.env.JWT_SECRET, {expiresIn:
process.env.JWT_EXPIRES_IN})
}

const createSendToken = (user, statusCode, res) => {
    const token = signToken(user._id);

    user.password = undefined

    res.send({
        status: 'success',
        token,
        data: {user}
    });
};

exports.signup = catchAsync(async (req, res, next) => {
    const newUser = await User.create({
        name: req.body.name,
        email: req.body.email,
        password: req.body.password,
        passwordConfirm: req.body.passwordConfirm,
```

```
        passwordChangedAt: req.body.passwordChangedAt,
        role: req.body.role
    });

    /*const url = `${req.protocol}://${req.get('host')}/me`;
    await new Email(newUser, url).sendWelcome();*/

    createSendToken(newUser, 201, res)
})

exports.login = catchAsync(async (req, res, next) => {
    const {email, password} = req.body;

    if (!email || !password) {
        return next(new AppError('Please provide email and password',
400))
    }

    const user = await User.findOne({email}).select('+password')

    if (!user || !await user.correctPassword(password, user.password))
{
        return next(new AppError('Incorrect email or password', 401))
    }

    createSendToken(user, 200, res)
})

exports.logout = (req, res) => {
    res.cookie('jwt', 'loggedout', {
        expires: new Date(Date.now() + 10 * 1000),
        httpOnly: true
    });
    res.status(200).json({ status: 'success' });
};


exports.forgotPassword = catchAsync(async (req, res, next) => {
    const user = await User.findOne({email: req.body.email})

    if (!user) {
        return next(new AppError('There is no user with that email
address'))
    }
```

```
    const resetToken = user.createPasswordResetToken();
    await user.save({validateBeforeSave: false});
    const resetURL =
`${req.protocol}://${req.get('host')}/api/v1/resetPassword/${resetToke
n}`
    const message = `Forgot your password? Submit a PATCH request with
your new password and passwordConfirm to ${resetURL}.\nIf you didnt
forget your password, please ignore this email.`
    try {
        await sendEmail({
            email: user.email,
            subject: 'Your password reset token (valid only for 10
minutes)',
            message
        })

        res.status(200).json({
            status: 'success',
            message: 'Token sent to email.'
        })
    } catch (err) {
        user.passwordResetToken = undefined
        user.passwordResetExpires = undefined
        await user.save({validateBeforeSave: false})

        return next(new AppError(err, 500))
    }
})
exports.resetPassword = catchAsync(async (req, res, next) => {
    const hashedToken = crypto
        .createHash('sha256')
        .update(req.params.token)
        .digest('hex');

    const user = await User.findOne({
        passwordResetToken: hashedToken,
        passwordResetExpires: {$gt: Date.now()}
    });

    // 2) If token has not expired, and there is user, set the new
password
    if (!user) {
        return next(new AppError('Token is invalid or has expired',
400));
```

```
    }
    user.password = req.body.password;
    user.passwordConfirm = req.body.passwordConfirm;
    user.passwordResetToken = undefined;
    user.passwordResetExpires = undefined;
    await user.save();




    // 4) Log the user in, send JWT
    createSendToken(user, 200, res)
})

exports.updatePassword = catchAsync(async (req, res, next) => {
    // 1) Get user from collection
    const user = await User.findById(req.user.id).select('+password');

    // 2) Check if POSTed current password is correct
    if (!(await user.correctPassword(req.body.passwordCurrent,
user.password))) {
        return next(new AppError('Your current password is wrong.',
401));
    }

    // 3) If so, update password
    user.password = req.body.password;
    user.passwordConfirm = req.body.passwordConfirm;
    await user.save();

    // 4) Log user in, send JWT
    createSendToken(user, 200, res);
});
const Booking = require("../models/bookingModel")
const Tour = require("../models/tourModel")
const Hotel = require("../models/hotelModel")
const Restaurant = require("../models/restaurantModel")
const catchAsync = require("../utils/catchAsync")
const factory = require('../utils/handlerFactory')
const AppError = require("../utils/appError")
const User = require("../models/userModel")
const stripe = require('stripe')(process.env.STRIPE_SECRET_KEY)

exports.setModelUserIds = (req, res, next) => {
```

```
    if (req.params.tourId) {
        if (!req.body.tour) req.body.tour = req.params.tourId
    }
    if (req.params.hotelId) {
        if (!req.body.hotel) req.body.hotel = req.params.hotelId
    }
    if (req.params.restaurantId) {
        if (!req.body.restaurant) req.body.restaurant =
req.params.restaurantId
    }
    if (!req.body.user) req.body.user = req.user.id
    next();
}

exports.getCheckoutSession = catchAsync(async (req, res, next) => {
    const { item, numberOfPeople, startDate } = req.body;
    const { price, name, summary, photo, slug } = item;

    const successURL = `${req.protocol}://${req.get('host')}/my`;
    const cancelURL =
`${req.protocol}://${req.get('host')}/tours/${slug}`;

    const images = photo.map(photo => String(photo));
    const unitAmount = price * numberOfPeople * 100;

    const session = await stripe.checkout.sessions.create({
        payment_method_types: ['card'],
        mode: 'payment',
        customer_email: req.user.email,
        success_url: successURL,
        client_reference_id: item._id,
        cancel_url: cancelURL,
        line_items: [{
            price_data: {
                currency: 'usd',
                unit_amount: unitAmount,
                product_data: {
                    name,
                    description: summary,
                    images,
                },
            },
            quantity: 1,
        }],
```

```
      metadata: {
         startDate,
         numberOfPeople
      }
   });

   res.status(200).json({
      status: 'success',
      session
   });
});

const updateAvailablePlaces = catchAsync(async (date,
numberOfPeople,client_reference_id) => {
   const tour = await Tour.findById(client_reference_id)
   const newDate = new Date(date).toISOString()
   const availablePlaces = tour.startDates.find(sd =>
sd.date.toISOString() === newDate).availablePlaces;
   const updatedAvailablePlaces = availablePlaces - numberOfPeople;
   tour.startDates.find(sd => sd.date.toISOString() ===
newDate).availablePlaces = updatedAvailablePlaces
   tour.markModified('startDates')
   await tour.save();
})


exports.checkoutWebhook = catchAsync(async (req, res, next) => {
   const sig = req.headers['stripe-signature'];


   try {
      const event = stripe.webhooks.constructEvent(
         req.body,
         sig,
         process.env.STRIPE_SECRET_WEBHOOK
      );


      if (event.type === 'checkout.session.completed') {
         console.log("COMPLETED!")
         const session = event.data.object;


         const { client_reference_id, metadata, customer_email,
amount_total } = session;
```

```
        const { startDate, numberOfPeople } = metadata;

        updateAvailablePlaces(startDate, numberOfPeople,
client_reference_id);

        const user = await User.findOne({ email: customer_email })
        const newStartDate = new Date(startDate).toISOString();

        await Booking.create({
           tour: client_reference_id,
           user: user._id,
           price: amount_total / 100,
           onDate: newStartDate,
           numberOfPeople
        });
     }
   } catch (err) {
     return next(new AppError(`Webhook error: ${err}`, 400));
   }

   res.status(200).json({ received: true });
});

exports.getMyBookings = catchAsync(async (req, res, next) => {
   const myBookings = await Booking.find({ user: req.user._id
}).populate({ path: "tour hotel restaurant", select: "name duration
photo" })

   res.status(200).json({
     status: 'success',
     data: {
        data: myBookings
     }
   });
})

exports.getBookingStats = catchAsync(async (req, res, next) => {
   const stats = await Booking.aggregate([
     {
        $match: {}
     },
     {
        $group: {
```

```
            _id: { $toUpper: '$tour.name' },
            num: { $sum: 1 },
            avgPrice: { $avg: '$price' },
            minPrice: { $min: '$price' },
            maxPrice: { $max: '$price' },
            averagePeople: { $avg: '$numberOfPeople' },
            minPeople: { $min: '$numberOfPeople' },
            maxPeople: { $max: '$numberOfPeople' }
         }
      },
      {
         $sort: {
            avgPrice: 1
         }
      }
   ]);
   res.status(200).json({
      status: 'success',
      data: { stats }
   })
})









exports.getAllBookings = factory.getAll(Booking, { path: 'tour',
select: 'name photo duration' })

exports.getBooking = factory.getOne(Booking)

exports.createBooking = factory.createOne(Booking)

exports.updateBooking = factory.updateOne(Booking)

exports.deleteBooking = factory.deleteOne(Booking)
```

```
const Destination = require("../models/destinationModel")
const factory = require('../utils/handlerFactory')
const cloudinary = require('./../utils/cloudinary');
const catchAsync = require("../utils/catchAsync");
```

```
const uploadDestPhoto = publicId => (req, res, next) => {
   cloudinary.createMulti('photo', 'Destinations', publicId, 900,
700)(req, res, (err) => {
      if (err) return next(err);
      if (req.files && req.files.length>0) {
         req.body.photo = req.files.map(el=>el.path);
      }
      next();
   })
};

const deleteDestPhoto = publicId => (req, res, next) => {
   cloudinary.deleteSingle('Destinations', publicId);
   next();
};

exports.uploadDestPhoto = (req, res, next) => {
   uploadDestPhoto(req.params.id)(req, res, next);
}

exports.deleteDestPhoto = (req, res, next) => {
   deleteDestPhoto(req.params.id)(req, res, next);
}


exports.getAllDestinations = factory.getAll(Destination)

exports.getDestination = factory.getOne(Destination, { path: 'tours
hotels restaurants' })

exports.createDestination = factory.createOne(Destination)

exports.updateDestination = factory.updateOne(Destination)

exports.deleteDestination = factory.deleteOne(Destination)
```

```
const Review = require('../models/reviewModel');
const AppError = require('../utils/appError');
const catchAsync = require('../utils/catchAsync');
const factory = require('../utils/handlerFactory');

exports.setModelUserIds = (req, res, next) => {
   if (req.params.tourId) {
```

```
      if (!req.body.tour) req.body.tour = req.params.tourId
      req.body.modelReview = "tour";
   }
   if (req.params.hotelId) {
      if (!req.body.hotel) req.body.hotel = req.params.hotelId
      req.body.modelReview = "hotel";
   }
   if (req.params.restaurantId) {
      if (!req.body.restaurant) req.body.restaurant =
req.params.restaurantId
      req.body.modelReview = "restaurant";
   }
   if (!req.body.user) req.body.user = req.user.id

   next();
}

exports.isReviewExisted = catchAsync(async (req, res, next) => {
   let existed = undefined;
   switch (req.body.modelReview) {
      case "tour":
         existed = await Review.findOne({ user: req.body.user, tour:
req.body.tour })
         break;
      case "hotel":
         existed = await Review.findOne({ user: req.body.user, tour:
req.body.hotel })
         break;
      case "restaurant":
         existed = await Review.findOne({ user: req.body.user,
restaurant: req.body.restaurant })
         break;
      default:
         existed = null;
         break;
   }
   req.body.modelReview = undefined
   if (existed) return next(new AppError("You've already added
review.", 400))
   next()

})

exports.getReviewStats = catchAsync(async (req, res,next) => {
```

```
    const stats = await Review.aggregate([
        {
          $group: {
             _id: null,
             totalReviews: { $sum: 1 },
             averageRating: { $avg: "$rating" },
             minRating: { $min: "$rating" },
             maxRating: { $max: "$rating" }
           }
        },
        {
            $sort: {
                avgPrice: 1
            }
        }
    ]);
    res.status(200).json({
        status: 'success',
        data: {stats}
    })
})

exports.getAllReviews = factory.getAll(Review)

exports.getReview = factory.getOne(Review)

exports.createReview = factory.createOne(Review)

exports.updateReview = factory.updateOne(Review)

exports.deleteReview = factory.deleteOne(Review)

exports.deleteAllReviews = factory.deleteAll(Review)
```

```
const multer = require('multer');
const sharp = require('sharp');
const Tour = require('./../models/tourModel');
const catchAsync = require('./../utils/catchAsync');
const factory = require('../utils/handlerFactory');
const AppError = require('./../utils/appError');
const cloudinary = require('./../utils/cloudinary');
```

```
exports.aliasTopTours = (req, res, next) => {
    req.query.limit = '4';
    req.query.sort = '-ratingsAverage,price';
    req.query.fields =
'name,price,photo,ratingsAverage,ratingQuantity,slug,summary';
    next();
}

exports.getAllTours = factory.getAll(Tour)

exports.getTourById = factory.getOne(Tour, {path:'reviews'})

exports.createTour = factory.createOne(Tour)

exports.updateTourById = factory.updateOne(Tour)

exports.deleteTour = factory.deleteOne(Tour)

exports.getTourStats = catchAsync(async (req, res,next) => {
    const stats = await Tour.aggregate([
        {
            $match: {
                ratingsAverage: {$gte: 0}
            }
        },
        {
            $group: {
                _id:{ $toUpper: '$difficulty' },
                num: {$sum: 1},
                numRatings: {$sum: '$ratingQuantity'},
                avgRating: {$avg: '$ratingsAverage'},
                avgPrice: {$avg: '$price'},
                minPrice: {$min: '$price'},
                maxPrice: {$max: '$price'}
            }
        },
        {
            $sort: {
                avgPrice: 1
            }
        }
    ]);
    res.status(200).json({
        status: 'success',
```

```
        data: {stats}
    })
})

exports.getMonthlyPlan = catchAsync(async (req,res,next) =>{
    const year = req.params.year * 1;
    const plan =  await Tour.aggregate([
        {
            $unwind: '$startDates'
        },
        {
            $match: {
                startDates: {
                    $gte: new Date(`${year}-01-01`),
                    $lte: new Date(`${year}-12-31`),
                }
            }
        },
        {
            $group: {
                _id: {$month: '$startDates'},
                numTourStats :{ $sum : 1 },
                tours:{$push: '$name'}
            }
        },
        {
            $addFields: { month: '$_id'}
        },
        {
            $project : {_id:0}
        },
        {
            $sort : {numberTourStarts: -1}
        },
        {
            $limit:12
        }
    ]);

    res.status(200).json({
        status: 'success',
        result: plan.length,
        data: {plan}
    })
```

```
})

exports.getToursWithin = catchAsync(async (req, res, next) => {
    const { distance, latlng, unit } = req.params;
    const [lat, lng] = latlng.split(',');

    const radius = unit === 'mi' ? distance / 3963.2 : distance /
6378.1;

    if (!lat || !lng) {
        next(new AppError('Please provide latitutr and longitude in
the format lat,lng.', 400));
    }

    const tours = await Tour.find({
        startLocation: { $geoWithin: { $centerSphere: [[lng, lat],
radius] } }
    });

    res.status(200).json({
        status: 'success',
        results: tours.length,
        data: {
            data: tours
        }
    });
});

exports.getDistances = catchAsync(async (req, res, next) => {
    const { latlng, unit } = req.params;
    const [lat, lng] = latlng.split(',');

    const multiplier = unit === 'mi' ? 0.000621371 : 0.001;

    if (!lat || !lng) {
        next(
            new AppError(
                'Please provide latitude and longitude in the format
lat,lng.',
                400
            )
        );
    }
```

```
    const distances = await Tour.aggregate([
        {
            $geoNear: {
                near: {
                    type: 'Point',
                    coordinates: [lng * 1, lat * 1]
                },
                distanceField: 'distance',
                distanceMultiplier: multiplier
            }
        },
        {
            $project: {
                distance: 1,
                name: 1
            }
        }
    ]);

    res.status(200).json({
        status: 'success',
        data: {
            data: distances
        }
    });
});

exports.updateAllStartDates = catchAsync(async (req,res,next) => {
    const tours = await Tour.find({})

    tours.forEach(async (tour) => {
        tour.startDates.forEach(date => {
            date.availablePlaces = tour.maxGroupSize;
        })
       await tour.save()
    })



    res.status(200).json({
        status: 'success',
        message: 'Start dates was successfully updated.'
    });
})
```

```
const uploadTourPhoto = publicId => (req, res, next) => {
    cloudinary.createMulti('photo', 'Tours', publicId, 900, 700)(req,
res, (err) => {
       if (err) return next(err);
       if (req.files && req.files.length>0) {
          req.body.photo = req.files.map(el=>el.path);
       }
       next();
    })
 };

 const deleteTourPhoto = publicId => (req, res, next) => {
    cloudinary.deleteSingle('Tours', publicId);
    next();
 };

 exports.uploadTourPhoto = (req, res, next) => {
    uploadTourPhoto(req.params.id)(req, res, next);
 }

 exports.deleteTourPhoto = (req, res, next) => {
    deleteTourPhoto(req.params.id)(req, res, next);
 }
```

```
const multer = require('multer');
const sharp = require('sharp');
const User = require('./../models/userModel');
const APIFeatures = require('./../utils/APIFeatures');
const catchAsync = require('./../utils/catchAsync');
const AppError = require('./../utils/appError');
const factory = require('./../utils/handlerFactory');
const cloudinary = require('./../utils/cloudinary');


const filterObj = (obj, ...allowedFields) => {
    const newObj = {};
    Object.keys(obj).forEach(el => {
        if (allowedFields.includes(el)) newObj[el] = obj[el];
    });
    return newObj;
};

exports.getMe = (req, res, next) => {
```

```
    req.params.id = req.user.id;
    next();
};

exports.updateMe = catchAsync(async (req, res, next) => {
    if (req.body.password || req.body.passwordConfirm) {
        return next(
            new AppError('This route is not for password updates.
Please use /updateMyPassword.', 400)
        );
    }

    const filteredBody = filterObj(req.body, 'name', 'email');
    if (req.file) filteredBody.photo = req.file.path;

    const updatedUser = await User.findByIdAndUpdate(req.user.id,
filteredBody, {
        new: true,
        runValidators: true
    });

    res.status(200).json({
        status: 'success',
        data: {
            user: updatedUser
        }
    });
});

exports.deleteMe = catchAsync(async (req, res, next) => {
    await User.findByIdAndUpdate(req.user.id, { active: false });
    res.status(204).json({
        status: 'success',
        data: null
    });
});

exports.uploadUserPhoto = (req, res, next) => {
    const upload = cloudinary.createSingle(
        'photo',
        'Users',
        req.user.id,
        500,
        500
```

```
    );

    upload(req, res, err => {
        if (err) return next(err);

        if (req.file) req.body.photo = req.file.path;
        next();
    });
};

exports.deleteUserPhoto =  (req, res, next) => {
    cloudinary.deleteSingle('Users', req.user.id);
    next();
};


exports.getAllUsers = factory.getAll(User)

exports.getUser = factory.getOne(User)

exports.updateUser = factory.updateOne(User)

exports.deleteUser = factory.deleteOne(User)
```

```
const mongoose = require('mongoose');

const bookingSchema = new mongoose.Schema({
  tour: {
    type: mongoose.Schema.ObjectId,
    ref: 'Tour'
  },
  hotel: {
    type: mongoose.Schema.ObjectId,
    ref: 'Hotel'
  },
  restaurant: {
    type: mongoose.Schema.ObjectId,
    ref: 'Restaurant'
  },
  user: {
    type: mongoose.Schema.ObjectId,
    ref: 'User',
    required: [true, 'Booking must belong to a User!']
  },
```

```
  price: {
    type: Number,
    require: [true, 'Booking must have a price.']
  },
  onDate: {
    type:Date,
    require: true
  },
  numberOfPeople: {
    type: Number,
    require: true
  }
}, { timestamps: true });

bookingSchema.pre(/^find/, function (next) {
  if (this.tour) this.populate({
    path: 'tour',
    select: 'name photo'
  });
  else if (this.hotel) this.populate({
    path: 'hotel',
    select: 'name photo'
  });
  else this.populate({
    path: 'restaurant',
    select: 'name photo'
  });
  next();
});

const Booking = mongoose.model('Booking', bookingSchema);

module.exports = Booking;
```

```
const mongoose = require('mongoose')
const slugify = require('slugify');

const destinationSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true, "A destination must have a name"],
        unique: true,
        trim: true,
```

```
        maxLength: [40, "The destination name must have at most 40
characters"],
        minLength: [4, "The destination name must have at least 10
characters"]
    },
    slug: String,
    description: {
        type: String,
        trim: true,
        required: true,
        minLength: [30, "The destination description must have at
least 30 characters"]
    },
    short_desc: {
        type: String,
        trim: true,
        required: true,
        minLength: [10, "The destination short description must have
at least 30 characters"]
    },
    photo: [
        {
            type: String,
            default: "https://icon-library.com/images/anonymous-
avatar-icon/anonymous-avatar-icon-25.jpg",
        }
    ],
    location: {
        type: {
            type: String,
            default: 'Point',
            enum: ['Point'],
        },
        coordinates: [Number],
    }
}, {
    timestamps: true,
    toJSON: { virtuals: true },
    toObject: { virtuals: true }
});

destinationSchema.pre('save', function (next) {
    this.slug = slugify(this.name, { lower: true });
    next();
```

```
});

destinationSchema.virtual('tours', {
    ref: 'Tour',
    foreignField: 'destination',
    localField: 'name'
});

destinationSchema.virtual('hotels', {
    ref: 'Hotel',
    foreignField: 'destination',
    localField: 'name'
});

destinationSchema.virtual('restaurants', {
    ref: 'Restaurant',
    foreignField: 'destination',
    localField: 'name'
});

const Destination = mongoose.model('Destination', destinationSchema)

module.exports = Destination;
```

```
const mongoose = require('mongoose');
const Tour = require('./tourModel');


const reviewSchema = new mongoose.Schema(
    {
        review: {
            type: String,
            required: [true, 'Review can not be empty!']
        },
        rating: {
            type: Number,
            min: 1,
            max: 5
        },
        tour: {
            type: mongoose.Schema.ObjectId,
            ref: 'Tour',
        },
```

```
        hotel: {
            type: mongoose.Schema.ObjectId,
            ref: 'Hotel',
        },
        restaurant: {
            type: mongoose.Schema.ObjectId,
            ref: 'Restaurant',
        },
        user: {
            type: mongoose.Schema.ObjectId,
            ref: 'User',
            required: [true, 'Review must belong to a user']
        }
    },
    {
        timestamps: true,
        toJSON: { virtuals: true },
        toObject: { virtuals: true }
    }
);

reviewSchema.pre(/^find/, function (next) {
    this.populate({
        path: 'user',
        select: 'name photo'
    });
    next();
})

reviewSchema.statics.calcAverageRatings = async function (tourId) {
    const stats = await this.aggregate([
        {
            $match: { tour: tourId }
        },
        {
            $group: {
                _id: '$tour',
                nRating: { $sum: 1 },
                avgRating: { $avg: '$rating' }
            }
        }
    ]);

    if (stats.length > 0) {
```

```
        await Tour.findByIdAndUpdate(tourId, {
            ratingQuantity: stats[0].nRating,
            ratingsAverage: stats[0].avgRating
        });
    } else {
        await Tour.findByIdAndUpdate(tourId, {
            ratingQuantity: 0,
            ratingsAverage: 4
        });
    }
};

reviewSchema.post('save', function () {
    this.constructor.calcAverageRatings(this.tour);
});

reviewSchema.post(/^findOneAnd/, function (doc, next) {
    doc.constructor.calcAverageRatings(doc.tour);
    next();
});


reviewSchema.pre('deleteMany', async function (next) {
    try {
        await Tour.updateMany({}, { ratingQuantity: 0, ratingsAverage:
4 })
        next();
    }
    catch (err) {
        console.log(err)
    }
});
```

```
const Review = mongoose.model('Review', reviewSchema);

module.exports = Review;
```

```
const mongoose = require('mongoose')
const slugify = require('slugify');
const validator = require('validator');
const User = require('./userModel');

const tourSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true, "A tour must have a name"],
        unique: true,
        trim: true,
        maxLength: [40, "The tour name must have at most 40
characters"],
        minLength: [10, "The tour name must have at least 10
characters"]
    },
    slug: String,
    duration: {
        type: Number,
        required: [true, 'The tour must have a duration']
    },
    maxGroupSize: {
        type: Number,
        required: [true, 'The tour must have a group size']
    },
    difficulty: {
        type: String,
        required: [true, 'The tour must have a difficulty'],
        enum: {
            values: ['easy', 'medium', 'difficult'],
            message: 'The tour must have a valid difficulty'
        },
    },
    ratingsAverage: {
        type: Number,
```

```
        default: 4,
        min: [1, "The rating must be above 1.0"],
        max: [5, "The rating must be below 5.0"],
        set: val => Math.round(val * 10) / 10
    },
    ratingQuantity: {
        type: Number,
        default: 0
    },
    price: {
        type: Number,
        required: [true, "A tour must have a price"]
    },
    priceDiscount: {
        type: Number,
        validate: {
            validator: function (value) {
                return value < this.price;
            },
            message: "The discount price must lower the the regular
price"
        }
    },
    summary: {
        type: String,
        trim: true,
        required: [true, "A tour must have a summary"]
    },
    description: {
        type: String,
        trim: true,
        required: [true, "A tour must have a description"]
    },
    photo: [String],
    startDates: [
        {
            date: {
                type: Date
            },
            availablePlaces: {
                type: Number
            }
        }
    ],
```

```
    secretTour: {
        type: Boolean,
        default: false
    },
    startLocation: {
        type: {
            type: String,
            default: 'Point',
            enum: ['Point']
        },
        coordinates: [Number],
        address: String,
        description: String
    },
    locations: [
        {
            type: {
                type: String,
                default: 'Point',
                enum: ['Point']
            },
            coordinates: [Number],
            address: String,
            description: String,
            hours: Number
        }
    ],
    destination: {
        type: String,
        required: true
    },
    guides: [
        {
            type: mongoose.Schema.ObjectId,
            ref: 'User'
        }
    ]
}, {
    toJSON: { virtuals: true },
    toObject: { virtuals: true },
    timestamps: true
});

tourSchema.index({ price: 1, ratingsAverage: -1 });
```

```
tourSchema.index({ slug: 1 });
tourSchema.index({ startLocation: '2dsphere' });

tourSchema.virtual('durationWeeks').get(function () {
    return this.duration / 7;
});

tourSchema.virtual('reviews', {
    ref: 'Review',
    foreignField: 'tour',
    localField: '_id'
});


tourSchema.pre('save', function (next) {
    this.slug = slugify(this.name, { lower: true });
    next();
});

tourSchema.pre(/^find/, function (next) {
    this.populate({
        path: 'guides',
        select: '-__v -passwordChangedAt'
    });

    next();
});

tourSchema.pre(/^find/, function (next) {
    this.find({ secretTour: { $ne: true } })
    this.start = Date.now();
    next();
})

tourSchema.pre('save', function(next) {
    if (this.isNew) {
        this.startDates.forEach(date => {
            date.availablePlaces = this.maxGroupSize;
        });
    }
    next();
});
```

```
const Tour = mongoose.model('Tour', tourSchema)

module.exports = Tour;
```

```
const crypto = require('crypto');
const mongoose = require('mongoose');
const validator = require('validator');
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true, 'Please tell us your name']
    },
    email: {
        type: String,
        required: [true, 'Please provide your email address'],
        unique: true,
        lowercase: true,
        validate: [validator.isEmail, 'Please provide a valid email
address']
    },
    password: {
        type: String,
        required: [true, 'Please provide your password'],
        minlength: 8,
        select: false
    },
    passwordConfirm: {
        type: String,
        required: [true, 'Please confirm your password'],
        validate: {
            validator: function (el) {
                return el === this.password
            },
            message: "Passwords are not the same!"
        }
    },
    photo:{
        type:String,
```

```
        default:"https://icon-library.com/images/anonymous-avatar-
icon/anonymous-avatar-icon-25.jpg",
    },
    role: {
        type: String,
        enum: ['user', 'guide', 'lead-guide', 'admin'],
        default: 'user'
    },
    token: {
        type:String
    },
    passwordChangedAt: Date,
    passwordResetToken: String,
    passwordResetExpires: Date,
    active: {
        type: Boolean,
        default:true,
        select:false
    }
}, {timestamps:true})


userSchema.pre('save', async function (next) {
    if (!this.isModified('password')) return next();

    this.password = await bcrypt.hash(this.password, 12);
    this.passwordConfirm = undefined;
    next();
})


userSchema.pre('save', function(next) {
    if (!this.isModified('password') || this.isNew) return next();

    this.passwordChangedAt = Date.now() - 1000;
    next();
});

userSchema.pre(/^find/, function(next) {
    // this points to the current query
    this.find({ active: { $ne: false } });
    next();
});
```

```
userSchema.methods.correctPassword = async function
(candidatePassword, userPassword) {
    return await bcrypt.compare(candidatePassword, userPassword);
}

userSchema.methods.changedPasswordAfter = function (JWTtimestamp) {
    if (this.passwordChangedAt) {
        const changedTimestamp =
parseInt(this.passwordChangedAt.getTime() / 1000, 10);
        return JWTtimestamp < changedTimestamp;
    }
    return false;
}
userSchema.methods.createPasswordResetToken = function () {
    const resetToken = crypto.randomBytes(32).toString('hex');

    this.passwordResetToken =
crypto.createHash('sha256').update(resetToken).digest('hex');

    this.passwordResetExpires = Date.now() + 10 * 60 * 1000;
    return resetToken;
}

const User = mongoose.model('User', userSchema)
module.exports = User;
```

```
const express = require("express");
const bookingController =
require('./../controllers/bookingController');
const authMiddleware = require('./../middlewares/authMiddleware');

const router = express.Router({mergeParams:true});

router.use(authMiddleware.protect)

router.get('/booking-stats',authMiddleware.restrictTo('admin', 'lead-
guide'),bookingController.getBookingStats);
router.get('/',authMiddleware.restrictTo('admin', 'lead-guide'),
bookingController.getAllBookings)
router.get('/my', bookingController.getMyBookings)

router.post('/checkout-session', authMiddleware.restrictTo('user'),
bookingController.getCheckoutSession)
```

```
router.post('/', authMiddleware.restrictTo('user'),
bookingController.setModelUserIds, bookingController.createBooking)
router.route('/:id').delete(authMiddleware.restrictTo('user',
'admin'),
bookingController.deleteBooking).patch(authMiddleware.restrictTo('user
', 'admin'),
bookingController.updateBooking).get(bookingController.getBooking)



module.exports = router;
```

```
const express = require('express');
const destinationController =
require('../controllers/destinationController');
const {protect, restrictTo} = require("../middlewares/authMiddleware")

const router = express.Router()

router.get('/', destinationController.getAllDestinations)
router.get('/:slug', destinationController.getDestination)

router.use(protect)
router.use(restrictTo('admin', 'lead-guide'))

router.post('/', destinationController.createDestination)
router.patch('/:id', destinationController.uploadDestPhoto,
destinationController.updateDestination)
router.delete('/:id', destinationController.deleteDestination)

module.exports = router


const express = require("express");
const reviewController = require('./../controllers/reviewController');
const authMiddleware = require('./../middlewares/authMiddleware');

const router = express.Router({ mergeParams: true });

router.route('/')
   .get(reviewController.getAllReviews)
   .post(
      authMiddleware.protect,
      authMiddleware.restrictTo('user'),
```

```
      reviewController.setModelUserIds,
      reviewController.isReviewExisted,
      reviewController.createReview
   )
   .delete(
      authMiddleware.protect,
      authMiddleware.restrictTo('admin'),
      reviewController.deleteAllReviews
   )


router.use(authMiddleware.protect)

router.get('/review-stats',authMiddleware.restrictTo('admin', 'lead-
guide'),reviewController.getReviewStats);


router.route('/:id').delete(authMiddleware.restrictTo('user',
'admin'),
reviewController.deleteReview).patch(authMiddleware.restrictTo('user',
'admin'),
reviewController.updateReview).get(reviewController.getReview)
module.exports = router;
```

```
const express = require('express');
const tourController = require('../controllers/tourController');
const {protect, restrictTo} = require("../middlewares/authMiddleware")
const reviewRouter = require("../routes/reviewRoutes")
const bookingRouter = require("../routes/bookingRoutes")

const router = express.Router()
router.use('/:tourId/reviews', reviewRouter);
router.use('/:tourId/bookings', bookingRouter);


router.route('/top-5-cheap').get(tourController.aliasTopTours,
tourController.getAllTours);
router.route('/tour-stats').get(protect, restrictTo('admin', 'lead-
guide'),tourController.getTourStats);
router.route('/monthly-plan/:year').get(protect, restrictTo('admin',
'lead-guide', 'guide'), tourController.getMonthlyPlan);
router.route('/tours-
within/:distance/center/:latlng/unit/:unit').get(tourController.getTou
rsWithin)
```

```
router.route('/distances/:latlng/unit/:unit').get(tourController.getDi
stances);
router.patch('/updateAllStartDates', protect, restrictTo('admin'),
tourController.updateAllStartDates)
router.route('/').get(tourController.getAllTours).post(protect,
restrictTo('admin', 'lead-guide'), tourController.createTour);
router.get('/:slug',tourController.getTourById)
router.route('/:id').patch(protect, restrictTo('admin', 'lead-guide'),
tourController.uploadTourPhoto,
tourController.updateTourById).delete(protect, restrictTo('admin',
'lead-guide'), tourController.deleteTour);

// router.route('/:id').get(tourController.getTourById).patch(protect,
restrictTo('admin', 'lead-guide'), tourController.uploadTourCover,
tourController.resizeTourImages,
tourController.updateTourById).delete(protect, restrictTo('admin',
'lead-guide'), tourController.deleteTour);


module.exports = router;
```

```
const express = require("express");
const userController = require('./../controllers/userController');
const authMiddleware = require('../middlewares/authMiddleware');
const authController = require('../controllers/authController');


const router = express.Router();


router.post('/signup', authController.signup)
router.post('/login', authController.login)
router.get('/logout', authController.logout)
router.post('/forgotPassword', authController.forgotPassword)
router.patch('/resetPassword/:token', authController.resetPassword)

router.use(authMiddleware.protect)

router.get('/me', userController.getMe, userController.getUser)
router.patch('/updateMyPassword', authController.updatePassword)
router.patch(
    '/updateMe',
    userController.uploadUserPhoto,
```

```
    userController.updateMe
);
router.delete('/deleteMe',authMiddleware.restrictTo('user'),
userController.deleteUserPhoto, userController.deleteMe)

router.use(authMiddleware.restrictTo('admin'))

router.get('/', userController.getAllUsers)
router.route('/:id').get(userController.getUser).patch(userController.
uploadUserPhoto,
userController.updateUser).delete(userController.deleteUserPhoto,
userController.deleteUser)


module.exports = router
```

```
const AppError = require("../utils/appError");
const catchAsync = require("../utils/catchAsync");
const APIFeatures = require("../utils/APIFeatures");




exports.deleteOne = Model => catchAsync(async (req, res, next) => {
    const doc = await Model.findByIdAndDelete(req.params.id)
    if (!doc) {
        next(new AppError('No tour found with that id', 404));
        return;
    }
    res.status(204).json({
        status: 'success',
        data: 'Object deleted successfully'
    })
})


exports.deleteAll = Model => catchAsync(async (req, res, next) => {
    await Model.deleteMany({})
    res.status(204).json({
        status: 'success',
        data: 'All Objects deleted successfully'
    })
})
```

```
exports.updateOne = Model => catchAsync(async (req, res, next) => {
    const doc = await Model.findByIdAndUpdate(req.params.id, req.body,
{
        new: true,
        runValidators: true
    })
    if (!doc) {
        next(new AppError('No document found with that id', 404));
        return;
    }
    res.status(200).json({
        status: 'success',
        data: { doc }
    })
})

exports.createOne = Model =>
    catchAsync(async (req, res, next) => {

        const doc = await Model.create(req.body);

        res.status(201).json({
            status: 'success',
            data: {
                data: doc
            }
        });
    });

exports.getOne = (Model, popOptions) =>
    catchAsync(async (req, res, next) => {
        let query;
        if (req.params.slug) query = Model.findOne({ slug:
req.params.slug });
        else query = Model.findById(req.params.id);
        if (popOptions) query = query.populate(popOptions);
        const doc = await query;
        if (!doc) {
            return next(new AppError('No document found with that ID',
404));
        }

        res.status(200).json({
            status: 'success',
```

```
            data: {
                data: doc
            }
        });
    });

exports.getAll = (Model, popOptions) =>
    catchAsync(async (req, res, next) => {
        let filter = {};
        if (req.params.tourId) filter = { tour: req.params.tourId };
        if (req.params.hotelId) filter = { hotel: req.params.hotelId
};
        if (req.params.restaurantId) filter = { restaurant:
req.params.restaurantId };

        const features = new APIFeatures(Model.find(filter),
req.query)
            .filter()
            .sort()
            .search()
            .limitFields()
            .paginate();
        if (popOptions) features.query =
features.query.populate(popOptions)
        const doc = await features.query;

        res.status(200).json({
            status: 'success',
            results: doc.length,
            data: {
                data: doc
            }
        });
    });
```

```
const multer = require('multer');
const cloudinary = require('cloudinary').v2;
const { CloudinaryStorage } = require('multer-storage-cloudinary');
const AppError = require('./appError');
```

```
cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

const fileFilter = (req, file, cb) => {
  if (file.mimetype.startsWith('image')) {
    cb(null, true);
  } else {
    cb(new AppError('Not an image! Please upload only images.', 400),
false);
  }
};

exports.createMulti = (keys, folder, public_id, width, height) => {
  let counter = 1;
  const storage = new CloudinaryStorage({
    cloudinary,
    params: {
      folder: `TourScape/${folder}/${public_id}`,
      public_id: (req, file) => `${public_id}_${counter++}`,
      overwrite: true,
      resource_type: 'image',
      crop: "scale",
      width: width,
      height: height,
    },
  });

  return multer({ storage: storage, fileFilter: fileFilter }).array(
    keys
  );
};

exports.createSingle = (key, folder, public_id, width, height) => {
  const storage = new CloudinaryStorage({
    cloudinary,
    params: {
      folder: `TourScape/${folder}`,
      public_id: () => public_id,
      overwrite: true,
      resource_type: 'image',
      crop: "scale",
```

```
      width: width,
      height: height,
    },
  });
  return multer({ storage: storage, fileFilter: fileFilter }).single(
    key
  );
}

exports.deleteSingle = (folder, public_id) => {
  cloudinary.uploader.destroy(`TourScape/${folder}/${public_id}`);
};
```

```
module.exports = fn => {
    return (req,res,next) => {
        fn(req,res,next).catch(next);
    }
}
```