# Tennis Court Booking Web Application Development

by

Suleyman Eminov

Submitted to the Department of Mathematics and Computer Science

School of Natural and Social Sciences

in partial fulfillment of the requirements

for the degree of Bachelor of Arts

Purchase College

State University of New York

May 2025

Sponsor: Knarik Tunyan

Second Reader: [[Enter Second Reader’s name here]]

Table of Contents

[Tennis Court Booking Web Application Development 1](#_Toc185023467)

[Abstract 3](#_Toc185023468)

[1. Introduction 4](#_Toc185023469)

[2. Literature Review 4](#_Toc185023470)

[2.1. History of Agile Methodology 4](#_Toc185023471)

[2.2. Continuous Integration and Continuous Delivery (CI/CD) 4](#_Toc185023472)

[2.3. Docker 4](#_Toc185023473)

[2.4. Tech Stack 5](#_Toc185023474)

[2.5. Features 5](#_Toc185023475)

[3. Design and Architecture 6](#_Toc185023476)

[3.1. System Architecture 6](#_Toc185023477)

[3.2. Frontend Design 7](#_Toc185023478)

[4. Development Process 7](#_Toc185023479)

[4.1. Agile Methodology 7](#_Toc185023480)

[4.2. Django and Django Rest Framework (DRF) 7](#_Toc185023481)

[4.3. React 7](#_Toc185023482)

[4.4. Integration 7](#_Toc185023483)

[4.5. JWT Authentication 8](#_Toc185023484)

[4.6. Database 10](#_Toc185023485)

[5. Testing 10](#_Toc185023486)

[6. Conclusion 10](#_Toc185023487)

[Bibliography 11](#_Toc185023488)

[Appendix 12](#_Toc185023489)

# Abstract

This research presents the design, development, and implementation of a full-stack web application for tennis court booking management. The system addresses the inefficiency of traditional call and email based court reservation systems by providing an online platform where clients can view availability and book courts remotely, while tennis clubs can effectively manage their facilities.

The application was developed using modern web technologies including Django and Django Rest Framework for the backend REST API, React for the frontend user interface, PostgreSQL for the database, and Docker for containerization. JSON Web Token (JWT) authentication was implemented to ensure secure access to the system.

Following Agile methodology principles with Scrum framework, the project was developed iteratively through a series of sprints. The system architecture was designed to be scalable and maintainable, utilizing a microservices approach with frontend and backend services containerized separately.

Key features implemented include user authentication and role-based access control, a court availability management system for club managers, a responsive booking system for clients, and a comprehensive manager dashboard. The application handles critical business logic including preventing double bookings, managing operating hours, and providing real-time availability information.

Extensive testing was conducted to ensure reliability and performance, with test cases covering user authentication, booking logic, and concurrency handling. The research proposes that this modern web application for tennis court bookings has the potential to significantly improve efficiency for both clients and club managers while providing a foundation for future enhancements. Actual efficiency improvements would need to be measured through real-world implementation and subsequent analysis.

**Keywords**: Django, Python, JavaScript, React, Docker, PostgreSQL, Web Development, REST API, JWT Authentication

# Introduction

To reserve a tennis court at a nearby tennis club, clients are currently required to contact the front desk and make a booking request. Unfortunately, there may be instances where the front desk is occupied, and the client's call goes unanswered. This presents an opportunity for the business to implement an online booking system, allowing clients to make reservations without relying on the front desk. This research aims to make this a reality. This research project aims to develop a full-stack web application that enables tennis clubs to manage their court time schedules and lets clients reserve a court remotely.

# Literature Review

The main question for this senior project is “What is the best way to develop a full-stack web application to enable tennis clubs to efficiently manage their court schedules and provide clients with a seamless remote reservation experience?”

## History of Agile Methodology

In 2001, a group of remarkable practitioners promoted an “Agile Manifesto.” The manifesto outlines the principles and values for developing software more flexibly and collaboratively, emphasizing customer satisfaction and continuous delivery. Since then, Agile methodologies have become increasingly popular in software development, with many teams adopting them as the standard approach. The Agile approach has been proven to increase productivity, improve quality, and reduce time-to-market for software products.

One of the popular frameworks of agile methodology is the Scrum methodology. It is an iterative and incremental approach to project management. Scrum breaks projects into manageable chunks called sprints which are time-boxed periods during which a team works on a set of specific tasks. This project will be using Scrum methodology to deliver functionality to the web app continuously.

## Continuous Integration and Continuous Delivery (CI/CD)

CI/CD is a set of best software development practices in which code changes are frequently and automatically built, tested, and deployed to production environments. This approach enables teams to deliver software quickly, reliably, and with minimal risk. By automating the build, test, and deployment processes, teams can reduce the time and effort required to release new features and fix bugs, while also improving the quality and stability of their software. One of the popular tools for CI/CD is known as GitHub Actions. Using GitHub Actions, we can define workflows that execute specific actions automatically whenever a push is made to a Git repository.

## Docker

Another important tool that will be used in this project is Docker. In software development, configuration discrepancies between your computer and the server on which your application is being run can lead to problems. For instance, you may have a different Python version, or additional packages installed on your computer that enable the application to run smoothly, while it would crash on the server. To circumvent these issues, it is necessary to ensure that everyone working on the project is using the same environment. Docker, a containerization software, can accomplish this by creating an isolated environment within your computer that can be standardized and shared among multiple collaborators and the server on which the site is hosted. It is worth noting that while Docker is similar to a virtual machine, they are distinct technologies. A virtual machine, such as the one used on GitHub Actions or AWS servers, is essentially an entire virtual computer with its own operating system, which can take up considerable space wherever it is running. Conversely, Docker operates by setting up a container within an existing computer, thereby occupying less space. (Adomnitei et al. *Web Portal Development with Different Cloud Containers: Docker vs. Kubernetes*)

## Tech Stack

The choice of using Django as the backend framework for this project is a wise one, as it is a powerful and flexible web framework that allows for quick and efficient development. Additionally, the use of React and JavaScript for the frontend will enable a seamless and intuitive user experience, as these technologies are known for their speed, interactivity, and scalability. Leveraging PostgreSQL as the database management system further enhances the capabilities of this tech stack, ensuring reliable and concurrent data management. Overall, this tech stack comprising Django, React, JavaScript, and PostgreSQL is a great choice for building a modern web application that is both robust and user-friendly.

## Features

Now, that we established the underlying tools that will be utilized in this project, let’s delve into the specifics of the web application. By the end of this project, we will have two parallel applications: one for businesses/admin and one for clients. The necessary features for the admin application are as follows:

* **Dashboard**: A centralized dashboard that displays real-time overviews of all the courts, reservations, and other pertinent information.
* **Court booking management**: The ability to add, edit, and delete courts, as well as set availability and schedule maintenance or other events.
* **Reservation management**: The ability to view, modify, and cancel reservations.
* **User management:** The ability to manage user accounts, including adding or removing users, setting permissions, and access levels.
* **Reporting and analytics**: The ability to generate reports and gather analytics on court usage, revenue, and other key metrics.

For the client application, the following features should be included:

* **Court search**: The ability to search for available courts based on location, date, time, and other criteria.
* **Court booking:** The ability to reserve a court for a specific date and time, and to view or modify existing reservations.
* **User account management**: The ability to create and manage user accounts, view past bookings, and update personal information.
* **Notification and reminders**: The ability to receive notifications and reminders about upcoming bookings, cancellations, and other important updates.

# Design and Architecture

## System Architecture

The project began by establishing the foundational development environment to ensure smooth development. Using Docker, a containerized setup was configured to standardize the development environment across various systems.

To begin implementing Docker in our project, we will create two Docker files: Dockerfile.frontend and Dockerfile.backend. These files will contain instructions for building the Docker image, detailing the libraries and binaries to be included in our container. Here is the Dockerfile.backend:

FROM python:3.12.6-slim

WORKDIR /app

COPY backend/ /app

RUN pip install --upgrade pip && \

pip install -r requirements.txt

CMD ["python", "manage.py","runserver","0.0.0.0:8000"]

Here is the Dockerfile.frontend:

FROM node:16

WORKDIR /app/frontend

COPY frontend/ /app/frontend

RUN npm install

CMD ["npm", "start"]

To create a new container for the Postgres database and connect it to the backend, we will use a feature called Docker Compose. This enables two different servers to run in separate containers while still being able to communicate with each other. We’ll specify this configuration in a YAML file named docker-compose.yml, as illustrated in Appendix A.

Next, we will start our services by executing the command “docker-compose up”. This command will initiate both of our servers in new Docker containers.

To view all active Docker containers, we can use the command “docker ps”. We can then identify the CONTAINER ID of the container we want to access and run “docker exec -it CONTAINER\_ID bash -l”. This command will take us into the directory we set up within the container, allowing us to execute our desired commands. We can exit the container by pressing CTRL-D.

## Frontend Design

To make the UI/UX appealing and user-friendly, this application uses Bootstrap, a popular and powerful front-end toolkit. Bootstrap provides pre-designed components and a responsive grid system, enabling developers to create consistent and attractive layouts efficiently.

To implement Bootstrap in our React app, we need to install it on our front-end server using npm. We need to run “npm install react-bootstrap bootstrap” in the front-end container. Then, we need to import the following code in our app.js file:

import 'bootstrap/dist/css/bootstrap.min.css';

Now, we can design our front-end using Bootstrap.

# Development Process

## Agile Methodology

This project was divided into Sprints. Notion’s Agile Project Management template was used to manage sprints, projects, and features.

* Sprint 1: Setting up the development environment with Docker and initializing the project repository on Github.
* Sprint 2: Implementing user authentication with JWT and creating database schema.
* Sprint 3: Developing Court Availability Management for managers.
  + Design the interface for clubs to add and manage court schedules.
  + Create a backend system to store and update court availability.
  + Develop APIs for querying available time slots.
* Sprint 4: Developing Court Booking System.
  + Users should be able to view available courts by date and time.
  + Implement booking functionality, ensuring real-time updates for availability.
  + Handle concurrent booking.
* Sprint 5: Dashboard views for managers and clients.

## Django and Django Rest Framework (DRF)

## React

The frontend of the Tennis Court Booking application is built using React, a popular JavaScript library for building user interfaces. React was chosen for its component-based architecture, which enables the development of reusable UI components, efficient state management, and a virtual DOM that optimizes rendering performance.

* + 1. Project Structure

## Integration

TODO: write about CORS, middleware

## JWT Authentication

A critical aspect of the Tennis Court Booking application is the implementation of a secure and stateless authentication system. This was achieved using JWT (JSON Web Token), a widely used protocol for managing user authentication in modern web applications.

JWT is an open standard (RFC 7519) for securely transmitting information between parties as a JSON object. It is compact, self-contained, and cryptographically signed, making it ideal for transmitting user credentials securely. The token contains three parts:

1. Header: Specifies the type of token (JWT) and the hashing algorithm (e.g., HS256).
2. Payload: Contains the claims, such as username, which is the information being transmitted.
3. Signature: Verifies the integrity of the token using a secret key.

Therefore, a JWT typically looks like the following: xxxxx.yyyyy.zzzzz.

Django Rest Framework utilizes the djangorestframework-simplejwt library to implement JWT-based authentication. The implementation is as follows:

1. Make sure the djangorestframework-simplejwt library is installed in the Django container. To install this, we can run “pip install djangorestframework-simplejwt” on the Django container.
2. Django setttings.py file needs to be updated to include JWT configurations:

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': [

'rest\_framework\_simplejwt.authentication.JWTAuthentication',

],

}

SIMPLE\_JWT = {

'ACCESS\_TOKEN\_LIFETIME': timedelta(minutes=15),

'REFRESH\_TOKEN\_LIFETIME': timedelta(days=1),

'ROTATE\_REFRESH\_TOKENS': True,

'BLACKLIST\_AFTER\_ROTATION': True

}

1. Two key endpoints were added to manage token creation and refreshing:
   1. /api/token/: Generates an access token and a refresh token upon successful login.
   2. /api/token/refresh: Provides a new access token using the refresh token.

On the react application, when a user registers or logs in to the application, access, and refresh tokens are initialized in the local storage. Here is an example from Login.js file:

const {data} = await axios.post("http://localhost:8000/api/token/", user,

{

headers:

{

"Content-Type": "application/json"},

withCredentials: true

});

// Initialize the access & refresh token in localstorage.

localStorage.clear();

localStorage.setItem('access\_token', data.access);

localStorage.setItem('refresh\_token', data.refresh);

axios.defaults.headers.common['Authorization'] = "Bearer " + data.access;

window.location.href = "/";

## Database

Postgres. ERD diagramA diagram of a function

Description automatically generated

Figure 1. Database ERD Diagram



# Testing

# Conclusion

The main conclusions of the research may be presented in a short Conclusions section, which may stand alone or form a subsection of a Discussion or Results and Discussion section.

List funding sources if any:

This work was supported by the NSS Undergraduate Research [grant number XX, YEAR].

You may also include acknowledgements

Acknowledgements: …

**References/Bibliography**

Please ensure that every reference cited in the text is also present in the reference list (and vice versa). Reference list should follow the standard reference style, such as APA, MLA, etc. (Strang, 2016)

*To create References follow steps*

Document Elements -> Bibliography -> Manage

OR

References -> Citations

# Bibliography

Elsevier. (2021, September 15). *Guide for Authors, Linear Algebra and its Applications* . Retrieved from https://www.elsevier.com/journals/linear-algebra-and-its-applications/0024-3795/guide-for-authors

Strang, G. (2016). *Introduction to Linear Algebra.* Wellesley, MA: Wellesley-Cambridge Press.

1. Nampala, Henry, and Mahad Kigenyi. “A Response Concurrent Web-Based Hotel Reservation System: A Case Study of Sports View Hotel.” *Kampala International University*, July 2011, [ir.kiu.ac.ug/items/1217c92f-4f2b-4106-9e5f-3f37db73cdb8](http://ir.kiu.ac.ug/items/1217c92f-4f2b-4106-9e5f-3f37db73cdb8).
2. Adomnitei, Cezar, et al. “Web Portal Development with Different Cloud Containers: Docker vs. Kubernetes.” *Purchase College Library*, 2021, [suny-pur.primo.exlibrisgroup.com/discovery/fulldisplay?docid=cdi\_doaj\_primary\_oai\_doaj\_org\_article\_29ce7c61706447d6a965bbd244e84e13&context=PC&vid=01SUNY\_PUR%3A01SUNY\_PUR&lang=en&search\_scope=MyInst\_and\_CI&adaptor=Primo%2BCentral&tab=Everything&query=any%2Ccontains%2Cdocker%2BAND%2Bweb%2Bdevelopment&offset=0](http://suny-pur.primo.exlibrisgroup.com/discovery/fulldisplay?docid=cdi_doaj_primary_oai_doaj_org_article_29ce7c61706447d6a965bbd244e84e13&context=PC&vid=01SUNY_PUR%3A01SUNY_PUR&lang=en&search_scope=MyInst_and_CI&adaptor=Primo%2BCentral&tab=Everything&query=any%2Ccontains%2Cdocker%2BAND%2Bweb%2Bdevelopment&offset=0).
3. Fireship. “100+ Docker Concepts You Need to Know.” *YouTube*, YouTube, 12 Mar. 2024, [www.youtube.com/watch?v=rIrNIzy6U\_g](http://www.youtube.com/watch?v=rIrNIzy6U_g).
4. C.J., Torrecilla-Salinas, et al. “Estimating, Planning and Managing Agile Web Development Projects under a Value-Based Perspective.” *ScienceDirect*, 2015, [www-sciencedirect-com.ezproxy.purchase.edu/science/article/pii/S0950584915000142](http://www-sciencedirect-com.ezproxy.purchase.edu/science/article/pii/S0950584915000142).
5. AnjalBam. “Dockerize a Django, React, and Postgres Application with Docker and Docker-Compose: By Anjal Bam.” *DEV Community*, DEV Community, 10 Sept. 2022, dev.to/anjalbam/dockerize-a-django-react-and-postgres-application-with-docker-and-docker-compose-by-anjal-bam-e0a.
6. Chitlangya, Ronak. “JWT Authentication with React JS and Django.” *Medium*, Medium, 24 Apr. 2023, [medium.com/@ronakchitlangya1997/jwt-authentication-with-react-js-and-django-c034aae1e60d](mailto:medium.com/@ronakchitlangya1997/jwt-authentication-with-react-js-and-django-c034aae1e60d).
7. Yu, Brian, and David Malan. “Lecture 7 - CS50’s Web Programming with Python and JavaScript.” *Lecture 7 - CS50’s Web Programming with Python and JavaScript*, Harvard University, cs50.harvard.edu/web/2020/notes/7/#docker. Accessed 13 Dec. 2024.
8. “React-Bootstrap Introduction.” *React Bootstrap*, react-bootstrap.netlify.app/docs/getting-started/introduction. Accessed 13 Dec. 2024.

# Appendix

Appendix A - docker-compose.yml:

version: '3'

services:

db:

image: postgres

environment:

POSTGRES\_DB: postgres

POSTGRES\_USER: postgres

POSTGRES\_PASSWORD: postgres

backend:

build:

context: .

dockerfile: Dockerfile.backend

command: python manage.py runserver 0.0.0.0:8000

volumes:

- ./backend:/app

depends\_on:

- db

ports:

- "8000:8000"

frontend:

build:

context: .

dockerfile: Dockerfile.frontend

command: npm start

volumes:

- ./frontend:/app/frontend

ports:

- "3000:3000"

depends\_on:

- backend

Appendix B – Directory Tree:

.

├── backend

│   ├── api

│   ├── backend

│   ├── manage.py

│   └── requirements.txt

├── frontend

│   ├── README.md

│   ├── node\_modules

│   ├── package-lock.json

│   ├── package.json

│   ├── public

│   └── src

├── venv

│   ├── bin

│   ├── include

│   ├── lib

│   └── pyvenv.cfg

├── docker-compose.yml

├── Dockerfile.backend

├── Dockerfile.frontend

├── README.md