|  |  |
| --- | --- |
| C:\Users\Емил Йорданов\Downloads\logo.png | **ВИСШЕ ВОЕННОМОРСКО УЧИЛИЩЕ „Н. Й. ВАПЦАРОВ“**  ***9002 Варна, ул. „В. Друмев“ 73, тел.052/632-015, факс 052/303-163*** |
| ***“FILII MARIS SUMUS”***  **ФАКУЛТЕТ „ИНЖЕНЕРЕН“ – КАТЕДРА „ИНФОРМАЦИОННИ ТЕХНОЛОГИИ“** |
| A picture containing text  Description automatically generated |  |

**ПРОЕКТ**

**по дисциплината „Програмиране на Интернет приложения“**

**теМА: „Планиране и разработка на платформа за**

**организиране на пътувания“**

**Студент: Доника Иванова Йорданова**

**Специалност: “Киберсигурност”**

**фак. № 126-22107**

**гр. Варна**

**2025 г.**

Съдържание

[1 Увод 2](#_Toc198276622)

[2 Изложение 6](#_Toc198276623)

[2.1 Литературен обзор 6](#_Toc198276624)

[2.1.1 Анализ на съществуващото положение в областта на уебсайтовете за планиране на пътувания 6](#_Toc198276625)

[2.2 Аналитична част 10](#_Toc198276626)

[2.2.1 Фаза на планиране 11](#_Toc198276627)

[2.2.1.1 Дефиниране на обхвата и изискванията към проекта 11](#_Toc198276628)

[2.2.1.2 Избор на технологичен стек 13](#_Toc198276629)

[2.2.1.3 Определяне и обосновка на основните функционалности 16](#_Toc198276630)

[2.2.1.4 Проектиране на потребителския интерфейс (UI) и потребителското изживяване (UX) – Дизайн с Figma 19](#_Toc198276631)

[2.2.1.5 Планиране на архитектурата на приложението 21](#_Toc198276632)

[2.2.2 Фаза на разработка 21](#_Toc198276633)

[2.2.2.1 Проектиране и имплементация на базата данни (MySQL) 22](#_Toc198276634)

[2.2.2.2 Разработка на Backend приложението (Node.js и Express.js) 33](#_Toc198276635)

[2.2.2.3 Разработка на Frontend приложението (React, Vite, TypeScript, Shadcn-UI) 42](#_Toc198276636)

[3 Заключение 50](#_Toc198276637)

# Увод

В съвременния глобализиран и динамичен свят, пътуванията са се превърнали от лукс в неразделна част от живота на милиони хора. Те обогатяват културно, разширяват хоризонтите, предоставят възможности за отдих, приключения и бизнес развитие. Въпреки това, процесът на планиране на едно пътуване, особено когато става въпрос за непознати дестинации или комплексни маршрути, често остава значително предизвикателство. Потенциалните пътешественици са изправени пред необходимостта да навигират в огромен океан от информация, разпръсната из множество уебсайтове, платформи и мобилни приложения. Търсенето на най-подходящите полети, изборът на удобно и достъпно настаняване, идентифицирането на интересни забележителности и добри места за хранене, както и запознаването с особеностите на дестинацията, изискват значителни времеви ресурси, усилия и аналитични умения. Тази фрагментираност не само утежнява процеса, но може да доведе до информационно претоварване, стрес, пропуснати възможности и в крайна сметка – до неоптимално планирано пътуване.

Необходимостта от разработване на интегрирани и интелигентни решения за планиране на пътувания е по-осезаема от всякога. Съвременните потребители, свикнали с удобството и ефективността на цифровите технологии, очакват интуитивни, бързи и персонализирани услуги. Те търсят платформи, които могат да предложат цялостно преживяване "от край до край" – от първоначалното вдъхновение и проучване, през детайлното планиране и резервация, до подкрепа по време на самото пътуване. Актуалността на този проблем се подчертава от непрекъснато нарастващия обем на онлайн туристическия пазар и засилващата се конкуренция, която стимулира иновациите и търсенето на по-добри потребителски решения. В областта на информационните технологии и софтуерното инженерство, това представлява благодатна почва за създаване на стойностни продукти, които решават реални проблеми и подобряват качеството на живот.

Развитието на уеб технологиите, включително мощни JavaScript рамки за frontend като React.js, гъвкави backend платформи като Node.js, надеждни системи за управление на бази данни като MySQL, както и широкото разпространение на приложни програмни интерфейси (API), предоставят необходимите инструменти за изграждането на сложни и високо функционални уебсайтове. Тези технологии позволяват създаването на динамични, интерактивни и отзивчиви потребителски интерфейси, ефективна обработка на данни в реално време и безпроблемна интеграция на услуги от трети страни. Проектирането на система, която интелигентно агрегира и представя информация за полети, хотели, атракции и заведения, като същевременно предлага допълнителни ресурси като информация за държави и готови маршрути, е задача, която напълно съответства на съвременните тенденции в разработката на уеб приложения. Този курсов проект има за цел да адресира именно тази необходимост, като предложи концепция и разработка на прототип на уебсайт, който да служи като централизиран хъб за планиране на пътувания.

Освен чисто практическата полза за крайните потребители, разработката на такъв проект има и значителна образователна стойност. Тя обхваща широк спектър от компетенции в областта на софтуерното инженерство – от анализ на изискванията и проектиране на софтуерна архитектура, през имплементация на клиентска и сървърна логика и работа с бази данни, до тестване и управление на проекта. Прилагането на съвременни технологии като React.js за изграждане на потребителски интерфейс с компоненти от библиотеката Shadcn-UI, Node.js за разработка на API и MySQL за управление на данните, осигурява придобиването на практически умения с инструменти, които са широко търсени в индустрията. Проектът също така дава възможност за изследване на принципите на потребителското изживяване (UX) и потребителския интерфейс (UI), които са критични за успеха на всяко съвременно уеб приложение.

* В този контекст се формулират следните основни компоненти на изследването:
* **Обект на изследването:** Обект на настоящия курсов проект е уебсайт за цялостно и интегрирано планиране на пътувания, предоставящ персонализирана информация и инструменти за организация на пътешествия.
* **Предмет на изследването:** Предметът на изследването обхваща процеса на анализ на изискванията, проектиране на софтуерната архитектура, разработване на функционалните модули (включващи backend логика с Node.js и frontend интерфейс с React.js и Shadcn-UI), дизайн и имплементация на релационна база данни с MySQL, както и последващо тестване на разработения прототип. Фокусът е върху качествените характеристики на системата като функционалност, използваемост, ефективност на търсенето, интеграция на различните компоненти на пътуването (полети, хотели, атракции, заведения) и представяне на допълнителна полезна информация (данни за държави, готови туристически пакети).
* **Теза, която се защитава:** Създаването на централизиран, интуитивен и многофункционален уебсайт за планиране на пътувания, който използва съвременни уеб технологии (Node.js, React.js, MySQL, Shadcn-UI), ще предостави на потребителите значително по-ефективен, удобен и персонализиран инструмент за организиране на техните пътешествия. Подобна платформа ще намали времето и усилията, необходими за проучване и планиране, ще минимизира стреса, свързан с този процес, и ще допринесе за по-информиран избор и по-добро цялостно преживяване преди и по време на пътуването, като същевременно обединява цялата необходима информация на едно място.
* **Цел:** Основната цел на този курсов проект е да се проектира, разработи и тества функционален прототип на уебсайт за планиране на пътувания. Системата трябва да позволява на потребителите да въвеждат своите критерии за търсене (дестинация, град на тръгване, начална и крайна дата, брой пътници) и въз основа на тях да получават персонализиран списък с подходящи оферти за самолетни билети, хотели, забележителности и заведения. Допълнително, уебсайтът трябва да предоставя възможност за четене на информация за определена държава и разглеждане на готови, предварително дефинирани пътувания, като накрая показва обобщен преглед на всички избрани от потребителя места и полети.
* **Задачи, които си поставя студентът при решаване на избрания проблем:**

1. Да се извърши задълбочен анализ на съществуващи онлайн платформи и приложения за планиране на пътувания, като се идентифицират техните основни функционалности, потребителски интерфейси, силни и слаби страни, с цел извличане на добри практики и дефиниране на конкурентни предимства за разработвания проект.
2. Да се дефинират детайлно функционалните и нефункционалните изисквания към разработвания уебсайт, включително потребителските сценарии, обхвата на предоставяната информация и очакваното потребителско изживяване.
3. Да се обоснове изборът на конкретни технологии за разработка: JavaScript (Node.js) за backend, React.js с библиотеката от компоненти Shadcn-UI за frontend, и MySQL за система за управление на бази данни.
4. Да се проектира цялостната софтуерна архитектура на уебсайта, включително дефиниране на структурата на базата данни, проектиране на RESTful API за комуникация между frontend и backend, и определяне на основните компоненти на потребителския интерфейс.
5. Да се разработят основните функционални модули на системата, съгласно дефинираните изисквания: модул за търсене на пътувания; модули за извличане и показване на информация за самолетни билети, настаняване (хотели, хостели, апартаменти), забележителности и заведения; модул за предоставяне на информация за държави; модул за представяне на готови пътувания; и модул за обобщаване на избрания от потребителя план.
6. Да се имплементира интуитивен, естетически издържан и отзивчив потребителски интерфейс, съобразен с визуалната концепция на проекта и съвременните практики за UI/UX дизайн.
7. Да се подготви и структурира цялата необходима документация към курсовия проект, отразяваща извършената работа и постигнатите резултати, съгласно зададените академични изисквания.

# Изложение

В настоящата глава се разглежда теоретичната основа и контекстът на разработвания курсов проект. Изложението включва литературен обзор, който анализира съществуващите решения в областта на уебсайтовете за планиране на пътувания и представя технологичната рамка на проекта. Следва аналитична част, в която детайлно се описва проектираната и разработена система.

## Литературен обзор

Литературният обзор има за цел да позиционира настоящия проект спрямо съществуващото състояние на изследвания обект, да идентифицира ключови проблеми и възможности, и да положи теоретичните основи за предложеното решение. Той включва анализ на конкурентни платформи, изследване на същността на проблема с планирането на пътувания и преглед на използваните технологии.

### Анализ на съществуващото положение в областта на уебсайтовете за планиране на пътувания

Както беше изтъкнато в увода, процесът на планиране на пътувания в дигиталната ера е белязан от парадокса на изобилието – наличието на огромно количество информация и инструменти често води до фрагментираност и затруднения за потребителя. Редица утвърдени платформи доминират пазара, всяка със своите силни страни и специфичен фокус. Въпреки техните безспорни предимства, анализът им разкрива и определени пропуски или области, в които потребителското изживяване може да бъде значително подобрено – именно тук се позиционира и настоящият проект.

* **Booking.com:** Първоначално концентриран изключително върху хотелски резервации, Booking.com разшири обхвата си, включвайки полети, коли под наем и туристически атракции. Основното му предимство остава обширната база данни от места за настаняване, придружена от милиони потребителски ревюта. **Проблем, който не решава напълно:** Въпреки добавянето на полети и атракции, платформата е силно ориентирана към индивидуални резервации, предимно на настаняване. Процесът на *цялостно, детайлно планиране*, при което потребителят последователно избира полети за конкретни дати, хотел, специфични атракции за посещение и дори ресторанти, и след това вижда всичко това обединено в единен, лесно прегледен *персонализиран маршрут в рамките на платформата*, често не е основният фокус. Потребителят обикновено трябва сам да "сглобява" тези елементи извън платформата или да разчита на отделни резервации. Настоящият проект цели да предложи именно такава интегрирана среда за планиране и финално обобщение на всички избрани компоненти.
* **Expedia.com (и свързаните с нея брандове):** Тази група компании предлага широк спектър от туристически услуги и е силна във възможността за комплексни резервации, често с отстъпки при комбиниране на услуги (пакети). **Проблем, който не решава напълно:** Макар да предлагат пакети, те често са предварително дефинирани или с ограничени възможности за финна настройка на всеки отделен компонент (конкретни атракции, ресторанти). Гъвкавостта потребителят *сам* да изгради своя уникален план стъпка по стъпка, избирайки всеки елемент индивидуално, и след това да получи ясен, консолидиран преглед на *своя* избор, може да бъде подобрена. Нашият проект акцентира върху предоставянето на пълен контрол на потребителя върху избора на всеки елемент от пътуването и последващото му ясно обобщение**.**
* **TripAdvisor.com:** Уникален със своя акцент върху потребителски генерирано съдържание (ревюта, снимки, форуми). Платформата е отличен ресурс за проучване на отделни компоненти на пътуването. **Проблем, който не решава напълно:** Основната сила на TripAdvisor е в предоставянето на информация и мнения, а не толкова в активното изграждане и управление на цялостен, структуриран маршрут. Макар да има инструменти за запазване на места, консолидирането на цялостен план, включващ полети, настаняване, подробен списък с атракции (с работно време, цени) и ресторанти за конкретни дати, в единен и лесен за преглед формат, не е централната му функция. Нашият проект цели да интегрира детайлната информация (като работно време и цени за атракции, средни цени за ресторанти) директно в процеса на планиране и в крайния обобщен план.
* **Kayak.com, Momondo.com, Skyscanner.net:** Тези платформи функционират като мета-търсачки, агрегиращи и сравняващи оферти от множество други сайтове. **Проблем, който не решават напълно:** Тяхната основна цел е сравнението на цени и пренасочването към доставчици за извършване на резервация, най-вече за полети и хотели. Те не предлагат инструменти за детайлно планиране на дейности на място (атракции, ресторанти) или за създаване на обобщен, многокомпонентен маршрут в рамките на самата платформа. Функционалности като "информация за определена държава" или "готови пътувания" за вдъхновение обикновено са извън техния обхват.
* **Google Travel (Google Flights, Google Hotels, Google Things to do):** Предлага силна интеграция и бързо търсене. **Проблем, който не решава напълно:** Въпреки широките си възможности, понякога потребителите могат да се почувстват претоварени от обема информация или да търсят по-фокусирано и изчистено изживяване. Представянето на финалния, консолидиран план от *всички* избрани от потребителя компоненти (включително специфични ресторанти и атракции за всеки ден) по изключително ясен и потребителски ориентиран начин, както и лесната навигация към подробна информация за всяка държава или готов маршрут, са аспекти, където нашият проект може да предложи по-специализирано и оптимизирано решение. Целта е не просто да се агрегира информация, а да се курира и представи по начин, който максимално улеснява потребителя.

В допълнение към тези специфични пропуски, много от съществуващите платформи могат да страдат от:

* **Фрагментирано потребителско изживяване:** Дори когато предлагат различни услуги, преходът между тях не винаги е плавен и интуитивен.
* **Прекомерна комерсиализация на интерфейса:** Постоянни изскачащи прозорци, реклами и техники за предизвикване на спешност (напр. "остава само 1 стая") могат да влошат потребителското изживяване.
* **Недостатъчна дълбочина на информацията за по-малки или специфични дестинации/атракции:** Често фокусът е върху най-популярните места.

Настоящият проект се стреми да адресира тези предизвикателства, като предлага **единна, интуитивна и информационно наситена платформа**. Ключови аспекти, които го отличават, са:

1. **Цялостно планиране на едно място:** От избора на полет и хотел до детайлно планиране на посещения на атракции и избор на ресторанти за конкретни дати.
2. **Ясен и обобщен финален план:** "Накрая му излизат всички избрани места и полети" – тази функционалност е централна за осигуряване на прегледност и контрол от страна на потребителя.
3. **Интегрирана детайлна информация:** Лесен достъп до "info" за всеки хотел и атракция (включително работно време, цени) в контекста на планирането.
4. **Допълнителни ресурси:** Възможност за четене на "информация за определена държава" и разглеждане на "готови пътувания" за вдъхновение и улеснение.
5. **Фокус върху потребителското изживяване:** Чрез използването на React.js и Shadcn-UI се цели създаването на модерен, изчистен и лесен за използване интерфейс, който да направи процеса на планиране по-приятен и по-малко стресиращ.

Анализът на съществуващите платформи показва, че докато всяка от тях предлага значителна стойност в определен аспект на планирането, все още съществува ясно дефинирана ниша за решение, което да обединява по-дълбоко различните компоненти на пътуването в единен, интуитивен и лесен за управление план, предлагайки същевременно богата контекстуална информация и модерен потребителски интерфейс.

## Аналитична част

Аналитичната част на настоящия курсов проект е посветена на детайлното описание на процеса по създаване на уебсайта за планиране на пътувания. Тя е структурирана в две основни фази: **планиране** и **разработка**. В тази първа фаза – планирането – ще бъдат разгледани всички предварителни стъпки и решения, които формират основата на последващата имплементация. Това включва дефиниране на обхвата и изискванията, избор и обосновка на технологичния стек, определяне на ключовите функционалности и тяхното значение за потребителското изживяване, проектиране на потребителския интерфейс с помощта на Figma, както и първоначално планиране на архитектурата на приложението, концептуалния дизайн на базата данни и тестовия процес.

### Фаза на планиране

Успешната реализация на всеки софтуерен проект започва със задълбочена и добре обмислена фаза на планиране. Тя позволява да се дефинират ясно целите, да се изберат най-подходящите инструменти и подходи, и да се минимизират рисковете по време на разработката.

#### Дефиниране на обхвата и изискванията към проекта

Основната цел на проекта е създаването на уеб-базирана платформа, която да служи като централизиран инструмент за планиране на пътувания. Платформата трябва да предоставя на потребителите възможност за търсене и избор на полети, настаняване, забележителности и заведения, както и да предлага допълнителна информация за дестинации и готови маршрути, като накрая обединява всички избори в единен план.

**Функционални изисквания:** Системата трябва да позволява на потребителите да извършват следните действия:

* Търсене на пътувания: Въвеждане на град на тръгване (опционално), дестинация, начална и крайна дата на пътуването, брой пътници.
* Преглед и избор на полети: Показване на списък с налични полети въз основа на критериите за търсене, с възможност за филтриране и преглед на детайли (авиокомпания, часове, цена, прекачвания).
* Преглед и избор на настаняване: Показване на списък с хотели, хостели и апартаменти в избраната дестинация за съответните дати, с опции за филтриране (цена, рейтинг, тип), преглед на снимки, оценки и детайлна информация.
* Преглед и избор на забележителности: Показване на списък с популярни туристически обекти в дестинацията, с кратко описание, работно време, цени на билети и препоръки.
* Преглед и избор на заведения: Показване на списък с препоръчани ресторанти, кафенета и барове, със средна цена, оценки и тип кухня.
* Създаване на персонализиран план: Възможност за "добавяне" на избрани полети, настаняване, забележителности и заведения към текущ план.
* Преглед на обобщен план: Показване на всички избрани от потребителя елементи на пътуването в единен, структуриран преглед.
* Разглеждане на информация за държави: Достъп до информационни страници за различни държави с общо описание, интересни факти, основни градове и др.
* Разглеждане на готови пътувания: Предлагане на предварително дефинирани примерни маршрути и планове за вдъхновение.

**Нефункционални изисквания:**

* Използваемост: Интуитивен, лесен за навигация и разбиране потребителски интерфейс.
* Производителност: Бързо зареждане на страниците и отговор на потребителските действия (в рамките на разумното за курсов проект с примерни данни или ограничени API заявки).
* Отзивчивост на дизайна (Responsive Design): Коректно визуализиране и функциониране на уебсайта на различни размери на екрана (десктоп, таблет, мобилно устройство).
* Надеждност: Стабилна работа на основните функционалности.
* Поддръжка: Добре структуриран и коментиран код за улеснение на евентуални бъдещи модификации.

Целеви потребители: Проектът е насочен към индивидуални пътешественици, които активно участват в планирането на своите пътувания и търсят удобен инструмент, който да им помогне да организират цялата необходима информация на едно място. Това включва както хора, които знаят точно какво искат, така и такива, които търсят вдъхновение и идеи.

#### Избор на технологичен стек

Изборът на подходящ технологичен стек е критичен фактор за успешното изпълнение на проекта, като влияе върху скоростта на разработка, производителността на приложението, възможностите за мащабиране и поддръжка. За настоящия проект е избран модерен full-stack JavaScript подход, допълнен с TypeScript за по-голяма надеждност и MySQL за управление на данните.

* **Backend:**
* **Node.js:** Избрах го заради своята ефективност при обработка на I/O операции, което го прави изключително подходящ за изграждане на API-та и уеб сървъри. Неговият асинхронен, събитийно-ориентиран модел позволява обработката на множество едновременни заявки без блокиране, което е важно за отзивчивостта на приложението. Обширната екосистема от пакети (NPM) предоставя готови решения за множество често срещани задачи, ускорявайки разработката. Използването на JavaScript и на сървърната страна допринася за консистентност в целия проект.
* **Express.js:** Като най-популярната и широко използвана минималистична уеб рамка за Node.js, Express.js е естествен избор. Тя улеснява значително създаването на RESTful API-та чрез интуитивни механизми за маршрутизация (routing), обработка на HTTP заявки и отговори, и лесна интеграция на middleware функции за задачи като валидация, автентикация и логване. Нейната гъвкавост позволява на разработчика да структурира приложението според нуждите си, без да налага строги ограничения.
* **Frontend:**
* **React:** Избрах React заради своята компонентно-базирана архитектура, която позволява изграждането на сложни потребителски интерфейси чрез малки, независими и преизползваеми части. Virtual DOM технологията осигурява висока производителност при обновяване на интерфейса. Голямата общност и богатата екосистема от библиотеки и инструменти правят React предпочитан избор за модерни Single Page Applications (SPA).
* **Vite:** За разработката на frontend частта с React избраx Vite като билд инструмент. Vite предлага изключително бърз Hot Module Replacement (HMR) по време на разработка и оптимизирани продукции билдoве. Конфигурирането му за проекти с React и TypeScript е лесно и интуитивно, което значително подобрява продуктивността на разработчика.
* **TypeScript:** Добавянето на TypeScript към JavaScript проекта носи предимствата на статичното типизиране. Това помага за ранното откриване на грешки (още по време на писане на кода), подобрява четимостта и поддръжката на кода, особено при по-големи проекти, и улеснява съвместната работа. Интелигентните подсказки (intellisense) в съвременните редактори също са значително подобрени.

* **JavaScript:** Остава основният език, тъй като TypeScript се компилира до JavaScript, който се изпълнява в браузърите и в Node.js средата.
* **Shadcn-UI:** Тази "не-библиотека" от компоненти избраx заради модерния си подход и фокуса върху предоставянето на пълен контрол на разработчика. Компонентите се инсталират директно в проекта, което позволява лесна персонализация. Те са изградени върху доказани примитиви за достъпност (Radix UI) и стилизирани с Tailwind CSS, което осигурява гъвкавост и възможност за бързо изграждане на красив и консистентен потребителски интерфейс, съответстващ на визуалната концепция на проекта (както е показано в приложените Figma дизайни).
* **React Query (TanStack Query):** Избрах React Query за управление на сървърното състояние в React. Работата с асинхронни данни (извличане, кеширане, синхронизиране, обновяване) е значително улеснена. React Query намалява нуждата от ръчно управление на състояния за зареждане и грешки, предоставя мощни механизми за кеширане, фоново обновяване на данните и оптимистични ъпдейти, което води до значително по-добро потребителско изживяване.
* **React Router:** Като стандартна библиотека за маршрутизация в React, тя позволява създаването на навигация между различните секции и страници на приложението по интуитивен начин, без да се презарежда цялата страница, което е характерно за SPA.
* База данни:
* **MySQL:** Избрах я като надеждна, широко използвана и добре документирана релационна система за управление на бази данни. Подходяща е за съхранение на структурираната информация на проекта (дестинации, хотели, атракции, потребителски планове и др.). Поддръжката на SQL стандарта и ACID транзакциите гарантира интегритета и консистентността на данните. Наличието на удобни инструменти за администриране (като MySQL Workbench) също е предимство.

**Взаимодействие на технологиите:** Frontend приложението (React, Vite, TypeScript, Shadcn-UI, React Query, React Router) ще комуникира с Backend API-то (Node.js, Express.js) чрез HTTP заявки. Backend-ът ще обработва тези заявки, ще взаимодейства с MySQL базата данни за извличане или запис на информация и ще връща отговори към Frontend-а, който ще визуализира данните на потребителя. TypeScript ще се използва както за Frontend, така и потенциално за Backend разработката за по-голяма надеждност на кода.

#### Определяне и обосновка на основните функционалности

Всяка от дефинираните функционалности е избрана с цел да допринесе за създаването на цялостно, полезно и приятно за използване приложение за планиране на пътувания.

* **Търсене на пътувания (град на тръгване, дестинация, дати, брой пътници):**
  + **Обосновка:** Това е фундаменталната начална точка за всеки процес на планиране. Предоставянето на ясни полета за въвеждане на основните критерии дава на потребителя контрол и позволява на системата да филтрира и предложи релевантна информация.
  + **Потребителско изживяване (UX):** Интуитивната форма за търсене, създава положително първо впечатление и улеснява старта на планирането.
* **Показване на самолетни билети:**
  + **Обосновка:** Транспортът, и по-специално полетите, са ключов и често най-скъп компонент на международните пътувания.
  + **UX:** Възможността за преглед на различни опции и виждането на ясна информация за часовете на излитане и кацане, дали е с прикачване или не, авиокомпанията, както и цената, помага на потребителя да вземе информирано решение.
* **Показване на настаняване (хотели, хостели, апартаменти с рейтенг и снимки):**
  + **Обосновка:** Настаняването е друг критичен елемент, определящ комфорта на пътуването.
  + **UX:** Представянето на обектите със качествени снимки, детайлни описания, снимки на стаите, както и на близките забележителности и ясни цени изгражда доверие и улеснява избора.
* **Показване на забележителности (описание, работно време, официален сайт, най-добро време за посещение):**
  + **Обосновка:** Пътуванията често са мотивирани от желанието за посещение на конкретни атракции или опознаване на културата.
  + **UX:** Предоставянето на изчерпателна и актуална информация директно в платформата спестява на потребителя необходимостта да търси тази информация на други места и улеснява планирането на дневната програма.
* **Показване на заведения (препоръчани ресторанти, кафенета, барове):**
  + **Обосновка:** Храната е важна част от туристическото преживяване.
  + **UX:** Препоръките за заведения, базирани на тип кухня, ценови клас и оценки от други потребители, помагат за откриването на добри места и допринасят за по-приятен престой.
* **Финално обобщение на всички избрани места и полети:**
  + **Обосновка:** Това е една от ключовите функционалности, която отличава проекта. Тя позволява на потребителя да види целия си персонализиран план на едно място.
  + **UX:** Осигурява прегледност, усещане за завършеност и контрол. Улеснява проверката на всички детайли преди евентуални резервации (които са извън обхвата на този прототип, но за които този план би бил основа).
* **Подробна информация (info) за всеки хотел и атракция:**
  + **Обосновка:** Потребителите се нуждаят от достатъчно детайли, за да направят своя избор.
  + **UX:** Лесният достъп до подробна информация (удобства в хотела, точно местоположение, история на забележителността и др.) без необходимост от напускане на платформата подобрява ефективността на планирането.
* **Информация за определена държава:**
  + **Обосновка:** Предлага по-широк контекст за дестинацията, полезен както за обща подготовка, така и за вдъхновение.
  + **UX:** Добавя образователна стойност и помага на потребителите да се почувстват по-подготвени и информирани за културните особености и други важни аспекти на страната, която планират да посетят.
* **Готови пътувания:**
  + **Обосновка:** Не всички потребители имат време или желание да планират всяка подробност от нулата. Готовите маршрути служат като отправна точка или бързо решение.
  + **UX:** Предлагат вдъхновение и улесняват потребителите, които търсят идеи или предпочитат по-структурирани предложения. Могат да бъдат адаптирани или използвани директно.

#### Проектиране на потребителския интерфейс (UI) и потребителското изживяване (UX) – Дизайн с Figma

Качеството на потребителския интерфейс (UI) и цялостното потребителско изживяване (UX) са от първостепенно значение за успеха и възприемането на уебсайт за планиране на пътувания. Тъй като процесът на планиране може да бъде сложен, целта е дизайнът да го направи възможно най-интуитивен, ефективен и приятен.

В рамките на фазата на планиране, инструментът Figma изигра ключова роля за визуализацията и конкретизирането на идеите за UI и UX.

A screenshot of a website

AI-generated content may be incorrect.A screenshot of a website

AI-generated content may be incorrect.фиг 1. Проектиране на Figma дизайн

фиг 2. Проектиране на Figma дизайн

#### Планиране на архитектурата на приложението

За да се осигури гъвкавост, мащабируемост и лесна поддръжка, за уебсайта за планиране на пътувания е избрана класическа **трислойна архитектура**:

1. **Презентационен слой (Presentation Tier – Frontend):** Реализиран с React, Vite, TypeScript и Shadcn-UI. Този слой отговаря за цялото взаимодействие с потребителя, визуализацията на данните и изпращането на потребителски заявки към сървърната част. Работи изцяло в уеб браузъра на клиента.
2. **Приложен слой (Application Tier – Backend):** Реализиран с Node.js и Express.js. Този слой съдържа цялата бизнес логика на приложението. Той приема заявки от frontend-а, обработва ги, взаимодейства с базата данни и/или външни услуги (ако има такива), и връща резултатите обратно към frontend-а. Реализиран е като RESTful API.
3. **Слой за данни (Data Tier – Database):** Реализиран с MySQL. Този слой отговаря за персистентното съхранение и управление на всички данни, необходими за функционирането на приложението (информация за дестинации, хотели, атракции, потребителски планове и др.).

### Фаза на разработка

След приключване на етапа на планиране, който дефинира обхвата, изискванията, технологичния стек и основните проектни решения, започва фазата на разработка. В тази фаза се пристъпва към реалната имплементация на уебсайта за планиране на пътувания. Процесът на разработка обхваща създаването на базата данни, изграждането на backend логиката и API, имплементацията на frontend потребителския интерфейс и взаимодействие.

Разработката на настоящия проект започна с фундамента – проектирането и създаването на базата данни, тъй като тя представлява основата, върху която се изграждат всички останали функционалности на приложението.

#### Проектиране и имплементация на базата данни (MySQL)

Базата данни е критичен компонент на уебсайта, отговорен за съхранението, организацията и извличането на цялата информация, необходима за неговото функциониране – от детайли за дестинации и хотели до потребителски планове и готови пътувания. Избраната система за управление на бази данни е MySQL, поради своята надеждност, производителност и широка поддръжка на SQL стандарта.

**Логически модел на данните (ER диаграма)**

A computer screen shot of a computer program

AI-generated content may be incorrect.Въз основа на концептуалния дизайн, разработен във фазата на планиране, беше създаден детайлен логически модел на данните. Този модел, представен чрез Entity-Relationship (ER) диаграма, визуализира основните същности (таблици), техните атрибути (колони) и връзките между тях.

Фиг 3. ER диаграма на базата данни

**Физически модел на данните – DDL (Data Definition Language) скрипт**

Физическото създаване на структурата на базата данни в MySQL се извършва чрез SQL команди от типа Data Definition Language (DDL), основно CREATE TABLE. По-долу предоставям моя DDL скрипт.

CREATE TABLE `continents` (

`continent\_id` int NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL,

`image\_url` varchar(255) NOT NULL,

PRIMARY KEY (`continent\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `gallery` (

`gallery\_entry` int NOT NULL AUTO\_INCREMENT,

`gallery\_id` int NOT NULL,

`image\_url` varchar(255) NOT NULL,

PRIMARY KEY (`gallery\_entry`),

KEY `idx\_gallery\_id` (`gallery\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1116 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `countries` (

`country\_id` int NOT NULL AUTO\_INCREMENT,

`continent\_id` int NOT NULL,

`name` varchar(255) NOT NULL,

`gallery\_id` int NOT NULL,

`history` text NOT NULL,

PRIMARY KEY (`country\_id`),

KEY `continent\_id` (`continent\_id`),

KEY `gallery\_id` (`gallery\_id`),

CONSTRAINT `countries\_ibfk\_1` FOREIGN KEY (`continent\_id`) REFERENCES `continents` (`continent\_id`),

CONSTRAINT `countries\_ibfk\_2` FOREIGN KEY (`gallery\_id`) REFERENCES `gallery` (`gallery\_entry`)

) ENGINE=InnoDB AUTO\_INCREMENT=36 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `cuisine` (

`cuisine\_id` int NOT NULL AUTO\_INCREMENT,

`country\_id` int NOT NULL,

`text` varchar(255) NOT NULL,

PRIMARY KEY (`cuisine\_id`),

KEY `country\_id` (`country\_id`),

CONSTRAINT `cuisine\_ibfk\_1` FOREIGN KEY (`country\_id`) REFERENCES `countries` (`country\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=246 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `cities` (

`city\_id` int NOT NULL AUTO\_INCREMENT,

`country\_id` int NOT NULL,

`name` varchar(255) NOT NULL,

`image\_url` varchar(255) NOT NULL,

PRIMARY KEY (`city\_id`),

KEY `country\_id` (`country\_id`),

CONSTRAINT `cities\_ibfk\_1` FOREIGN KEY (`country\_id`) REFERENCES `countries` (`country\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=63 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `flights` (

`flight\_id` int NOT NULL AUTO\_INCREMENT,

`origin\_city\_id` int NOT NULL,

`destination\_city\_id` int NOT NULL,

`depart\_at` timestamp NOT NULL,

`arrive\_at` timestamp NOT NULL,

`number\_of\_stops` int NOT NULL,

`price` double NOT NULL,

`image\_url` varchar(255) NOT NULL,

`airline` varchar(255) NOT NULL,

PRIMARY KEY (`flight\_id`),

KEY `origin\_city\_id` (`origin\_city\_id`),

KEY `destination\_city\_id` (`destination\_city\_id`),

CONSTRAINT `flights\_ibfk\_1` FOREIGN KEY (`origin\_city\_id`) REFERENCES `cities` (`city\_id`),

CONSTRAINT `flights\_ibfk\_2` FOREIGN KEY (`destination\_city\_id`) REFERENCES `cities` (`city\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=2481 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `hotels` (

`hotel\_id` int NOT NULL AUTO\_INCREMENT,

`gallery\_id` int NOT NULL,

`city\_id` int NOT NULL,

`name` varchar(255) NOT NULL,

`location` varchar(255) NOT NULL,

`about` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL,

`price` double NOT NULL,

`preferred\_gallery\_entry\_id` int NOT NULL,

PRIMARY KEY (`hotel\_id`),

KEY `gallery\_id` (`gallery\_id`),

KEY `city\_id` (`city\_id`),

KEY `hotels\_ibfk\_3` (`preferred\_gallery\_entry\_id`),

CONSTRAINT `hotels\_ibfk\_1` FOREIGN KEY (`gallery\_id`) REFERENCES `gallery` (`gallery\_entry`),

CONSTRAINT `hotels\_ibfk\_2` FOREIGN KEY (`city\_id`) REFERENCES `cities` (`city\_id`),

CONSTRAINT `hotels\_ibfk\_3` FOREIGN KEY (`preferred\_gallery\_entry\_id`) REFERENCES `gallery` (`gallery\_entry`)

) ENGINE=InnoDB AUTO\_INCREMENT=189 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `hotels\_rating` (

`rating\_id` int NOT NULL AUTO\_INCREMENT,

`hotel\_id` int NOT NULL,

`rating` double NOT NULL,

PRIMARY KEY (`rating\_id`),

KEY `hotel\_id` (`hotel\_id`),

CONSTRAINT `hotels\_rating\_ibfk\_1` FOREIGN KEY (`hotel\_id`) REFERENCES `hotels` (`hotel\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=189 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `restaurants` (

`restaurant\_id` int NOT NULL AUTO\_INCREMENT,

`city\_id` int NOT NULL,

`name` varchar(255) NOT NULL,

`location` varchar(255) NOT NULL,

`image\_url` varchar(255) NOT NULL,

`price\_symbols` varchar(255) NOT NULL,

PRIMARY KEY (`restaurant\_id`),

KEY `city\_id` (`city\_id`),

CONSTRAINT `restaurants\_ibfk\_1` FOREIGN KEY (`city\_id`) REFERENCES `cities` (`city\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=246 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `restaurants\_rating` (

`rating\_id` int NOT NULL AUTO\_INCREMENT,

`restaurant\_id` int NOT NULL,

`rating` double NOT NULL,

PRIMARY KEY (`rating\_id`),

KEY `restaurant\_id` (`restaurant\_id`),

CONSTRAINT `restaurants\_rating\_ibfk\_1` FOREIGN KEY (`restaurant\_id`) REFERENCES `restaurants` (`restaurant\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=246 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `rooms` (

`room\_id` int NOT NULL AUTO\_INCREMENT,

`hotel\_id` int NOT NULL,

`name` varchar(255) NOT NULL,

`image\_url` varchar(255) NOT NULL,

PRIMARY KEY (`room\_id`),

KEY `hotel\_id` (`hotel\_id`),

CONSTRAINT `rooms\_ibfk\_1` FOREIGN KEY (`hotel\_id`) REFERENCES `hotels` (`hotel\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1512 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `tours` (

`tour\_id` int NOT NULL AUTO\_INCREMENT,

`city\_id` int NOT NULL,

`image\_url` varchar(255) NOT NULL,

`name` varchar(255) NOT NULL,

`days\_nights` varchar(255) NOT NULL,

`description` varchar(255) NOT NULL,

`highlights` varchar(255) NOT NULL,

PRIMARY KEY (`tour\_id`),

KEY `city\_id` (`city\_id`),

CONSTRAINT `tours\_ibfk\_1` FOREIGN KEY (`city\_id`) REFERENCES `cities` (`city\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=125 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `tours\_rating` (

`rating\_id` int NOT NULL AUTO\_INCREMENT,

`tour\_id` int NOT NULL,

`rating` double NOT NULL,

PRIMARY KEY (`rating\_id`),

KEY `ready\_tour\_id` (`tour\_id`),

CONSTRAINT `tours\_rating\_ibfk\_1` FOREIGN KEY (`tour\_id`) REFERENCES `tours` (`tour\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=125 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `attractions` (

`attraction\_id` int NOT NULL AUTO\_INCREMENT,

`city\_id` int NOT NULL,

`name` varchar(255) NOT NULL,

`location` varchar(255) NOT NULL,

`image\_url` varchar(255) NOT NULL,

`description` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL,

`opening\_hours` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL,

`best\_time\_to\_visit` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL,

`tickets\_website` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL,

`additional\_information` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL,

PRIMARY KEY (`attraction\_id`),

KEY `city\_id` (`city\_id`),

CONSTRAINT `attractions\_ibfk\_1` FOREIGN KEY (`city\_id`) REFERENCES `cities` (`city\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=248 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `hotel\_services` (

`hotel\_service\_id` int NOT NULL AUTO\_INCREMENT,

`hotel\_id` int NOT NULL,

`text` varchar(255) DEFAULT NULL,

PRIMARY KEY (`hotel\_service\_id`),

KEY `hotel\_id` (`hotel\_id`),

CONSTRAINT `hotel\_services\_ibfk\_1` FOREIGN KEY (`hotel\_id`) REFERENCES `hotels` (`hotel\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=946 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `nearby\_attractions` (

`attraction\_id` int NOT NULL,

`hotel\_id` int NOT NULL,

PRIMARY KEY (`attraction\_id`,`hotel\_id`),

KEY `hotel\_id` (`hotel\_id`),

CONSTRAINT `nearby\_attractions\_ibfk\_1` FOREIGN KEY (`attraction\_id`) REFERENCES `attractions` (`attraction\_id`),

CONSTRAINT `nearby\_attractions\_ibfk\_2` FOREIGN KEY (`hotel\_id`) REFERENCES `hotels` (`hotel\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `tour\_program` (

`tour\_program\_id` int NOT NULL AUTO\_INCREMENT,

`tour\_id` int NOT NULL,

`day` int NOT NULL,

`title` varchar(255) NOT NULL,

`description` text NOT NULL,

PRIMARY KEY (`tour\_program\_id`),

KEY `ready\_tour\_id` (`tour\_id`),

CONSTRAINT `tour\_program\_ibfk\_1` FOREIGN KEY (`tour\_id`) REFERENCES `tours` (`tour\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=617 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

**Създване на DML (Data Manipulation Language) скрипт**

След създаването на структурата на таблиците е необходимо те да бъдат попълнени с примерни данни, за да може приложението да бъде тествано и демонстрирано. Поради значителния обем на DML (Data Manipulation Language) скриптовете, необходими за цялостното попълване на базата с разнообразни данни, пълният им вариант е приложен като отделен .sql файл в директорията на проекта, а в настоящия текст ще бъдат представени само кратки илюстративни примери.

INSERT INTO travel\_db.continents (name,image\_url) VALUES

('Europe','https://i.imgur.com/cL2NtHu.png'),

('Africa','https://i.imgur.com/HsiPAql.png'),

('North America','https://i.imgur.com/2qaG7VW.png'),

('Asia','https://i.imgur.com/rBit4NZ.png'),

('South America','https://i.imgur.com/X2ILZ8L.png'),

('Australia / Oceania','https://i.imgur.com/X0A5hLP.png');

INSERT INTO travel\_db.cities (country\_id,name,image\_url) VALUES

(1,'Paris','https://i.imgur.com/wc6YKGY.png'),

(1,'Nice','https://i.imgur.com/ewrgYDE.png'),

(2,'Madrid','https://i.imgur.com/McgA98j.png'),

(2,'Barcelona','https://i.imgur.com/LWJlV2K.jpeg'),

(3,'Rome','https://i.imgur.com/vxrtBMX.jpeg'),

(3,'Milan','https://i.imgur.com/IQiZbZk.png'),

(3,'Venice','https://i.imgur.com/ENUGsHP.jpeg'),

(4,'London','https://i.imgur.com/GIttnwX.png'),

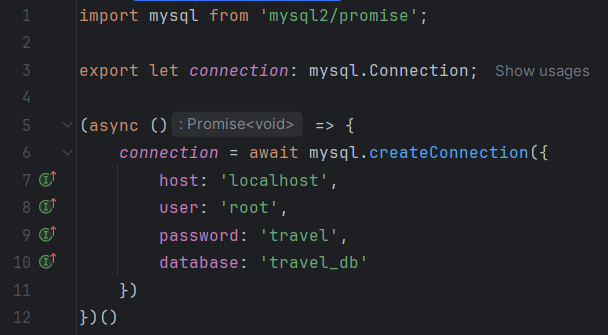
(4,'Edinburgh','https://i.imgur.com/oB0EKwc.png');

#### Разработка на Backend приложението (Node.js и Express.js)

Backend частта на уебсайта за планиране на пътувания е отговорна за обработката на бизнес логиката, взаимодействието с базата данни и предоставянето на RESTful API, което frontend приложението да консумира. За нейната реализация избрах Node.js и Express.js, а архитектурата е структурирана в три слоя (Handler, Service, Repository) с ясно дефиниран Domain модел.

* **Връзка с базата данни (MySQL)**

Основата на всяко динамично уеб приложение е неговата база данни. Връзката с MySQL базата данни се установява с помощта на библиотеката mysql2/promise, която предоставя модерен, базиран на Promises, начин за работа с MySQL в Node.js. Това е предпочитан подход пред традиционните callback-базирани методи, тъй като улеснява писането и поддръжката на асинхронен код чрез async/await синтаксиса.



Фиг 4. Свързване с базата данни

**export let connection: mysql.Connection;** - декларира променлива connection, която ще съхранява установената връзка. Export ключовата дума позволява тази връзка да бъде използвана в други модули на приложението (например в Repository слоя).

**mysql.createConnection({...})**: Асинхронна функция, която създава връзка към MySQL сървъра. Конфигурационният обект включва:

* **host**: Адресът на сървъра (в случая 'localhost', тъй като базата данни е на същата машина, на която се разработва).
* **user**: Потребителското име за достъп до MySQL.
* **password**: Паролата за съответния потребител. (В продукционна среда тези данни ще се управляват сигурно, например чрез променливи на средата (environment variables), а не да са хардкоднати)
* **database**: Името на базата данни, с която приложението ще работи (travel\_db).

Този подход осигурява еднократно установяване на връзката при стартиране на приложението, като инстанцията на връзката (connection) след това се преизползва от различните части на Repository слоя за изпълнение на SQL заявки.

* **Стартиране на Backend сървъра и основни конфигурации**

За стартирането на HTTP сървъра и обработката на заявките се използва рамката Express.js. Примерният код за инициализация на сървъра е:

A screen shot of a computer program

AI-generated content may be incorrect.

Фиг 5. Стартиране на Backend сървъра

Обяснение на ключовите елементи:

**app.use(express.json())**: Този middleware е от съществено значение за обработката на API заявки, които изпращат данни във формат JSON в тялото на заявката (request body), каквито са типичните POST или PUT заявки. express.json() парсва тези JSON данни и ги прави достъпни в req.body обекта на съответния handler. Без него, req.body би бил undefined при получаване на JSON данни.

**app.use(cors())**: CORS (Cross-Origin Resource Sharing) е механизъм за сигурност в уеб браузърите, който ограничава HTTP заявките от един произход (origin: схема, домейн, порт) към друг. Тъй като frontend приложението (React) вероятно ще се изпълнява на различен порт (напр. localhost:5173 за Vite) от backend сървъра (напр. localhost:8000) по време на разработка, браузърът би блокирал заявките. app.use(cors()) добавя необходимите HTTP хедъри към отговорите на сървъра, които указват на браузъра, че заявките от други произходи са разрешени. В продукционна среда CORS може да бъде конфигуриран по-рестриктивно, за да разрешава заявки само от конкретни домейни.

**app.listen(port, () => { ... })**: Тази функция стартира HTTP сървъра и го кара да "слуша" за входящи заявки на указания порт. process.env.PORT позволява портът да бъде конфигуриран чрез променлива на средата (полезно при deployment), а ако такава не е зададена, се използва порт 8000 по подразбиране. Когато сървърът стартира успешно, се изпълнява предоставената callback функция, която в случая извежда съобщение в конзолата.

**app: Express = express()**: Създава инстанция на Express приложението. Тази инстанция app се използва за конфигуриране на middleware, дефиниране на маршрути и стартиране на сървъра.

**Маршрутизация (app.use('/path', routes)):** Кодът демонстрира добър подход за модуларизация на маршрутите. Вместо всички маршрути да се дефинират в един файл, те са групирани по ресурси (напр. /continents, /countries, /hotels) и имплементирани в отделни модули (continentRoutes, countryRoutes и т.н., които вероятно са инстанции на express.Router()). app.use('/continents', continentRoutes) указва, че всички заявки, започващи с /continents, трябва да бъдат обработени от continentRoutes маршрутизатора.

* **Трислойна архитектура (Handler, Service, Repository) и Domain модел**

За структуриране на backend логиката избрах трислойна архитектура, която е широко възприет подход за изграждане на поддържаеми и мащабируеми приложения. Тази архитектура разделя отговорностите на приложението на логически слоеве, което носи редица предимства:

* **Separation of Concerns (Разделяне на отговорностите):** Всеки слой има ясно дефинирана роля и се фокусира върху конкретен аспект на приложението. Handler-ите се занимават с HTTP комуникацията, Service слоят – с бизнес логиката, а Repository слоят – с достъпа до данни.
* **Maintainability (Поддръжка):** Промени в един слой (напр. смяна на базата данни, което би засегнало Repository слоя) е по-малко вероятно да изискват значителни промени в другите слоеве, стига интерфейсите между тях да се запазят. Това улеснява поддръжката и еволюцията на приложението.
* **Testability (Тестируемост):** Отделните слоеве могат да бъдат тествани по-лесно и изолирано. Например, Service слоят може да бъде тестван чрез mock-ване на Repository слоя, без да е необходима реална база данни.
* **Reusability (Преизползваемост):** Бизнес логиката в Service слоя може да бъде преизползвана от различни Handler-и или дори от други части на системата (напр. конзолни скриптове, други API-та).
* **Readability and Organization (Четимост и организация):** Ясно дефинираните слоеве правят кода по-структуриран, по-лесен за разбиране и навигация, особено при по-големи проекти.

Нека разгледаме всеки слой и Domain модела:

* **Domain Модел** - слоят на домейна (или бизнес моделът) дефинира основните същности (entities) и типове данни, с които приложението работи. Той представлява сърцето на бизнес логиката. Пример с хотелите:

A screenshot of a computer program

AI-generated content may be incorrect.

Фиг 6. Domain модел на хотел

**Hotel**: Този тип дефинира как изглежда един хотел в рамките на бизнес логиката на приложението (Service и Handler слоевете). Имайте предвид, че полетата imageUrl и rating са добавени като опционални, тъй като примерният Service код ги добавя допълнително.

**HotelDb**: Този тип представлява данните така, както се връщат директно от MySQL базата данни. RowDataPacket е тип от mysql2 библиотеката. Често имената на колоните в базата данни са в snake\_case (напр. hotel\_id), докато в JavaScript/TypeScript кодът е прието да се използва camelCase (напр. hotelId). Repository слоят е отговорен за преобразуването (мапването) между тези два формата.

* **Repository слой** - Repository слоят е отговорен изключително за взаимодействието с източника на данни – в случая MySQL базата данни. Той абстрахира логиката за достъп до данни от останалата част на приложението.

A screen shot of a computer program

AI-generated content may be incorrect.

Фиг 7. Repository слой

**Инкапсулация на SQL:** Всички SQL заявки са концентрирани тук. Ако в бъдеще се наложи промяна на SQL диалекта или дори на самата база данни, промените ще бъдат локализирани предимно в този слой.

**Мапване на данни:** Както споменах, Repository-то преобразува данните от формата на базата данни (HotelDb) към домейния модел (Hotel).

**Динамични заявки:** getAllHotels демонстрира как могат да се изграждат заявки с опционални филтри.

* A screenshot of a computer program

  AI-generated content may be incorrect.**Service Слой** - Service слоят съдържа бизнес логиката на приложението. Той координира извличането на данни (чрез Repository слоя), прилага бизнес правила, комбинира данни от различни източници и подготвя данните за Handler слоя.

Фиг 8. Service слой

**Оркестрация:** getHotelById ясно показва как Service слоят може да извиква множество методи от Repository-та или други Service-и (Promise.all за паралелно изпълнение) и да агрегира резултатите.

**Бизнес правила:** Тук биха се намирали по-сложни бизнес правила, валидации (които не са свързани директно с HTTP заявката) или изчисления.

**Декуплиране:** Service слоят не знае как точно се извличат данните от базата; той просто използва интерфейса, предоставен от Repository слоя.

* Handler (Controller) Слой - Handler слоят е входната точка за HTTP заявките. Неговата основна отговорност е да приеме заявката, да извлече необходимите параметри, да извика A screen shot of a computer program

  AI-generated content may be incorrect.съответния метод от Service слоя и да върне HTTP отговор.

Фиг 9. Handler слой

**HTTP взаимодействие:** Този слой се занимава изцяло с HTTP аспектите – req (request) и res (response) обекти, статус кодове, хедъри.

**Валидация на входни данни (от HTTP):** Въпреки че не е силно застъпена в примера, Handler-ите са добро място за валидиране на входните данни от заявката (параметри, тяло на заявката), преди да се извика Service слоят. Могат да се използват библиотеки като Joi или express-validator.

**Форматиране на отговора:** Handler-ът форматира данните, получени от Service слоя, в подходящ HTTP отговор (най-често JSON).

**Обработка на грешки (на ниво HTTP):** Улавя грешки, хвърлени от по-долните слоеве, и ги преобразува в подходящи HTTP статус кодове и съобщения за грешка.

Използването на тази трислойна архитектура, заедно с ясно дефиниран Domain модел, допринася значително за създаването на добре структуриран, гъвкав и лесен за поддръжка backend за уебсайта за планиране на пътувания. Тя позволява ясно разделение на отговорностите и улеснява бъдещото развитие и тестване на системата.

#### Разработка на Frontend приложението (React, Vite, TypeScript, Shadcn-UI)

Frontend частта на уебсайта за планиране на пътувания е лицето на приложението – това, с което потребителят директно взаимодейства. Нейната цел е да предостави интуитивен, отзивчив и визуално привлекателен интерфейс, който улеснява потребителя в процеса на търсене, избор и организиране на неговото пътуване. За реализацията са използвани съвременни технологии и подходи като React, Vite, TypeScript, Shadcn-UI, React Query и React Router.

* **Комуникация с Backend API чрез Axios**

A screenshot of a computer

AI-generated content may be incorrect.За взаимодействие с backend API-то избрах библиотеката Axios. За да се избегне повтарящото се изписване на базовия URL адрес на API-то и за по-лесна конфигурация, създадох персонализирана инстанция на Axios:

Фиг 10. Инстанция на Axios

**axios.create({...})**: Този метод позволява създаването на нова инстанция на Axios с предварително зададени конфигурации.

**baseURL**: Една от най-полезните конфигурации. Всички заявки, направени чрез axiosInstance, автоматично ще добавят този baseURL към относителния път на заявката (напр. axiosInstance.get('/hotels') ще направи заявка към <http://localhost:8000/hotels>).

Тази конфигурация прави кода за API заявки по-чист и по-лесен за поддръжка.

* **Frontend архитектура: Service - Custom Hook – Component**

За организиране на логиката за извличане и управление на данни от API-то използвах архитектура, която разделя отговорностите на три основни части: Service, Custom Hook и Component.

* A screen shot of a computer program

  AI-generated content may be incorrect.**Service слой** - Този слой е отговорен за капсулирането на логиката за извършване на конкретни API заявки. Той абстрахира детайлите на HTTP комуникацията от останалата част на приложението.

Фиг 11. Service слой

**Капсулация:** Всеки метод (напр. getHotels, getHotelById) отговаря за конкретна API операция.

**Използване на axiosInstance:** Гарантира, че всички заявки използват конфигурирания baseURL.

**Типизиране:** Използването на TypeScript типове (Hotel, HotelInfo, HotelParamsProps) подобрява четимостта и надеждността на кода.

**Обработка на грешки:** Базовата обработка на грешки е показана, като е важно грешките да се хвърлят нагоре, за да могат да бъдат уловени и обработени от React Query или други консуматори на тези функции.

* A screenshot of a computer screen

  AI-generated content may be incorrect.**Custom Hooks** - Custom Hooks са механизъм в React за преизползване на логика, свързана със състояние (stateful logic). В контекста на извличане на данни, те често се използват за интегриране с библиотеки като React Query, като абстрахират конфигурацията на заявката и управлението на нейното състояние (зареждане, данни, грешки).

Фиг 12. Custom Hook

**Интеграция с React Query (@tanstack/react-query):** useQuery е основната "кука" от React Query. Тя управлява целия жизнен цикъл на асинхронната заявка: извличане на данни, кеширане, фоново обновяване, обработка на състояния на зареждане и грешки.

**queryKey**: Уникален идентификатор за всяка заявка. React Query използва този ключ за управление на кеша. Ако данните за определен queryKey вече са в кеша и не са остарели, те се връщат незабавно, което подобрява производителността и потребителското изживяване. Включването на параметрите (params, hotelId) в ключа е важно, за да се разграничават заявки с различни входни данни.

**queryFn**: Функция, която връща Promise (в случая, методите от HotelsService). React Query извиква тази функция, за да извлече данните.

**Абстракция:** Компонентите, които използват тези custom hooks, не трябва да се грижат за детайлите на API заявките или управлението на състоянието на данните; те просто консумират резултата.

* A computer screen shot of a program

  AI-generated content may be incorrect.Използване в компоненти - Компонентите използват тези custom hooks, за да получат достъп до данните и състоянията, управлявани от React Query.

Фиг 13. Използване в компоненти

**Декларативност:** Компонентът декларира какви данни са му необходими (useGetHotels) и React Query (чрез custom hook-а) се грижи за останалото.

**Управление на състояния:** useQuery връща обект със полезни свойства като data (извлечените данни), isLoading (булева стойност, показваща дали заявката е в процес на изпълнение) и isError/error (за обработка на грешки). Това улеснява показването на съобщения за зареждане или грешки в потребителския интерфейс.

Тази архитектура (Service - Custom Hook - Component) насърчава разделянето на отговорностите, прави кода по-тестируем и преизползваем, и подобрява организацията на frontend приложението.

* **Компонентно-базирана архитектура и Single Responsibility Principle**

Както е характерно за React, потребителският интерфейс е изграден от множество компоненти. В този проект е следван принципът на преизползваемост и Single Responsibility Principle (Принцип на единствената отговорност). Всеки компонент е проектиран да изпълнява една конкретна задача или да представя определена част от интерфейса.

**Предимства на SRP за компоненти:**

* По-лесно разбиране и поддръжка: Малките, фокусирани компоненти са по-лесни за разбиране отколкото големи, монолитни такива
* По-добра тестируемост: Всеки компонент може да бъде тестван изолирано
* Повишена преизползваемост: Компоненти, които правят едно нещо добре, могат лесно да бъдат използвани на различни места в приложението
* **Управление на глобално състояние с React Context API**

A screen shot of a computer program

AI-generated content may be incorrect.За управление на глобалното състояние, свързано с избора на потребителя по време на планирането на пътуването (избрани полети, хотели, ресторанти, атракции), е използван React Context API. Това позволява на различни компоненти в дървото да имат достъп до тези данни и да ги модифицират, без да се налага сложно предаване на props (prop drilling) през множество нива.

Фиг 14. React Context – 1

A screen shot of a computer program

AI-generated content may be incorrect. фиг 15. React Context - 2

**React.createContext()**: Създава обект Context.

**TravelProvider**: Компонент, който използва TravelContext.Provider за да предостави стойността на контекста (в случая обект с избраните елементи и функциите за тяхната промяна) на всички свои дъщерни компоненти.

**React.useState**: Използва се вътре в TravelProvider за управление на състоянието на избраните елементи.

**useTravelPlan**: Custom hook, който улеснява консумирането на контекста в компонентите. Той абстрахира React.useContext(TravelContext) и включва проверка, за да се увери, че се използва правилно (в рамките на TravelProvider).

**Приложение:** Този контекст ще се използва от компонентите, които позволяват на потребителя да избира полети, хотели и т.н., както и от страницата TravelPlan, която ще визуализира финалния обобщен план.

* **Маршрутизация (Routing) с React Router**

A screen shot of a computer program

AI-generated content may be incorrect.За навигация между различните страници и изгледи на Single Page Application (SPA) се използва библиотеката react-router-dom. Предоставеният код за конфигурация на маршрутизацията в главния App компонент:

Фиг 16. Маршрутизация с React Router

Тази структура на frontend приложението, с ясно разделение на отговорностите, ефективно управление на състоянието и данни, и добре дефинирана навигация, е ключова за създаването на качествен и лесен за поддръжка уебсайт за планиране на пътувания. Използването на TypeScript навсякъде допълнително повишава надеждността и улеснява разработката. Интеграцията на Shadcn-UI (спомената във фазата на планиране) допринася за бързото изграждане на визуално привлекателен и достъпен потребителски интерфейс.

# Заключение

Настоящият курсов проект беше посветен на проектирането и разработката на уебсайт за планиране на пътувания – задача с нарастваща актуалност в съвременния дигитален свят. Целта беше да се създаде функционален прототип, който да улесни потребителите в комплексния процес на организиране на техните пътешествия, като предоставя интегрирана информация и интуитивни инструменти на едно място.

Самостоятелно решени задачи и принос:

По време на разработката на този проект, самостоятелно реших редица ключови задачи, демонстрирайки комплексни умения в областта на софтуерното инженерство:

* Анализ и планиране - извърших анализ на съществуващи платформи за планиране на пътувания, на базата на който дефинирах основните проблеми и нужди на потребителите. Формулирах детайлни функционални и нефункционални изисквания към системата. Напраних обоснован избор на модерен технологичен стек, включващ Node.js, Express.js, MySQL за backend, и React (с Vite и TypeScript), Shadcn-UI, React Query и React Router за frontend. Проектирах потребителският интерфейс (UI) и потребителското изживяване (UX) с помощта на Figma. Планирах архитектурата на приложението – трислойна за backend и компонентно-базирана с разделяне на отговорностите (service-hook-component) за frontend, както и концептуалният модел на базата данни.
* **Разработка на база данни:** Самостоятелно проектирах и имплементирах DDL (Data Definition Language) скриптове за създаване на структурата на релационната база данни в MySQL, включваща таблици за държави, дестинации, хотели, атракции, ресторанти, полети и готови пътувания. Създадох и DML (Data Manipulation Language) скриптове за попълване на базата с разнообразни примерни данни, необходими за тестване и демонстрация.
* **Разработка на Backend:** Имплементирах връзката с MySQL базата данни чрез mysql2/promise. Конфигуриран бе Express.js сървър, включително необходимите middleware (CORS, JSON parser) и система за маршрутизация. Разработих RESTful API, следвайки трислойна архитектура (handler – service - repository).
* **Разработка на Frontend:** Настроих Axios инстанция за ефективна комуникация с backend API-то. Имплементирах потребителският интерфейс с React, TypeScript и Vite, като приложих следните концепции – Service слой, Custom Hooks с React Query, компонентна архитектура, управление на глобално състояние и маршрутизация.

**Изводи:**

Разработеният прототип успешно демонстрира осъществимостта и ползите от създаването на интегрирана уеб платформа за планиране на пътувания. Избраният технологичен стек (Node.js, Express.js, MySQL, React, Vite, TypeScript, Shadcn-UI, React Query, React Router) e подходящ и ефективен за целите на проекта, позволявайки бърза разработка и постигане на висококачествен резултат. Прилагането на трислойна архитектура в backend-а и компонентно-базиран подход с разделяне на отговорностите в frontend-а допринесоха за създаването на добре организиран, четим и лесен за поддръжка код. Проектът потвърждава, че централизирането на информацията и инструментите за планиране може значително да улесни потребителите и да подобри тяхното преживяване.

**Препоръки за по-нататъшно доразработване на темата:**

* **Имплементиране на потребителски профили:** Въвеждане на система за регистрация и автентикация на потребители, което ще позволи запазване на търсения, планове за пътуване, предпочитания и персонализирани препоръки.
* **Интеграция с реални външни API-та:** Свързване с API-та на доставчици на туристически услуги (авиокомпании, хотелски агрегатори, системи за резервации на атракции) за достъп до актуална информация в реално време и евентуално реализиране на функционалност за директни резервации.
* **Разширена филтрация и сортиране:** Добавяне на по-детайлни и гъвкави опции за филтриране и сортиране на резултатите от търсенето.
* **Интерактивни карти:** Интегриране на карти за визуализация на местоположението на хотели, атракции, ресторанти и планиране на маршрути в рамките на дестинацията.
* **Система за потребителски оценки и коментари:** Възможност потребителите да оценяват и коментират хотели, атракции, ресторанти и готови пътувания директно в платформата.

Полезността на този проект се изразява в способността му да трансформира често сложния и времеемък процес на планиране на пътувания в едно по-структурирано, интуитивно и приятно занимание. Чрез централизиране на информацията и инструментите за избор, уебсайтът спестява ценно време на потребителите, намалява стреса от търсенето в множество източници и им помага да организират по-добре своите пътешествия, допринасяйки за по-пълноценни и безпроблемни преживявания.