# A logo of a university AI-generated content may be incorrect.

Datenbanken und Web-Techniken

Project Task Summer Semester 2025

Name: Farid Mammadov

Study course: Web Engineering

Matriculation number: 762696

Chemnitz, 2025

Contents

[1](#_Toc202385962)

[1. Introduction 2](#_Toc202385963)

[1.1. Background and Motivation 2](#_Toc202385964)

[2. Implementation and Utilized Technologies 2](#_Toc202385965)

[2.1. Backend Implementation 2](#_Toc202385966)

[2.1.1. Database Models 2](#_Toc202385967)

[2.1.2. API Endpoints 3](#_Toc202385968)

[2.1.3. Business Logic 4](#_Toc202385969)

[2.2. Frontend Implementation 5](#_Toc202385970)

[2.2.1. Page Structure & Routing 5](#_Toc202385971)

[2.2.2. UI Components 6](#_Toc202385972)

[2.2.3. API Integration 8](#_Toc202385973)

[2.3. Database Technology Choice 11](#_Toc202385974)

[2.3.1 Database Configuration 11](#_Toc202385975)

[2.3.2 Database Schema and Models 12](#_Toc202385976)

[3. API Design and Documentation (Web Service API) 12](#_Toc202385977)

[3.1 Overview 12](#_Toc202385978)

[3.2 API Endpoints Overview 13](#_Toc202385979)

[3.3 Endpoint Descriptions and Code Examples 13](#_Toc202385980)

[3.3.1 User Registration (api/register) 13](#_Toc202385981)

[3.3.2 Login (/api/login/) 14](#_Toc202385982)

[3.3.3 Get User Profile (/api/profile/) 14](#_Toc202385983)

[3.3.4 Update User Profile (api/profile/update) 14](#_Toc202385984)

[3.3.5 Get All Places (/api/places/) 15](#_Toc202385985)

[3.3.6 Save Place to Favorites (/api/save\_place/) 15](#_Toc202385986)

[3.3.7 Get User’s Saved Places (/api/saved\_places/) 15](#_Toc202385987)

[3.3.8 Leaderboard (/api/leaderboard/) 16](#_Toc202385988)

[4. Manual Testing & Test Cases 16](#_Toc202385989)

[4.1 Automated Tests 17](#_Toc202385990)

[4.2. Bug Fixes and Improvements 17](#_Toc202385991)

[5. Conclusion 17](#_Toc202385992)

[6. References 18](#_Toc202385993)

# **1. Introduction**

## **1.1. Background and Motivation**

As Chemnitz was selected as the European Capital of Culture 2025, It gives us a great idea to create a platform, which offers a unique opportunity for us to showcase both modern attractions and also historical places through innovative web technologies

This term paper shows the design and implementation of **ChemBuzz**, an interactive website that was built for tourists of our city to see and discover cultural places both Chemnitz and the Saxony state. In this project you can find database design, whole web development, RESTful APIs and modern frameworks.

# **2. Implementation and Utilized Technologies**

To ensure a high-quality user experience, modern and robust technologies were used throughout the development of ChemBuzz.

For the frontend part, React was used to build dynamic and reusable interfaces, hooks such as useState, useEffect and useContext for state management. React Router was used to navigate through the pages, while axios handles HTTP requests to the backend. Exploring cultural sites is done by React Leaflet which provides interactive mapping features. CSS modules are used for styling pages smoothly and keeping styles organized and maintainable.

Because of its widespread adoption and active community React was chosen for the frontend part of the project. React Router and axios libraries are also well-designed within React ecosystem and makes development smooth and further streamlining.

For the backend part Django server implemented with its ORM for database management and Django REST framework is used to create secure RESTful APIs. SimpleJWT is used for authentication which enables secure and stateless user sessions by JSON Web Tokens. User registration, authentication, profile management and also handling saved places is managed on the backend.

PostgreSQL integrated via Django’s ORM is used for database part. Its reliability, scalability and its performance are the main factors when it comes to choosing the right database model. PostgreSQL handles complex queries and data integrity which makes it a perfect choice for production environments. In database models like User, SavedPlace and Products are saved and stored.

## **2.1. Backend Implementation**

### **2.1.1. Database Models**

Django ORM is built to manage the database. The system contains models for user profiles, saved places (favorites), and products (for the shop feature). The main models are:

**User & Profile Model**:

class SavedPlace(models.Model):

user = models.ForeignKey(User, on\_delete=models.CASCADE)

name = models.CharField(max\_length=255)

address = models.CharField(max\_length=255, blank=True)

category = models.CharField(max\_length=100, blank=True)

image = models.URLField(blank=True)

website = models.URLField(blank=True)

phone = models.CharField(max\_length=30, blank=True)

def \_\_str\_\_(self):

return f"{self.name} ({self.user.username})"

**CulturalSite Model:**

class CulturalSite(models.Model):

    name = models.CharField(max\_length=255)

    category = models.CharField(max\_length=100, blank=True, null=True)

    tourism = models.CharField(max\_length=100, blank=True, null=True)

    amenity = models.CharField(max\_length=100, blank=True, null=True)

    city = models.CharField(max\_length=100, blank=True, null=True)

    postcode = models.CharField(max\_length=20, blank=True, null=True)

    street = models.CharField(max\_length=200, blank=True, null=True)

    district = models.CharField(max\_length=100, blank=True, null=True)

    website = models.URLField(blank=True, null=True)

    phone = models.CharField(max\_length=40, blank=True, null=True)

    image = models.URLField(blank=True, null=True)

    lat = models.FloatField()

    lng = models.FloatField()

    def \_\_str\_\_(self):

        return f"{self.name} ({self.category or self.tourism or self.amenity})"

### **2.1.2. API Endpoints**

The Django REST Framework (DRF) is used to create RESTful endpoints for authentication, profile management, and handling user favorites. The key API endpoints are:

**Example API View (views.py):**

from rest\_framework.decorators import api\_view, permission\_classes

from rest\_framework.permissions import IsAuthenticated

from rest\_framework.response import Response

@api\_view(['POST'])

@permission\_classes([IsAuthenticated])

def save\_place(request):

user = request.user

data = request.data

place = SavedPlace.objects.create(

user=user,

name=data['name'],

address=data.get('address', ''),

category=data.get('category', ''),

image=data.get('image', ''),

website=data.get('website', ''),

phone=data.get('phone', '')

)

return Response({"success": True, "id": place.id})

### **2.1.3. Business Logic**

This layer ensures secure authentication permission handling and consistent data validation. The main rules are:

* **Registration & Authentication:**

Every user must register with unique emails and for stateless authentication and therefore backend’ s secure management SimpleJWT tokens used.

* **Profile Updates:**  
  Only authenticated users can update their own profile. Uploaded images are validated and saved in the media folder inside backend.
* **Saving Places:**  
  Users are able to save their favorites inside database and also give reviews afterwards.

**User Registration:**

from rest\_framework.decorators import api\_view, parser\_classes, permission\_classes

from rest\_framework.parsers import MultiPartParser, FormParser

from rest\_framework.permissions import AllowAny

from rest\_framework.response import Response

from rest\_framework import status

from .serializers import RegisterSerializer

@api\_view(['POST'])

@parser\_classes([MultiPartParser, FormParser])

@permission\_classes([AllowAny])

def register\_user(request):

serializer = RegisterSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response({'message': 'User registered successfully'}, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

## **2.2. Frontend Implementation**

### **2.2.1. Page Structure & Routing**

The frontend uses **React.js** and **React Router** for navigation between pages such as Home, Explore, Login, Register, Profile, Update Profile, Shop, and About.

App.jsx

import React, { useState } from 'react';

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import Home from './pages/Home/Home';

import Explore from './pages/Explore/Explore';

import Login from './pages/Login/Login';

import Register from './pages/Register/Register';

import Profile from './pages/Profile/Profile';

import About from './pages/About/About';

import Update from './pages/Update/UpdateProfile';

import Competition from './pages/Competition/Competition';

import Shop from './pages/Shop/Shop';

const App = () => {

return (

<Router>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/explore" element={<Explore />} />

<Route path="/login" element={<Login />} />

<Route path="/register" element={<Register />} />

<Route path="/profile" element={<Profile />} />

<Route path="/about" element={<About />} />

<Route path="/update" element={<Update />} />

<Route path="/competition" element={<Competition />} />

<Route path="/shop" element={<Shop />} />

</Routes>

</Router>

);

};

export default App;

### **2.2.2. UI Components**

All major sections are implemented as reusable React components. For example:

// src/pages/Home/Home.jsx

import React from 'react';

import Hero from '../../Components/Hero/Hero';

import Footer from '../../Components/Footer/Footer';

import Navbar from '../../Components/Navbar/Navbar';

import './Home.css';

const Home = () => {

return (

<div className="home-page">

<Navbar/>

<Hero />

<Footer />

</div>

);

};

export default Home;

A close-up of a building

AI-generated content may be incorrect.

*Figure 1: Home Page with Navbar on top.*

In Explore page users can browse all the cultural sites of Chemnitz on an interactive map and by filtering categories the user can view detailed information about the places. Users can add places to favorites and calculate routes in each time range.

A screenshot of a map

AI-generated content may be incorrect.

*Figure 2: Explore Page with Map and Filter Panel.*

In About page user can find the brief history and cultural places of Chemnitz. IT features the main achievements of the city and project’s mission presented in a visually appealing and modern layout.

A screenshot of a website

AI-generated content may be incorrect.

*Figure 3: About page with the history of Chemnitz.*

The Shop page shows future ChemBuzz merchandise like t-shirts, hoodies, and some other accessories. Even though the shop is not open for now, users can still browse by category and view all the products with their details.

A screenshot of a website

AI-generated content may be incorrect.

*Figure 4: Chemnitz Franchise’s Shop Page.*

### **2.2.3. API Integration**

The frontend communicates with the backend API using **fetch** or **axios**. Here’s how you handle login and profile fetching:

Login Example:

const handleLogin = async (e) => {

e.preventDefault();

try {

const response = await fetch('http://localhost:8000/api/login/', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ email, password }),

});

const data = await response.json();

if (response.ok && data.access) {

localStorage.setItem('token', data.access);

// Redirect or update UI

} else {

setError('Invalid credentials');

}

} catch (err) {

setError('Something went wrong. Please try again.');

}

};

A screenshot of a login screen

AI-generated content may be incorrect.

*Figure 5. Login Page*

User’s personal information such as name and picture location is shown on Profile page. The saved and viewed places from explore pages are seen on the front page. Users can also update their details, manage and remove saved places and change their profile image.

useEffect(() => {

const token = localStorage.getItem('token');

fetch('http://localhost:8000/api/profile/', {

headers: { Authorization: `Bearer ${token}` },

})

.then(res => res.json())

.then(data => {

setProfile(data);

})

.catch(() => setError('Failed to load profile'));

}, []);

A screenshot of a website

AI-generated content may be incorrect.

*Figure 6. Profile Page* The Visited part of the profile page provides user with an overview of the places saved and after visited. By having a review button on the visited page, you can send a review about the place and the user’s genuine review of the place.

A screenshot of a gallery

AI-generated content may be incorrect.

*Figure 7. Profile Page with Visited Places*

**Dashboard/Leaderboard Page:** A cool interactive dashboard page shows the ranking of allusers by their total number of places visited. Users can track their position in the community and see top explorers with badges and achievements earned. This approach gives motivation to all the users to discover more places and potentially climb on the rankings.

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 8: Competition page with all users listed by ranking*

## **2.3. Database Technology Choice**

For this project, **PostgreSQL** was selected as the main database due to its reliability, performance, and seamless integration with Django’s ORM. PostgreSQL is well-suited for modern web applications that require robust data management, support for advanced data types, and transactional integrity.

### **2.3.1 Database Configuration**

The database connection is configured in Django’s settings.py file as shown below:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'postgres',

'USER': 'postgres',

'PASSWORD': '\_your\_password\_',

'HOST': 'localhost',

'PORT': '5432',

}

}

After any changes to the models, you must run Django migrations:

python manage.py makemigrations

python manage.py migrate

### **2.3.2 Database Schema and Models**

Django models define the structure of the data in Python classes. Below are simplified versions of the main models used in this project:

User Profile Model (Profile):

from django.contrib.auth.models import User

from django.db import models

class Profile(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

bio = models.TextField(blank=True)

phone = models.CharField(max\_length=20, blank=True)

country = models.CharField(max\_length=50, blank=True)

gender = models.CharField(max\_length=10, blank=True)

dob = models.DateField(null=True, blank=True)

profile\_image = models.ImageField(upload\_to='profiles/', blank=True, null=True)

def \_\_str\_\_(self):

return self.user.username

Saved Places / Favorites Model:

class SavedPlace(models.Model):

user = models.ForeignKey(User, on\_delete=models.CASCADE)

place = models.ForeignKey(Place, on\_delete=models.CASCADE)

saved\_at = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self):

return f"{self.user.username} saved {self.place.name}"

# **3. API Design and Documentation (Web Service API)**

## **3.1 Overview**

The communication between the frontend and backend of this project is realized through a RESTful Web Service API, developed using Django REST Framework. The API enables secure data exchange by employing JSON Web Tokens (JWT) for authentication and authorization. All endpoints are designed following REST principles, supporting CRUD operations for users and cultural places.

The API allows the frontend (React) to perform actions such as user registration, login, profile management, retrieving available places, saving favorites, and more. CORS middleware is enabled to allow cross-origin requests during development.

## **3.2 API Endpoints Overview**

The following table summarizes the main API endpoints, their HTTP methods, authentication requirements, and purposes:

Table 1. Lists of API Endpoints

|  |  |  |  |
| --- | --- | --- | --- |
| **Endpoint** | **Method** | **Auth** | **Purpose** |
| /api/register/ | POST | No | User registration |
| /api/login/ | POST | No | Login, returns JWT token |
| /api/profile/ | GET | Yes | Get current user's profile |
| /api/profile/update/ | PUT | Yes | Update profile data |
| /api/places/ | GET | No | Get all available places |
| /api/save\_place/ | POST | Yes | Save a place to favorites |
| /api/saved\_places/ | GET | Yes | List user's saved places |
| /api/leaderboard/ | GET | No | Get top users by places visited |

## **3.3 Endpoint Descriptions and Code Examples**

Below, key endpoints are described in detail. Where useful, code snippets are included for clarity, following the instructor’s suggestion to present small code fragments.

### **3.3.1 User Registration (api/register)**

* **Method:** POST
* **Auth Required:** Yes
* **Request:** FormData with fields: username, email, password, gender, dob, country, profile\_image (optional)
* **Response:** Success message or error details

@api\_view(['POST'])

@parser\_classes([MultiPartParser, FormParser])

@permission\_classes([AllowAny])

def register\_user(request):

serializer = RegisterSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response({'message': 'User registered successfully'}, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

### **3.3.2 Login (/api/login/)**

* **Method:** POST
* **Auth Required:** Yes
* **Request:** List of places user has added to their profile
* **Response:** JWT access and refresh tokens

from rest\_framework\_simplejwt.views import TokenObtainPairView

urlpatterns = [

path('api/login/', TokenObtainPairView.as\_view(), name='token\_obtain\_pair'),

]

### **3.3.3 Get User Profile (/api/profile/)**

* **Method:** GET
* **Auth Required:** Yes (JWT Bearer token)
* **Response:** JSON with user info (name, email, country, gender, dob, profile\_image, etc.)

@api\_view(['GET'])

@permission\_classes([IsAuthenticated])

def profile\_view(request):

user = request.user

profile = user.profile # Assuming Profile is linked with OneToOneField to User

serializer = ProfileSerializer(profile)

return Response(serializer.data)

### **3.3.4 Update User Profile (api/profile/update)**

* **Method:** PUT
* **Auth Required:** Yes
* **Request:** FormData with fields to update
* **Response:** Success or error message

@api\_view(['PUT'])

@parser\_classes([MultiPartParser, FormParser])

@permission\_classes([IsAuthenticated])

def update\_profile(request):

user = request.user

profile = user.profile

serializer = ProfileSerializer(profile, data=request.data, partial=True)

if serializer.is\_valid():

serializer.save()

return Response({'message': 'Profile updated successfully'})

return Response(serializer.errors, status=400)

### **3.3.5 Get All Places (/api/places/)**

* **Method:** GET
* **Auth Required:** No
* **Response:** JSON list of all cultural places, with details such as name, type, address, category, image, etc.

@api\_view(['GET'])

@permission\_classes([AllowAny])

def list\_places(request):

places = Place.objects.all()

serializer = PlaceSerializer(places, many=True)

return Response(serializer.data)

### **3.3.6 Save Place to Favorites (/api/save\_place/)**

* **Method:** POST
* **Auth Required:** Yes
* **Response:**JSON body with place\_id
* **Response:** Success or error

@api\_view(['POST'])

@permission\_classes([IsAuthenticated])

def save\_place(request):

user = request.user

place\_id = request.data.get('place\_id')

# Save logic...

return Response({'message': 'Place saved!'})

### **3.3.7 Get User’s Saved Places (/api/saved\_places/)**

* **Method:** GET
* **Auth Required:** Yes
* **Response:** List of places user has added to their profile

@api\_view(['GET'])

@permission\_classes([IsAuthenticated])

def list\_saved\_places(request):

user = request.user

places = user.saved\_places.all()

serializer = PlaceSerializer(places, many=True)

return Response(serializer.data)

### **3.3.8 Leaderboard (/api/leaderboard/)**

* **Method:** GET
* **Auth Required:** No
* **Response:**  
  List of top users, sorted by the number of places visited/favorited

@api\_view(['GET'])

@permission\_classes([AllowAny])

def leaderboard(request):

users = User.objects.annotate(

total=Count('saved\_places')

).order\_by('-total')[:10]

data = [{'name': u.username, 'total': u.total} for u in users]

return Response(data)

# **4. Manual Testing & Test Cases**

During the development of ChemBuzz, comprehensive manual testing was conducted for both backend and frontend functionalities. Key test cases included:

* **User Registration:** Checked successful registration, potential invalid passwords and duplicate emails.
* **User Login/Logout:** Checking whether verified login is valid or invalid and JWT token storage.
* **Profile Update:** Updates on Profile checked and saved correctly
* **Explore Page:** Tested if map rendering, filtering and search and route calculation are working or not.
* **Favorites (Saved Places):** Confirmed places could be added/removed to/from favorites and were shown on the Profile page.
* **API Endpoints:** /api/places/, /api/save\_place/, /api/saved\_places/, and other endpoints were personally verified using Postman.
* **Mobile Responsiveness:** Checking if mobile version works smoothly for user’s experience.

## **4.1 Automated Tests**

Due to the project timeline and scope, automated testing was limited. However, Django’s built-in test framework was used to run basic unit tests for API endpoints (e.g., registration and login).

Example (Django test case for registration):

from django.urls import reverse

from rest\_framework.test import APITestCase

from rest\_framework import status

class UserRegistrationAPITest(APITestCase):

def test\_register\_user(self):

url = reverse('register')

data = {

"username": "testuser",

"email": "testuser@example.com",

"password": "TestPassword123!"

}

response = self.client.post(url, data)

self.assertEqual(response.status\_code, status.HTTP\_201\_CREATED)

self.assertIn('message', response.data)

## **4.2. Bug Fixes and Improvements**

While developing the website, I have faced bugs and tried and finally resolved them.

* Issues about image uploads were fixed.
* Authentication, specifically token expiration.
* Map display and filtering categories were improved and upgraded.
* Navbar for both mobile versions was fixed using the hamburger icon.

Regular testing and bug fixing contributed to a more stable and user-friendly application.

# **5. Conclusion**

The ChemBuzz project demonstrates a complete version of a modern web application, both for requirements and deployment. Using Django, Django REST Framework, React, and its libraries enables creating a secure, responsive platform for users to explore cultural places in Chemnitz.

Main objectives such as profile management, browsing on a map, user authentication, and place saving were implemented successfully. Problems such as managing authentication and handling file uploads were fixed while creating mobile responsiveness. Continuous testing allowed us to spot the main problems on the website.

Overall, the project gave me great experience in full-stack development and a chance to explore a lot of technical and functional stuff.

# **6. References**

1. Django Documentation. <https://docs.djangoproject.com/>
2. Django REST Framework Docs. <https://www.django-rest-framework.org/>
3. React Documentation. <https://react.dev/>
4. Leaflet.js Documentation. <https://leafletjs.com/>
5. Tailwind CSS Docs. <https://tailwindcss.com/>
6. Lucide React Icons. <https://lucide.dev/>
7. Stack Overflow. <https://stackoverflow.com/>
8. Icons. <https://www.flaticon.com/>
9. React Router. https://reactrouter.com/