Toilet Finder

Документация

Атанас Ружинов 11В

# Въведение

В съвременното градско ежедневие, намирането на обществена тоалетна често се оказва по-трудно, отколкото изглежда. Особено за туристи, хора с увреждания, родители с малки деца или просто в непозната среда, липсата на актуална и достоверна информация за локацията и състоянието на обществените тоалетни може да предизвика сериозен дискомфорт. Точно с тази цел е създаден проектът Toilet Finder.

**Toilet Finder** е уеб базирано приложение с отворена и общностно поддържана база от данни, в която потребителите могат да намират, добавят, оценяват и описват обществени тоалетни, които са използвали. Проектът цели не само да улесни ежедневието на хората, но и да насърчи култура на споделяне на полезна информация в полза на обществото.

Приложението е разработено с помощта на **Python** и уеб рамката **Flask**, с използване на **SQLite** база данни и **SQLAlchemy ORM**. За автоматизация на разработката и внедряване на нови версии е използвана система за непрекъсната интеграция (**Jenkins**) и контейнеризация чрез **Docker**. Чрез тези технологии проектът постига гъвкавост, лесна поддръжка и възможност за бързо надграждане в бъдеще.

Настоящата документация има за цел да представи структурата, функционалността и технологичната реализация на приложението. В нея ще бъдат разгледани основните потребителски сценарии, архитектурата на системата и процесът на разработка.

# Функционалности на приложението

Приложението **Toilet Finder** предоставя на потребителите удобен начин да намират и споделят информация за обществени тоалетни. То е разработено с мисъл за достъпност, бързина и практичност, както за ежедневна употреба, така и в ситуации, при които бързото намиране на тоалетна е наложително. Приложението е с напълно responsive дизайн и може да се използва безпроблемно чрез браузър на мобилно устройство. Основните функционалности са групирани по роли и действия, които потребителите могат да извършват.

* Търсене на обществени тоалетни
  + Потребителят може да прегледа списък с тоалетни в системата.
  + На интегрираната карта може да ги види визуално по местоположение.
  + Всяка тоалетна съдържа информация за:
    - Географски координати
    - Кратко описание
    - Ниво на чистота (по скала от 1 до 5)
    - Достъпност за хора с увреждания
    - Наличност на тоалетна хартия
  + В зависимост от нивото на чистота, иконката на тоалетната на картата има различен цвят, по следната система:
    - Оценка 1 (Много мръсна) - Черна
    - Оценка 2 - Червена
    - Оценка 3 (Средно чиста) - Светлосиня
    - Оценка 4 - Лилава
    - Оценка 5 (Много чиста) - Златна
* Преглед на детайли за конкретна тоалетна
  + Всеки обект има индивидуална страница с детайлна информация.
  + Показват се оценките и коментарите, оставени от други потребители.
  + Пресмятат се обобщени стойности за чистота, достъпност и наличност на тоалетна хартия въз основа на всички ревюта.
* Добавяне на нова тоалетна
  + Всеки регистриран потребител може да добави нов обект чрез форма.
  + Полетата включват: координати, описание, достъпност, наличност на хартия и първоначална оценка за чистота.
  + Валидация се извършва за географски координати, текстови полета и рейтинг.
* Оценяване и коментиране (ревюта)
  + Всеки регистриран потребител може да остави оценка и коментар за съществуваща тоалетна.
  + Формата включва:
    - Чекбокси за достъпност и хартия
    - Възможност за докладване, че тоалетната вече не съществува (или е недостъпна/затворена)
    - Скала за чистота (1 до 5)
    - Свободен текстов коментар (до 200 символа)
  + Всеки ревю запис се свързва с потребителя и обекта.
* Управление на сесия и профил - потребителите могат да:
  + Създадат профил (регистрация с email, потребителско име и парола)
  + Влязат в системата
  + Излязат от акаунта

# Архитектура на приложението

## 3.1 MVC архитектура

Toilet Finder следва класическия MVC (Model–View–Controller) архитектурен модел, който разделя логиката на приложението на три ясно разграничени слоя:

* Model - моделите са представени чрез **SQLAlchemy ORM** и отговарят за бизнес логиката и връзката с базата данни. Те съдържат структурата на данните, валидации и взаимовръзки между обектите. Основните модели са:
  + User – съхранява информация за потребителите.
  + Toilet – представя обществена тоалетна.
  + Review – оценка и коментар, свързани с тоалетна и потребител.
* View - изгледната част използва **Jinja2** шаблони, намиращи се в директорията templates/ . Те визуализират **HTML** страници, които се генерират динамично от сървъра в зависимост от състоянието на данните.
* Controller - контролерите са реализирани чрез **Flask** маршрутите (@app.route ). Те приемат заявки от потребителя, обработват данните (напр. чрез формуляри), взаимодействат с моделите и връщат изглед (View) или **JSON** отговор при API повиквания. Примери:
  + /signup – контролира регистрацията на нов потребител.
  + /add\_toilet – обработва формата за добавяне на тоалетна.
  + /api/toilets – предоставя **JSON** списък на всички тоалетни (за frontend или мобилни клиенти).

## 3.2 База данни и релации

Базата данни използва **SQLite**, подходяща за леки уеб приложения и локално тестване. Основните релации са:

* Един потребител може да добави много тоалетни ( User → Toilet ).
* Една тоалетна има множество ревюта ( Toilet → Review ).
* Всеки потребител може да оставя множество ревюта ( User → Review ).

## 3.3 REST API слой

Приложението включва базови REST API маршрути, които позволяват frontend клиент или мобилно приложение да получава данни в **JSON** формат. API е документирано чрез **Swagger UI** с помощта на **Flasgger**.

| **Ендпойнт** | **Метод** | **Описание** |
| --- | --- | --- |
| /api/toilets | GET | Връща списък с всички тоалетни |
| /api/toilet/<id> | GET | Връща детайли и ревюта за конкретна тоалетна |

## 3.4 Сесии - използват се стандартни Flask сесии със session[] , за да се управлява логването на потребителите.

# Security

## 4.1 CSRF (Cross-Site Request Forgery) защита

За предотвратяване на CSRF атаки, приложението използва разширението **Flask-WTF**, което осигурява **CSRF токен** във всяка **HTML** форма, генерирана от сървъра. Това означава, че всяка POST заявка към сървъра трябва да съдържа валиден токен, генериран от сесията на съответния потребител. Ако липсва токен или той не съвпада, заявката се счита за потенциално злонамерена и се отхвърля. Това значително ограничава възможността злонамерен скрипт или външен сайт да изпълни неоторизирани действия от името на логнат потребител. CSRF защитата е активна за всички **HTML** форми по подразбиране, като API маршрутите (напр. /api/toilets ) са експлицитно освободени от CSRF проверка чрез декоратора @csrf.exempt , за да могат да бъдат използвани от frontend или външни клиенти, които не използват **HTML** форми.

## 4.2 XSS (Cross-Site Scripting) защита

Защитата срещу XSS атаки (вкарване на зловреден **JavaScript** код през входове от потребителя) се реализира чрез:

1. Автоматично ескейпване на **HTML** в шаблоните: **Flask** използва **Jinja2**, която по подразбиране ескейпва съдържанието в {{ }} блоковете. Това означава, че ако потребител въведе например <script>alert('XSS')</script> в коментар или описание, то ще бъде визуализирано като обикновен текст, а не изпълнено като **JavaScript**.
2. Изрично ескейпване при обработка на потребителски вход: Във app.py , всички текстови полета като описание и коментари преминават през функцията escape() от **MarkupSafe** – допълнителна защита, която преобразува потенциално опасен **HTML** в безвреден текст.

Пример:

comment = escape(request.form.get('comment', '').strip())

## 4.3 **SQL Injection** защита

Приложението **Toilet Finder** използва **SQLAlchemy** – обектно-релационен модел (ORM), който предоставя висок слой на абстракция над базата от данни. Вместо директни **SQL** заявки с низове, всички операции по създаване, четене, редактиране и изтриване на данни се извършват чрез обекти и методи, които автоматично управляват параметрите и ги ескейпват правилно. Това елиминира риска от **SQL injection**, тъй като входът от потребителя никога не се използва директно в **SQL низове**.

Например, заявка като:

user = User.query.filter\_by(username=username).first()

използва параметризирана форма под капака и не допуска изпълнение на зловреден **SQL** код, дори ако стойността на username е опит за инжекция. Допълнително, всякакъв вход се валидира и почиства с регулярни изрази и strip() преди да бъде използван, което добавя още един защитен слой.

# Използвани технологии и инструменти

Проектът **Toilet Finder** използва съвременен и олекотен технологичен стек, подходящ за уеб приложения с REST API функционалност и потенциал за разширение. При разработката са използвани доказани библиотеки и инструменти, които осигуряват стабилност, сигурност и лесна поддръжка.

Основната логика на приложението е изградена с помощта на **Python 3** и уеб рамката **Flask** – минималистична, но разширяема платформа, която позволява бързо създаване на уеб приложения с ясно разделение между логика, изгледи и модели. За работата с базата данни е използвана **SQLAlchemy** – ORM библиотека, която позволява дефиниране на модели чрез класове, автоматично генериране на заявки и защита срещу **SQL инжекции**.

Базата данни е реализирана с **SQLite** – лека и вградена релационна база, която не изисква отделен сървър и е подходяща за прототипи и малки до средни проекти. Ако приложението бъде разширено, същият код може лесно да бъде мигриран към **PostgreSQL** например.

Потребителският интерфейс е реализиран с помощта на **Jinja2** шаблони и базови **HTML/CSS** компоненти, което го прави съвместим с всички съвременни браузъри. За визуализация на картата се използва **JavaScript** библиотеката **Leaflet** и **OpenStreetMap**.

Проектът използва **Flasgger**, библиотека за автоматично генериране на **Swagger** документация. Това позволява лесна визуализация и тестване на API-то чрез уеб интерфейс на /apidocs/ .

За unit тестове е използван **Python** framework-ът **PyTest**.

За автоматизация на процесите по тестване, изграждане и внедряване е използван **Jenkins** – сървър за continuous integration & deployment (CI/CD), който изпълнява предварително дефинирани стъпки при всяко обновяване на кода. Приложението е контейнеризирано с **Docker**, което гарантира, че то ще работи еднакво на всяка среда, без зависимости от конкретни локални настройки.

# Continuous Integration & Deployment

Процесите на непрекъсната интеграция (CI) и непрекъснато внедряване (CD) са изключително важни за гарантиране на качеството, стабилността и бързото развитие на уеб приложения. В **Toilet Finder** е внедрен базов CI/CD pipeline чрез **Jenkins**, който автоматизира ключови етапи от жизнения цикъл на разработката: тестване, изграждане и внедряване на нови версии.

## 6.1 Контейнеризация с **Docker**

Цялото приложение е контейнеризирано с помощта на **Docker**, като се използва персонализиран Dockerfile, който дефинира средата за изпълнение. Това гарантира, че приложението работи еднакво независимо от хост операционната система. Контейнерът включва:

* Минимална **Python** среда ( python:3.10-slim )
* Инсталация на всички зависимости от requirements.txt
* Стартиране на **Flask** приложението на порт 5000

Контейнеризацията улеснява тестването, споделянето и разгръщането в продукционна среда.

## 6.2 **Jenkins** pipeline

За автоматизация на процесите по интеграция и внедряване се използва **Jenkins**, инсталиран в **Docker** контейнер. **Jenkins** сървъра е expose-нат чрез **ngrok**. В основната директория на проекта е добавен Jenkinsfile, който дефинира pipeline с няколко стъпки:

* Pull на кода от **GitHub**
* Изграждане на **Docker** image на приложението
* Изпълнение на **PyTest** тестовете
* Качване на **Docker** image-a в **Docker Hub**
* Почистване на временни ресурси

Този pipeline се активира при всеки push в **GitHub** репото. За целта се изполва **GitHub webhook**, свързан към **Jenkins**.

Този процес позволява на всеки да вижда резултата от последните промени, да валидира, че приложението може да бъде изградено, и при нужда – да го деплойне на сървър или в облачна среда.

# Заключение

Проектът **Toilet Finder** демонстрира как с помощта на съвременни уеб технологии може да бъде създадено полезно, обществено-ориентирано приложение, което решава реален проблем – бързото намиране на обществена тоалетна. Благодарение на добре структурирана архитектура, REST API, сигурност на данните и автоматизиран CI/CD процес, системата е стабилна, мащабируема и лесна за надграждане.

Разделянето на отговорностите чрез MVC модела, използването на ORM за работа с базата данни, както и наличието на API документация със **Swagger** я правят подходяща както за крайни потребители, така и за интеграция с други приложения. В бъдеще проектът може да бъде разширен с нови функционалности, визуални подобрения и мобилна версия (въпреки че и в момента то е напълно пълноценно използваемо през браузър на мобилно устройство), което допълнително ще увеличи неговата полезност и обхват.