ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

КАТЕДРА “КОМПЮТЪРНА ИНФОРМАТИКА”



**Спецификация на изискванията**

**за**

**Wander Studio**

Изготвена от: **Габриела Балабанска**

Фак. № **2301727003**

Специалност: **Софтуерни технологии със специализация софтуерни архитектури и средства**Дата:07.06.2025г.

Съдържание

[1. Въведение 3](#_Toc200310107)

[1.1 Цел 3](#_Toc200310108)

[1.2 Конвенции 3](#_Toc200310109)

[1.5 Референции 3](#_Toc200310110)

[2. Общо Описание 4](#_Toc200310111)

[2.1 Обща Перспектива 4](#_Toc200310112)

[2.2 Характеристики на продукта 4](#_Toc200310113)

[2.3 Видове Потребители и Характеристики 4](#_Toc200310114)

[2.4 Операционна среда 5](#_Toc200310115)

[2.5 Ограничения за дизайна и реализацията 5](#_Toc200310116)

[2.6 Предположения и зависимости 6](#_Toc200310117)

[3. Външни Изисквания към интерфейса 6](#_Toc200310118)

[3.1 Потребителски интерфейси 6](#_Toc200310119)

[3.2 Хардуерни Интерфейси 9](#_Toc200310120)

[3.3 Софтуерни Интерфейси 9](#_Toc200310121)

[3.4 Комуникационни Интерфейси 10](#_Toc200310122)

[4. Функционални изисквания към проекта 10](#_Toc200310123)

[4.1 Регистрация на потребител 10](#_Toc200310124)

[4.2 Включване на потребител 12](#_Toc200310125)

[4.3 Добавяне на приятел 13](#_Toc200310126)

[5. Архитектура на проекта 13](#_Toc200310127)

[5.1 Back-end 13](#_Toc200310128)

[5.2 Модел на базата данни 18](#_Toc200310129)

[5.3 Front-end 19](#_Toc200310130)

[6. Използване на Google Maps API 22](#_Toc200310131)

[7. Бъдещи цели 22](#_Toc200310132)

# 1. Въведение

## 1.1 Цел

Този документ описва функционалните и нефункционалните изисквания на софтуерния продукт Wander Studio, разработен като курсов проект по дисциплината „*Практикум по практически проект №2*“.  
Приложението Wander Studio има за цел да улесни потребителите в процеса на организиране на пътувания, като им предоставя интуитивен интерфейс за въвеждане на основните характеристики на пътуването, покана на приятели, както и визуализация на свързан маршрут върху карта с обозначени дестинации.

1.2 Конвенции  
  
Към настоящия момент в документа не се използват конвенции.   
  
1.3 За кого е предназначен този документ  
  
Настоящият документ е предназначен за разработчика на проекта, както и за преподавателя, отговарящ за провеждането на дисциплината „*Практикум по практически проект №2“*. Съдържанието на документа предоставя база за бъдещо усъвършенстване или допълнително развитие на системата.   
  
1.4 Обхват на продукта  
  
Wander Studio представлява уеб-базирано приложение, предназначено за създаване, планиране и споделяне на пътувания между потребители. Основната му функционалност включва дефиниране на маршрути, избор на дестинации, добавяне на описание на пътуването, покана на приятели, както и визуализиране на избраните места върху карта чрез интеграция с Google Maps API.  
  
Приложението цели да подпомогне индивидуални потребители и групи в организиране на лични или групови пътувания, като предлага визуално ясен и лесен за използване интерфейс.   
  
За момента системата поддържа основна функционалност за създаване и управление на маршрути, като в бъдеще се предвижда добавяне на възможности като съвместна редакция, качване на мултимедийно съдържание и синхронизация с външни календари.

1.5 Референции  
  
Google Maps Platform – <https://developers.google.com/maps>

# 2. Общо Описание

## 2.1 Обща Перспектива

Wander Studio предлага интуитивен и изчистен интерфейс, който съчетава необходимата функционалност с минимална сложност. Приложението елиминира излишните стъпки и менюта, осигурявайки бързо и лесно планиране на пътувания.   
  
Системата на приложението Wander Studio използва интеграция с Google Maps API за:

* търсене по въведен от потребителя град или адрес
* отбелязване на места с последователни маркери (A, B, C…), показващи избраните дестинации
* визуализация на последователен маршрут между дестинациите
* динамично центриране и приближаване на картата спрямо зададените локации

## 2.2 Характеристики на продукта

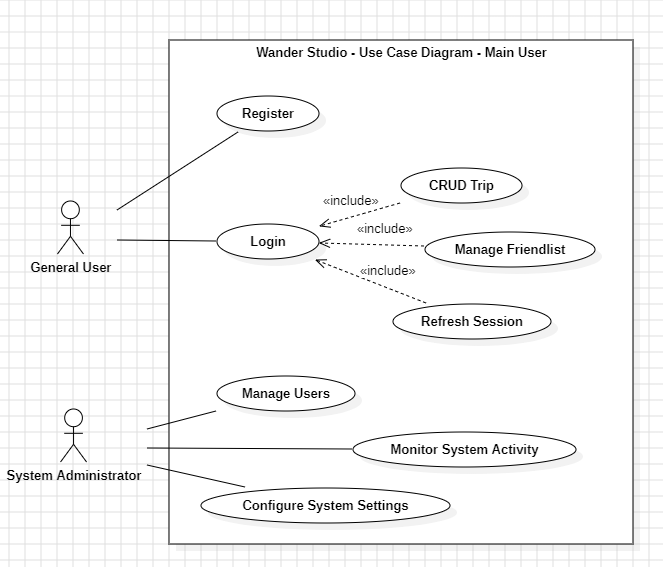
Приложението Wander Studio предоставя следните функционалности на потребителя:

* потребителски вход и регистрация
* преглед на всички пътувания
* създаване и описване на ново пътуване
* добавяне на дестинации чрез търсене по град, адрес или забележителност
* визуализация на последователен маршрут между дестинациите
* редакция и изтриване на вече създадени пътувания
* управление на списъка с приятели
* навигиране чрез странично меню

## 2.3 Видове Потребители и Характеристики

Wander Studio е насочено към широк кръг потребители, включително такива без задълбочени технически познания. То предлага лесен и достъпен начин за създаване, организиране и споделяне на пътувания, като фокусът е върху интуитивното потребителско изживяване. Към момента приложението е уеб-базирано и може да се използва през браузър на различни устройства.  
  
С помощта на Use Case диаграмата са представени участниците в системата. Това са:

* потребител – има достъп до пълната функционалност на приложението след регистрация и включване в системата с валидни потребителски данни
* системен администратор – следни за правилното функциониране на системата



## 2.4 Операционна среда

Wander Studio е уеб-базирано приложение, достъпно чрез съвременни интернет браузъри (като Google Chrome, Mozilla Firefox, Microsoft Edge и др.). Работи коректно както на настолни, така и на мобилни устройства, при наличие на интернет връзка.

Сървърната част е реализиран със Spring Boot, а потребителската част с Angular. За база данни се използва MySQL като Docker контейнер. Интеграцията с Google Maps API позволява визуализиране и управление на маршрути директно в приложението.

## 2.5 Ограничения за дизайна и реализацията

Основно ограничение за работата на Wander Studio е необходимостта от постоянна интернет връзка. Без активна връзка системата не може да зареди картата, да извърши търсене на дестинации или да синхронизира информация между потребителите.  
  
Освен това, поради използването на Google Maps API, съществуват ограничения, свързани с обема на заявките и броя на безплатните заявки на месец.   
  
Дизайнът също така е оптимизиран за устройства със средна към висока резолюция, което може да доведе до по-ограничено визуално представяне на по-стари или малки устройствa.

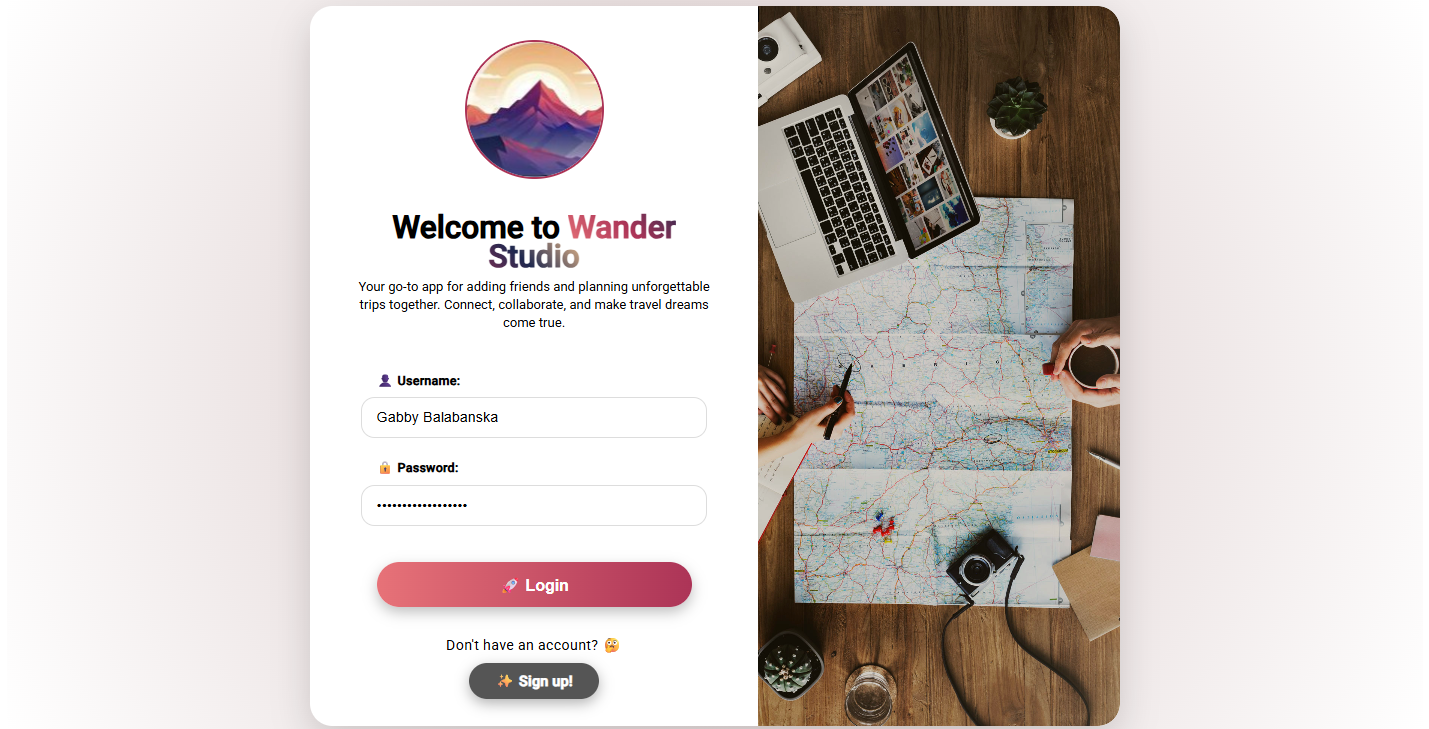
## 2.6 Предположения и зависимости

Предполага се, че потребителите използват устройства с поддържан браузър и стабилна интернет връзка. Системата разчита на външни услуги като Google Maps API, чиято наличност и достъпност са критични за основната функционалност на приложението. В случай на прекъсване или ограничение на достъпа до тези услуги, Wander Studio няма да може да визуализира маршрути и дестинации както е предвидено.  
  
Също така се приема, че потребителите имат базови умения за работа с уеб приложения.

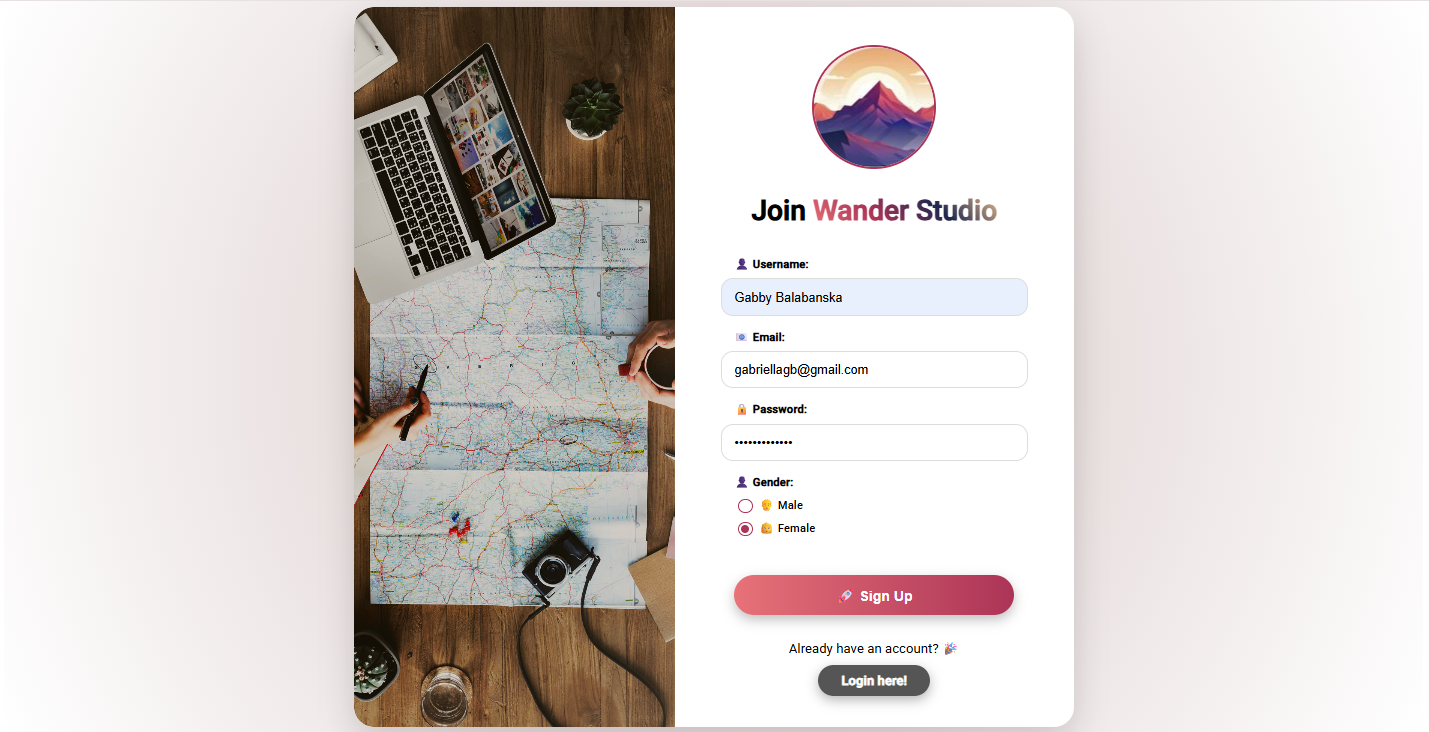
# 3. Външни Изисквания към интерфейса

## 3.1 Потребителски интерфейси

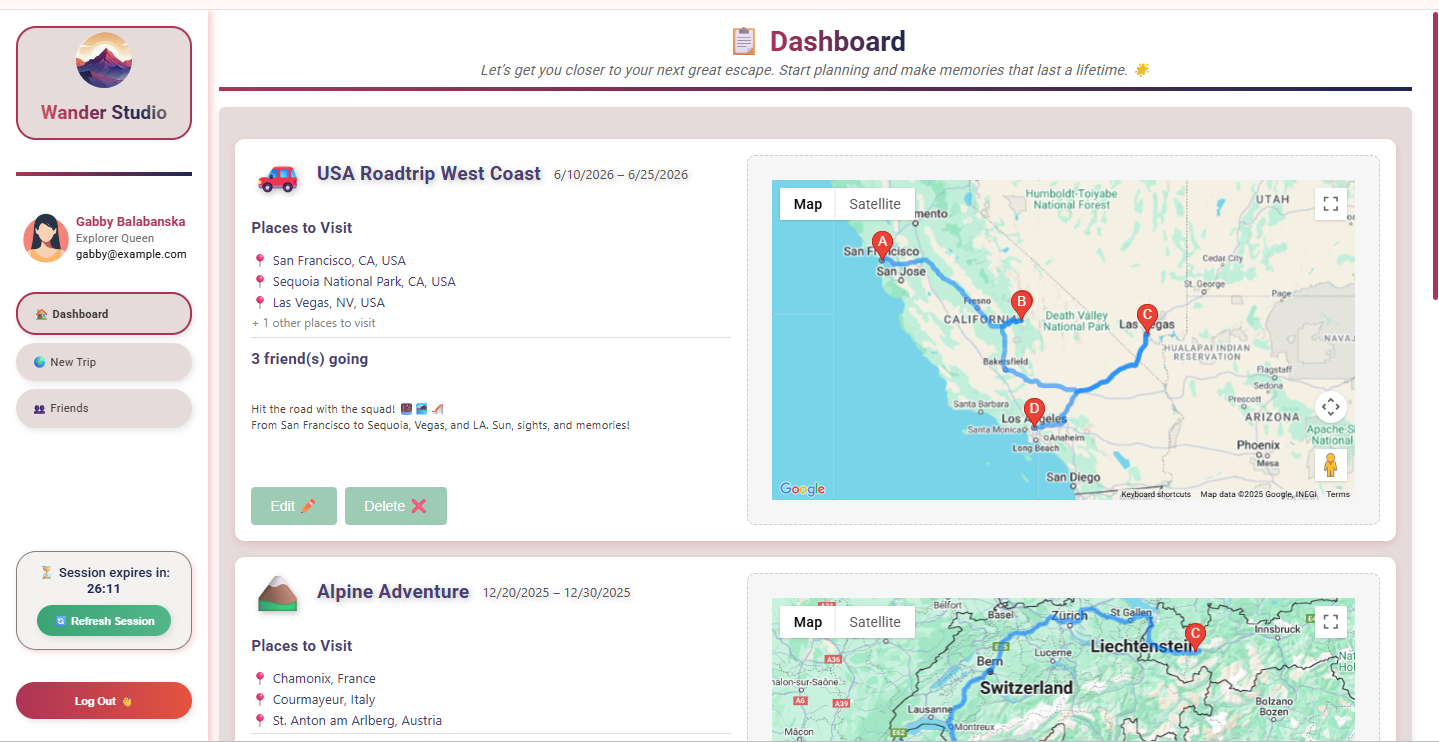
При отваряне на приложението се зарежда форма за вход:



Ако потребителят няма съществуващ акаунт, той може да се регистрира чрез регистрационната форма:



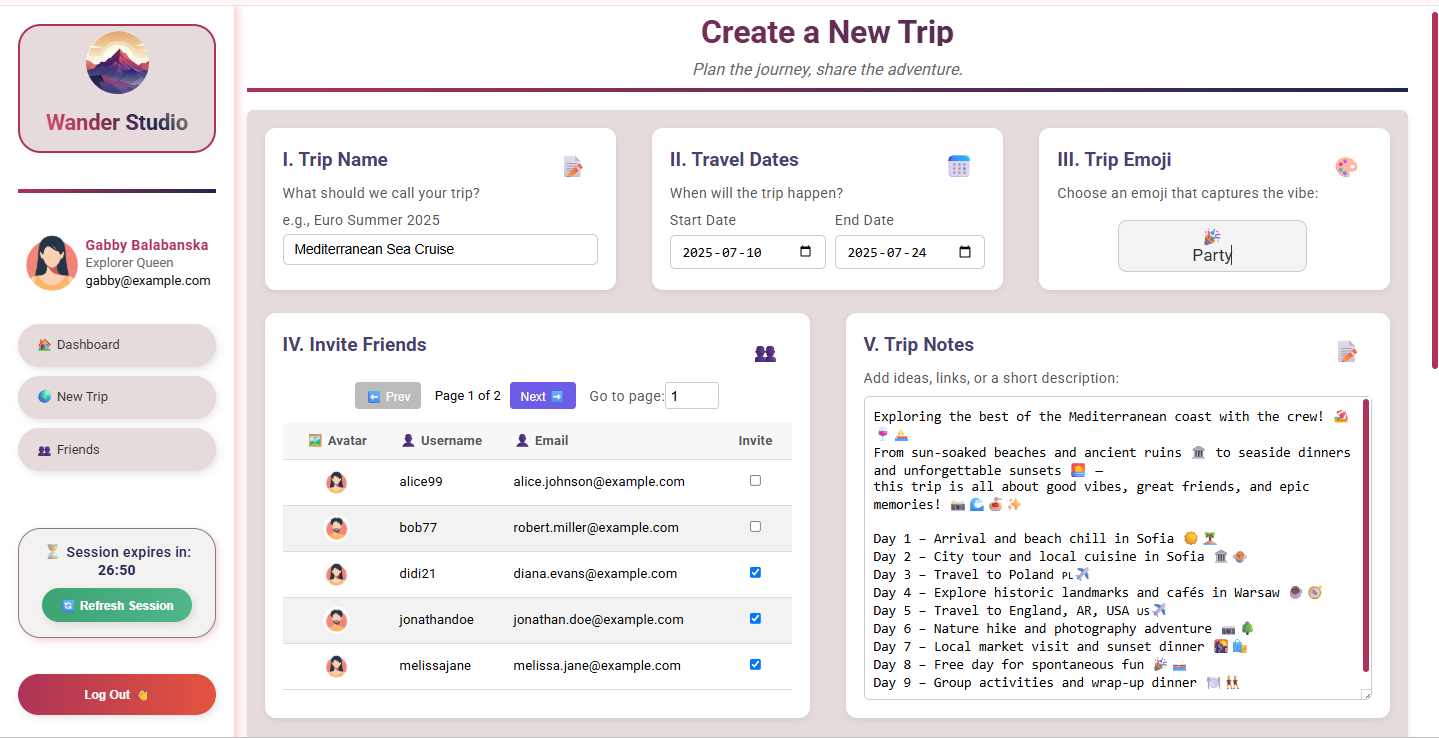
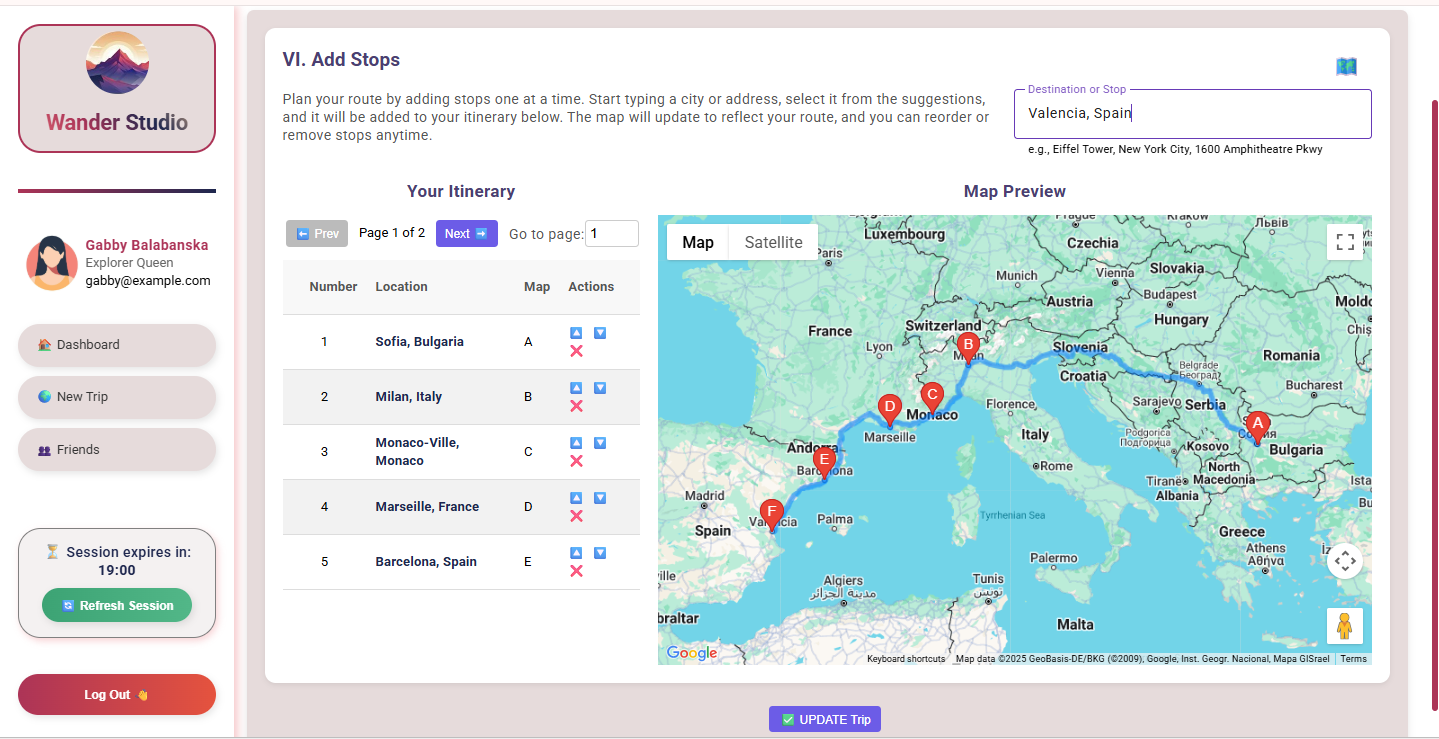
След успешно въвеждане на валидни потребителско име и парола, потребителят се пренасочва към екрана с всички пътувания, в които участва. При вход се генерира и изпраща към браузъра бисквитка, съдържаща авторизационен JWT токен. След изтичане на валидността на токена, достъпът до приложението се ограничава, докато потребителят не влезе отново в системата.  
  
Навигацията в приложението се осъществява чрез странично меню, което съдържа кратка информация за текущия потребител, връзки към основните страници и бутон за опресняване на сесията. При натискане на бутона се изпраща нова бисквитка с обновен JWT токен с удължена валидност, с цел да се избегне необходимостта от повторно влизане на всеки 30 минути.

  
  
Чрез началната страница (Dashboard) потребителят има достъп до информация за всяко пътуване, в което участва:

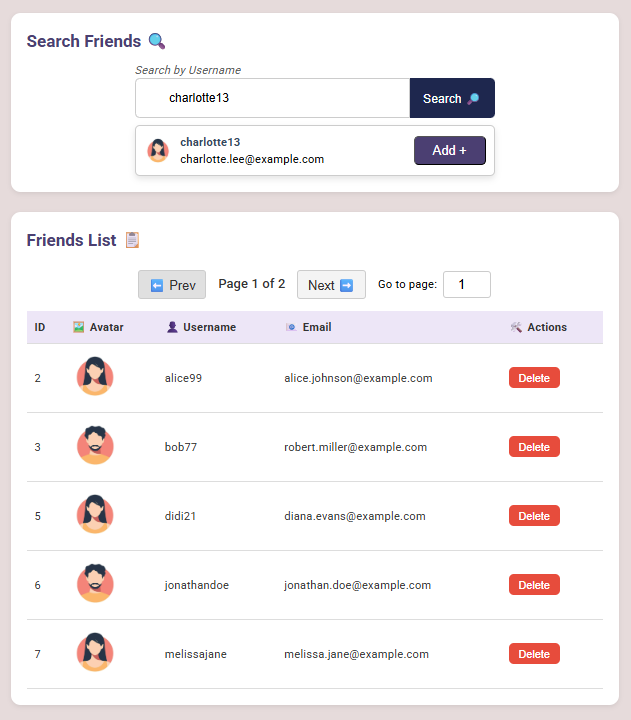
* име на пътуването
* начална и крайна дата
* последователност на дестинациите, визуализирани върху карта
* описание на пътуването
* списък с приятели, участващи в пътуването

Освен това, от Dashboard страницата потребителят, който е собственик на дадено пътуване може да:

* редактира пътуване
* изтрива пътуване

Страницата за редакция е аналогична на тази за създаване на ново пътуване, показана по-долу:  
  
 

Последната функционална страница в приложението е предназначена за управление на списъка с приятели, където потребителят може да търси, добавя или премахва приятели от своя профил.



## 3.2 Хардуерни Интерфейси

Wander Studio не изисква специализиран хардуер и не комуникира директно с хардуерни устройства. Приложението е уеб-базирано и функционира коректно на всяко устройство, което поддържа съвременен уеб браузър и интернет връзка.  
  
В случай на бъдещо разширение, при което се използва вградения GPS на мобилните устройства за определяне на текуща локация, управлението му ще се извършва чрез браузъра и операционната система, без необходимост от директна хардуерна интеграция.

## 3.3 Софтуерни Интерфейси

Wander Studio използва следните основни софтуерни компоненти и интерфейси:

* Клиентска част (Angula), която осъществява комуникация със сървъра чрез REST API
* Сървърна част (Spring Boot), която обработва заявките от клиента, управлява логиката и комуникира с базата данни
* База данни (MySQL), която се намира в отделен Docker контейнер и се използва за съхранение на информация за потребители, пътувания, дестинации и приятели
* Информацията между клиент и сървър се предава под формата на JSON. За сигурността на комуникацията се използва HTTPS протокол и JWT (JSON Web Token) за авторизация.
* За визуализация на маршрути и дестинации се използва Google Maps JavaScript API, който осигурява търсене на местоположения и изчертаване на маршрути директно в браузъра.

## 3.4 Комуникационни Интерфейси

Комуникацията в системата се осъществява чрез стандартни мрежови протоколи, като всички взаимодействия между клиентската и сървърната част преминават през HTTPS.   
  
Комуникацията между сървъра и базата данни се реализира чрез стандартни JDBC конектори, осигурявайки надежден достъп до MySQL базата, разположена в Docker среда.

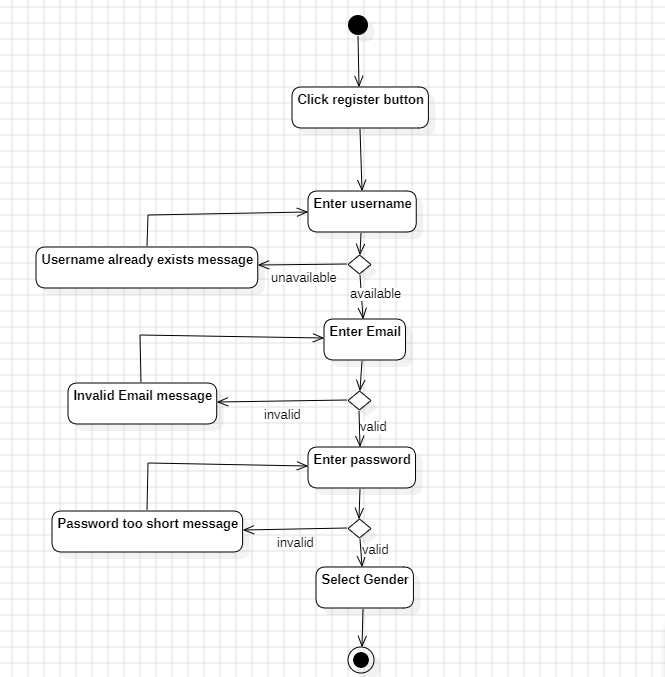
Цялостната архитектура е изградена така, че да бъде лесно разширяема с допълнителни външни услуги или микросървиси при нужда.

# 4. Функционални изисквания към проекта

Функционалните изисквания описват какво трябва да прави приложението и как да се държи в различни ситуации. Те са основа за дизайна и разработката, като гарантират, че се отговаря на нуждите на потребителите. Добре дефинираните функционалности предотвратяват отклонения от целите на проекта и допринасят за създаването на ефективно и лесно за използване приложение.

## 4.1 Регистрация на потребител

Activity диаграмата по-долу показва стъпките, които се изпълняват при процеса на регистрация.

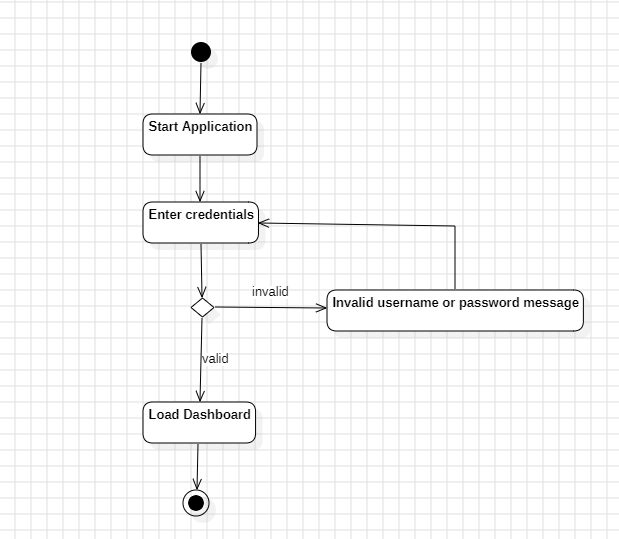


За да се регистрира успешно, потребителят трябва да въведе следната информация:

1) **Потребителско име** – уникално в рамките на системата;  
2) **Валиден имейл адрес**;  
3) **Парола** – с дължина от минимум 8 символа.

## 4.2 Включване на потребител

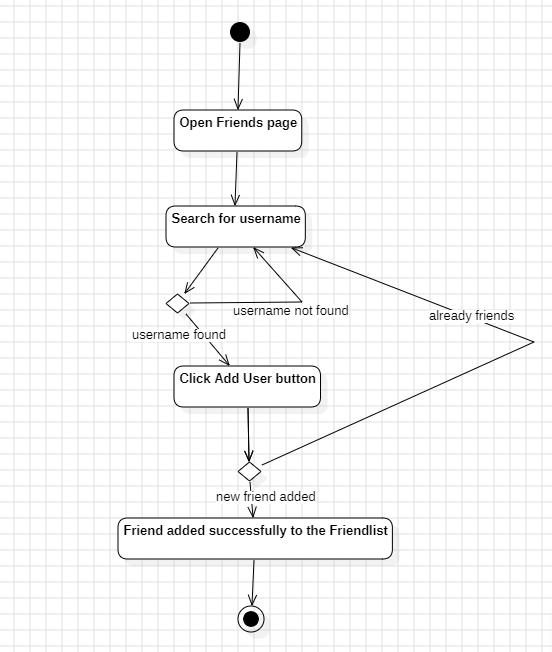
Включване в системата (автентикация) се осъществява съгласно следната диаграма:



Потребителят трябва предварително да е регистриран чрез формата за регистрация. При опит за вход в системата, той въвежда своите потребителско име и парола.Ако въведените данни са валидни, потребителят получава достъп до приложението и се зарежда началната страница - Dashboard.  
Ако данните са невалидни, системата извежда съобщение за грешка: *„Неправилни потребителско име или парола“*.

## 4.3 Добавяне на приятел

За описание на процеса по добавяне на приятел се използва следната Activity диаграма:



1) Избира се менюто „Friends“  
2) Извършва се търсене по потребителско име (username)  
3) Ако потребителят е намерен:

 • Ако вече е приятел – показва се съобщение: „Потребителят вече е в списъка с приятели“

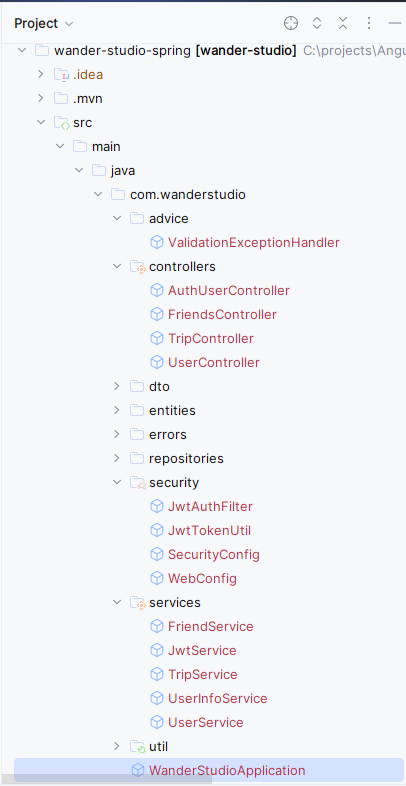
 • Ако не е – потребителят се добавя успешно и се показва съобщение: „Успешно добавен приятел“

4) Ако потребителят не е намерен – показва се съобщение: „Потребителят не е намерен“

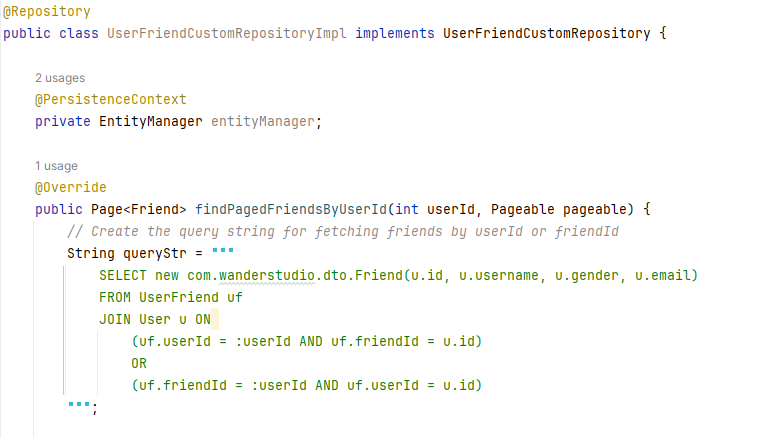
# 5. Архитектура на проекта

## 5.1 Back-end

По-долу е представен изглед на структурата на Back-end проекта:



Проектът използва добре установения Repository-Service-Controller архитектурен модел, който осигурява ясно разделение на отговорностите, лесна поддръжка и добра разширяемост.  
  
**Repository слой**  
За достъп до базата данни в проекта се използва Spring Data JPA, което осигурява високо ниво на абстракция при работа с персистентни данни. Стандартните CRUD операции се реализират чрез интерфейси, които разширяват JpaRepository. Това позволява автоматично генериране на заявки въз основа на именувани методи, без нужда от ръчно дефиниране на SQL или JPQL, което прави този подход изключително подходящ за по-семпли и често използвани операции.  
  
Въпреки това, при по-специфични и нетривиални заявки, които изискват сложна логика или нестандартна структура на данните, се налага използване на персонализирана имплементация на репозитори чрез EntityManager. Такъв е случаят с класа UserFriendCustomRepositoryImpl, който реализира ръчно дефинирана заявка с JPQL за извличане на списък от приятели на даден потребител, включително поддръжка на странициране (Pageable).



Причината за тази нужда произтича от начина, по който е моделирана връзката за приятелство в базата от данни:  
Таблицата user\_friend съдържа записи, представляващи двустранна връзка между двама потребители, като редът на стойностите в колоните user\_id и friend\_id няма значение. С други думи, двама потребители са приятели, ако тяхната двойка от ID-та се съдържа в таблицата — независимо дали един е user\_id, а другият friend\_id, или обратно.  
  
Тъй като тази логика не може лесно да бъде изразена с готовите методи на Spring Data JPA (които очакват еднозначни релации), се използва ръчно написана JPQL заявка, която обединява двете възможни посоки на връзката чрез OR условие. Това позволява гъвкаво и точно извличане на приятелствата, съобразено с модела на данните.  
  
**Service слой**Слой Service посредничи между контролерите и репозиториата. Той капсулира бизнес логиката и осигурява по-добра повторна употреба и разделяне на отговорностите. Всички зависимости се инжектират чрез @Autowired или конструктори.

**Controller слой**Контролерите (например TripController, UserController) отговарят за приемане на HTTP заявки, обработка на входни параметри и извикване на съответни методи от Service слоя. Те връщат отговори под формата на DTO обекти или ResponseEntity.  
  
**Обработка на грешки чрез аспектно-ориентиран подход**  
Класът ValidationExceptionHandler използва аспектно-ориентирания механизъм на Spring Framework чрез @ControllerAdvice, за да управлява глобално изключенията в уеб слоя.



Някои от основните методи в него са:

* handleValidationExceptions(MethodArgumentNotValidException ex)

Обработва грешки от невалидни входни данни (напр. при @Valid). Извлича всички полета с грешки и ги връща като ключ-стойност (име на поле – съобщение за грешка) със статус 400 BAD\_REQUEST.

* handleAllExceptions(Exception ex)

Улавя всички останали непредвидени грешки. Връща обобщено съобщение със статус 500 INTERNAL\_SERVER\_ERROR и записва пълните детайли в логовете.  
  
Използването на аспектно-ориентиран подход за обработка на грешки в приложението осигурява редица предимства. На първо място, той позволява централизирана и последователна обработка на изключения, без нужда от дублиране на логика във всеки отделен контролер. Това води до по-чист и поддържаем код, тъй като контролерите се фокусират единствено върху бизнес логиката. Допълнително, логването и диагностиката на грешки са значително подобрени благодарение на уеднаквения подход към обработка на изключенията. Клиентите на приложението получават унифицирани и предсказуеми отговори при грешки, което улеснява интеграцията и потребителското изживяване. В резултат на това архитектурата на системата е по-устойчива и ясно структурирана, с добре разделени отговорности между бизнес логика и техническа обработка на грешки.  
  
**Сигурност и достъп до защитени ресурси**Сигурността в приложението е изградена чрез употреба на JWT (JSON Web Token) и Spring Security. Вместо да използва сесии, системата разчита на статична автентикация (stateless authentication): след успешен вход потребителят получава JWT токен, който се съхранява в бразъра и се изпраща автоматично с всяка заявка към сървъра.

Токенът съдържа информация за самоличността на потребителя, което позволява сървърът да валидира заявката и да предостави достъп само до ресурсите, за които потребителят е оторизиран. Това осигурява едновременно гъвкав контрол на достъпа и по-добра мащабируемост на приложението.  
  
**Генериране и изпращане на JWT токен**Когато потребител се логне успешно, Back-end-ът генерира JWT токен, съдържащ уникалния идентификатор на потребителя (userId). Това става чрез JwtTokenUtil.generateToken(userId). Токенът е подписан със секретен ключ и има срок на валидност (например 30 минути). След това този токен се изпраща към фронтенда чрез HTTP cookie, маркирано с флаг HttpOnly, така че да не е достъпен чрез JavaScript.  
  
**Проверка на достъпа – JwtAuthFilter**Всяка входяща заявка към защитен ресурс преминава през филтъра JwtAuthFilter, който:

* Извлича токена от cookie с име "token"
* Валидира токена чрез JwtService.validateToken(...) и проверява съответствието с потребителя

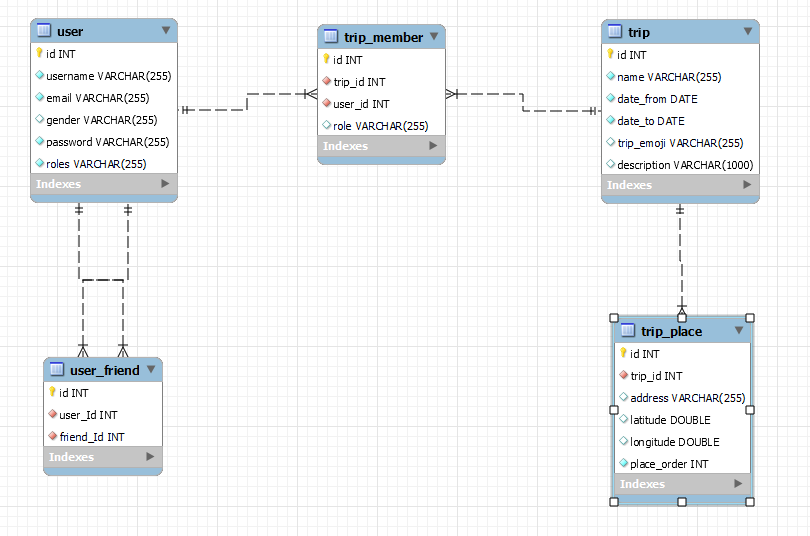
Ако всичко е валидно, създава UsernamePasswordAuthenticationToken и го задава в SecurityContextHolder, което удостоверява потребителя за текущия request. Така всички защитени ресурси могат да разчитат, че в контекста вече има автентифициран потребител.

**Конфигурация на сигурността**Класът SecurityConfig конфигурира сигурността на приложението, като задава основните правила и компоненти за автентикация и авторизация. Той дефинира, че приложението работи в безсесийна (stateless) среда, тоест сървърът не пази информация за потребителите между отделните заявки. Това означава, че всяка заявка е независима и носи цялата необходима информация за автентикация, което също намалява риска от атаки, свързани със сесии (например session fixation). Освен това, този подход е по-подходящ за съвременни SPA фронтенд приложения (като Angular на порт 4200), които комуникират с бекенда чрез REST API и изпращат JWT токена при всяка заявка.  
  
Основните моменти, които SecurityConfig настройва, са:

* Разрешаване на публичен достъп до определени ендпойнти като /auth/generateToken, /auth/addNewUser и /auth/logout, докато всички останали заявки изискват валидна автентикация.
* Използване на stateless сесийна политика
* Добавяне на собствен филтър за JWT автентикация (JwtAuthFilter)
* Дефиниране на AuthenticationProvider, който използва UserDetailsService и PasswordEncoder (BCrypt) за проверка на потребителските данни при вход.

## 5.2 Модел на базата данни

MySQL базата данни е разположена в Docker контейнер, което се счита за добра практика, тъй като осигурява изолирана, стандартизирана и лесно възпроизводима работна среда. Това позволява стартиране на базата без необходимост от локална инсталация и елиминира възможни конфликти с други проекти или версии. Контейнеризацията улеснява съвместната работа в екип чрез гарантиране на идентична конфигурация във всички среди, както и предоставя възможност за бързо създаване на временни или тестови инстанции без риск за продукционните данни.

По-долу е представена Entity-Relationship диаграма на базата данни:  
  


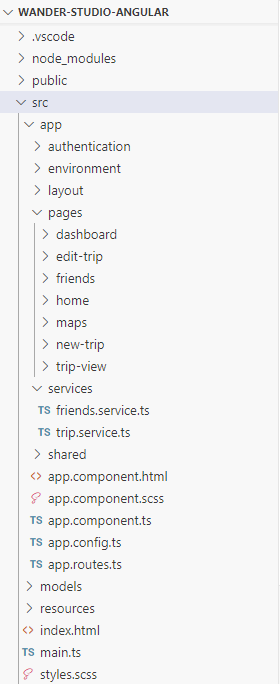
**User** — съхранява потребителите с полета за име, имейл, пол, парола и роли. **Trip** — съдържа информация за пътуванията: име, дати, описание и емоджи.  
**Trip\_Member** — свързва потребители с пътувания и роли (напр. организатор, участник).  
**Trip\_Place** — описва местата в дадено пътуване с адрес, координати и ред на посещение.

Тази структура позволява:

* Гъвкаво създаване на пътувания с множество участници
* Подредени маршрути със спирки
* Управление на приятелства и споделяне на пътувания между потребители

## 5.3 Front-end

По-долу е представен изглед на структурата на Front-end проекта:

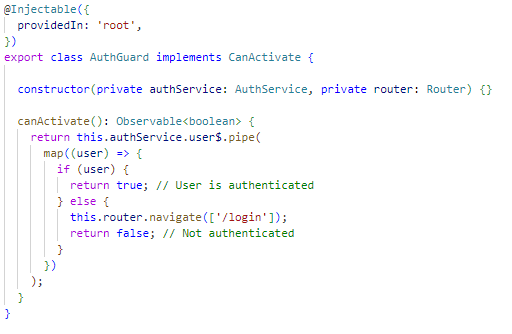


Angular проектът, който консумира Back-end ендпойнтите, е организиран по модулен и функционален начин с цел осигуряване на добра структура, повторна употреба на код и лесна поддръжка. Той включва няколко ключови елемента:  
  
**Процес по включване в системата**  
Процесът на логване в системата започва чрез извикване на метода login(...) в класа AuthService. Angular използва HttpClient, за да изпрати POST заявка към endpoint-а /auth/generateToken, като в тялото се подават потребителското име и паролата. Заявката съдържа опцията withCredentials: true, която позволява обмен на бисквитки между Front-end и Back-end, ако такива са нужни.  
  
След като сървърът върне отговор, съдържащ потребителски данни и срок на валидност на токена (expiresAt), информацията се съхранява в sessionStorage. Потребителят се записва и в BehaviorSubject, което позволява останалите компоненти в приложението, които са абонирани за user$, да реагират автоматично при промяна в статуса на потребителя (напр. логване или излизане).  
  
Освен това се стартира таймер чрез TokenTimerService, който следи изтичането на токена. Това осигурява възможност за автоматично излизане на потребителя или подновяване на сесията преди изтичане на валидността.



**Защита от неотовизиран достъп**  
Angular приложението използва AuthGuard, за да ограничи достъпа до защитени маршрути. Guard-ът проверява чрез AuthService дали има логнат потребител (чрез BehaviorSubject). Ако няма — потребителят се пренасочва към страницата за логин.

Допълнително, ErrorInterceptor прихваща HTTP грешки (напр. 401 Unauthorized) и също пренасочва към логин при невалиден токен или изтекла сесия.



**Визуална организация на проекта**Потребителският интерфейс е разделен на две колони:

* Лява колона (Side Menu): Постоянно видимо странично меню, съдържащо логото, информация за потребителя (снимка, име, роля, имейл), връзки към основните страници (Dashboard, New Trip, Friends), таймер на сесията и бутон за изход.
* Дясна колона (Съдържание): В тази зона чрез <router-outlet> динамично се зарежда съдържанието на избраната страница, без презареждане на цялото приложение.

Тази структура осигурява удобно навигиране и ясно разграничение между навигация и съдържание.  
  
**Интеграция с Google Maps**Проектът използва Google Maps JavaScript API чрез Angular компонентите @angular/google-maps и google.maps.places за осигуряване на визуална и функционална интеграция с карта. Основните функционалности включват:

Подсказка за име на място: Чрез PlaceAutocompleteComponent се осъществява търсене по адрес или име на място. Използва се Google Places Autocomplete, който при избор връща информация за избраното място – адрес, координати, име, икона и снимка (ако е налична).

Визуализация на карта: В MapDisplayComponent се използва компонентът <google-map>, който показва карта с възможност за добавяне на маркери върху избрани локации. Позицията на картата се променя динамично според избраното място чрез метода panTo(...).

# 6. Използване на Google Maps API

За правилното функциониране на приложението, особено за картови визуализации и търсене на локации, е необходим Google Maps API ключ. Този ключ се използва за достъп до услугите на Google Maps Platform и управление на потреблението.

**1. Управление на API ключ при ниско потребление**В случай на развойна дейност, тестване или малък брой потребители, един API ключ е напълно достатъчен. При ниско потребление, разходите обикновено се покриват от месечния безплатен кредит, предоставен от Google. Важно е редовно следене на потреблението през Google Cloud Console.  
  
**2. Управление на API ключ при високо потребление**При увеличаване на броя на потребителите или интензивно използване на функциите, базирани на Google Maps API (напр. множество заявки за локации, маршрутизации), е възможно месечният безплатен кредит да бъде надвишен. В такъв случай за ползване на приложението трябва да се въведе минимална потребитлска такса или абонамент.  
  
**3. Интеграция в други софтуерни продукти**Ако настоящото приложение е част от по-голяма софтуерна система или друга софтуерна разработка, е възможно даден модул или друг екип да предостави собствен Google Maps API ключ. Тази практика позволява по-голям контрол върху разходите, сигурността и ресурсите за всеки отделен компонент на системата.

# 7. Бъдещи цели

**1. Разширяване на функционалността**

* Добавяне на възможност за качване и разглеждане на снимки към пътувания и профили
* Въвеждане на система за коментари и оценки към пътувания и места
* Възможност за редакция на потребителски профил — промяна на данни, парола и аватар

**2. Осигуряване на качество**

* Имплементиране на модулни тестове с JUnit за Back-end логиката
* Добавяне на end-to-end тестове с Selenium за проверка на ключови потребителски сценарии
* Настройка на CI/CD pipeline за автоматично тестване при всяка промяна в кода