

# THESIS

SEIFELDIN SALEH  
JGDCU1

Debrecen,  
2024

# UNIVERSITY OF DEBRECEN

# FACULTY OF INFORMATICS

## *Football Booking System*

*Supervisor :*

*Dr. Vagner Aniko*

*Associate Professor*

*Candidate :*

*Seifeldin Saleh*

*Computer Science Bsc.*

*Debrecen 2024*

# *Table of Contents*

1. Introduction
2. Technology Overview
  - 2.1. JRE Stack
  - 2.2. MySQL
  - 2.3. React js
  - 2.4. Java
3. Development Lifecycle
  - 3.1. Installment
4. Front-End
  - 4.1. App.js
  - 4.2. Home Page
  - 4.3. NavBar
  - 4.4. Log in Link
  - 4.5. Customer
  - 4.6. Wallet
  - 4.7. Booking
  - 4.8. Admin
  - 4.9. Admin NavBar
  - 4.10. Add Ground
  - 4.11. View All Grounds

4.12. View All Customers

4.13. View All Bookings

## *Table of Figures*

Figure 1 Project Overview

Figure 2 App.js

Figure 3 Home Page

Figure 4 KickCrew HomePage Links

Figure 5 NavBar Home

Figure 6 About Us

Figure 7 Contact Us

Figure 8 Header.jsx

Figure 9 AboutUs.jsx

Figure 10 ContactUs.jsx

Figure 11 Login Link

Figure 12 User Login

Figure 13 UserLoginForm.jsx

Figure 14 Customer Registration

Figure 15 Customer Login

Figure 16 Customer HomePage

Figure 17 CustomerHeader.jsx

Figure 18 UserRegister.jsx

Figure 19 User Apis

Figure 20 Customer All grounds

Figure 21 ViewAllGround.jsx

Figure 22 Wallet

Figure 23 Wallet Balance

Figure 24 MyWallet.jsx

Figure 25 Booking Ground

Figure 26 Ground.jsx

Figure 27 My Bookings

Figure 28 UserRegister.jsx

Figure 29 Admin Register

Figure 30 UserLoginForm.jsx

Figure 31 Admin Login

Figure 32 Admin NavBar

Figure 33 AddGroundForm.jsx

Figure 34 Add Ground

Figure 35 View All Ground

Figure 36 ViewAllGround.jsx

Figure 37 View All Customers

Figure 38 ViewAllCustomer.jsx

Figure 39 All bookings

Figure 40 ViewAllBooking.jsx

Figure 41 AddWalletMoneyRequestDTO.java

Figure 42 Ground.java

## *1. Introduction*

In the digital realm, web applications, commonly known as web apps, primarily function within web browsers rather than a device's operating system. This approach enables a broader audience to access and interact with the web application via the World Wide Web.

The development of web applications entails creating software programs that can be accessed over the internet via remote servers.

Web applications are designed to be accessed exclusively through internet browsers such as Mozilla Firefox, Internet Explorer, and Google Chrome. They leverage essential technologies like JavaScript, CSS (Cascading Style Sheets), and HTML5 (Hyper Text Markup Language) for development. Unlike traditional software, web apps do not need to be downloaded and installed locally on devices; instead, they operate entirely online, relying on network connections to function seamlessly. This architecture allows users to access and utilize web apps from various devices without the need for extensive storage or installation procedures.

The front end of web applications is built using client-side programming, primarily relying on HTML, CSS, and JavaScript. HTML governs the structure and content display of web pages, while CSS ensures proper styling and layout. JavaScript enhances user interaction by enabling dynamic and responsive elements within the web app interface.

Complementing client-side functionality, server-side programming involves scripting languages such as Ruby, Java, and Python. These languages are instrumental in creating server-side scripts that manage data processing, generate dynamic content, and secure the underlying source code of web applications. Server-side scripting also facilitates the creation of unique user interfaces tailored to specific application requirements.

Furthermore, online applications rely on databases like MySQL or MongoDB for efficient data storage and management. These databases play a crucial role in storing and retrieving information essential for the functioning and persistence of web applications.

## *2. Technology Overview*

### *2.1. JRE Stack*

The project KickCrew uses JRE Stack. Java for backend development using Eclipse IDE, React.js for frontend development with Visual Studio Code, and MySQL for efficient database management. This stack ensures a comprehensive and streamlined approach to full-stack application development.

## *2.2. MySQL*

For efficient querying and manipulation of the KickCrew application, I have used MySql Workbench.

## *2.3. React JS*

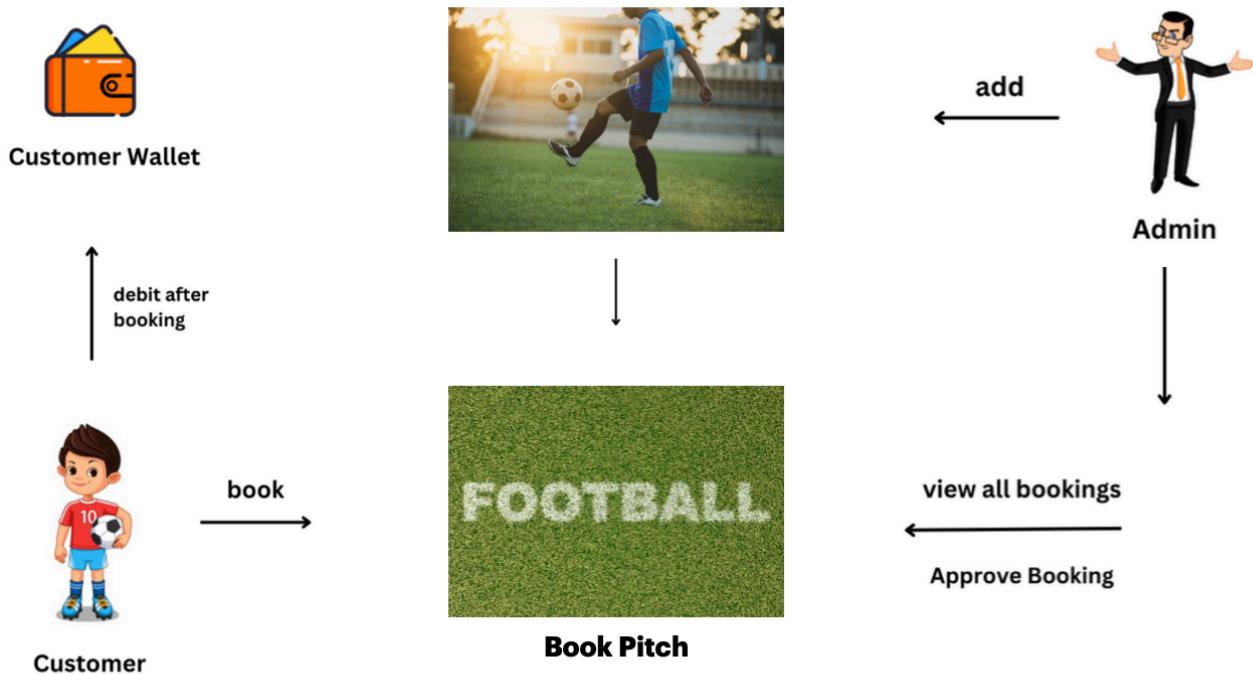
React.js was chosen for frontend development due to its powerful and flexible nature in user interfaces. It uses a component-based approach which allows it to break down the user interface into smaller components which allows for easier managing and therefore, updates can be made a lot more efficiently. The virtual DOM updates the page a lot faster this way because it refreshes a certain component rather than the whole page. As well as, the simplicity within the React.js syntax that allows for quicker writing and understanding of the code.

## *2.4. Java*

Choosing Java with Eclipse IDE for the backend development brought several benefits to the project. One of Java's strengths is its platform independence. Which means, that applications that are developed in Java can work on several operating systems without needing major changes. This flexibility simplifies the development process. As well as, Java can handle large amounts of requests at once and in an efficient manner. The reason for Eclipse IDE was due to the user-friendly environment it provides.

By choosing Java with Eclipse IDE for the backend, a reliable and versatile combination was chosen. This supports the complex needs for the task management while still enabling us to build and maintain the application effectively.

Figure 1 Project Overview



KickCrew the Booking System project using React JS and Eclipse is a 2-module project where the administrator can add multiple grounds in the system. After adding the football grounds, the customers are then able to view all the available grounds on the website. However, to book the football ground they must be logged in through the website.

### 3. Development Lifecycle

#### 3.1. Installment

After setting up my Java backend with everything it needed, I started working on the frontend of my application. Using Visual Studio Code, I created a React application template by running "npx create-react-app" to set up "my react app." Once that was done, I went to the project directory using the terminal and used "npm start" to get the project up and running. From there, I focused on refining the frontend user interface to fit the needs of my project.

## 4. Front-End

### 4.1. App.js

In my project's App.js file, I've set up routes for each page of the application. Think of these routes like pathways that direct users to different parts of the app. Each route corresponds to a specific page or feature, like a home page, a profile page, or maybe a page for tasks.

To make these routes work, I've imported components that represent each page. These components are like building blocks that define what users see and interact with. By organizing these routes and components in App.js, it helps keep everything structured and makes it easier for me to manage how the app flows and behaves. This was installed by going in our project terminal and writing `npm install react-router-dom`.

```
import "./App.css";

import { Route, Routes } from "react-router-dom";

import AboutUs from "./page/AboutUs";
import ContactUs from "./page/ContactUs";
import Header from "./NavbarComponent/Header";
import HomePage from "./page/HomePage";
import UserRegister from "./UserComponent/UserRegister";
import UserLoginForm from "./UserComponent/UserLoginForm";
import AddGroundForm from "./GroundComponent/AddGroundForm";
import Ground from "./GroundComponent/Ground";
import ViewAllCustomer from "./UserComponent/ViewAllCustomer";
import ViewMyBooking from "./BookingComponent/ViewMyBooking";
import ViewAllBooking from "./BookingComponent/ViewAllBooking";
import VerifyBooking from "./BookingComponent/VerifyBooking";
import MyWallet from "./UserComponent/MyWallet";
import AddReview from "./ReviewComponent/AddReview";
import ViewAllGround from "./GroundComponent/ViewAllGround";
```

```
import ViewAllTurf from "./GroundComponent/ViewAllTurf";
```

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under "KICK-CREW-FRONTEND".
- JS App.js:** The active code editor pane displays the `App.js` file content.
- Terminal:** The bottom pane shows the output of a webpack compilation command: "webpack compiled successfully".

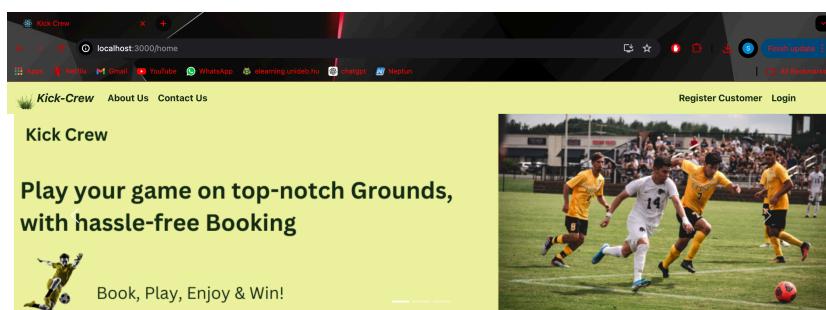
```
src > JS App.js > (e) default
  20  function App() {
  21    return (
  22      <div>
  23        <Header />
  24        <Routes>
  25          <Route path="/" element={<HomePage />} />
  26          <Route path="/home" element={<HomePage />} />
  27          <Route path="/home/all/hotel/location" element={<HomePage />} />
  28
  29          <Route path="contact" element={<ContactUs />} />
  30          <Route path="about" element={<AboutUs />} />
  31
  32          <Route path="user/hotel/register" element={<UserRegister />} />
  33          <Route path="user/customer/register" element={<UserRegister />} />
  34          <Route path="user/admin/register" element={<UserRegister />} />
  35          <Route path="/user/login" element={<UserLoginForm />} />
  36
  37          <Route path="admin/ground/add" element={<AddGroundForm />} />
  38          <Route path="book/ground/add" element={<AddGroundForm />} />
  39          <Route path="user/customer/all" element={<ViewAllCustomer />} />
  40
  41          <Route path="/book/ground/:groundId" element={<Ground />} />
  42          <Route path="user/ground/bookings" element={<ViewMyBooking />} />
  43          <Route path="user/ground/booking/all" element={<ViewAllBooking />} />
  44
  45          <Route path="user/admin/verify/booking/:bookingId"
  46            element={<VerifyBooking />} />
  47
  48          <Route path="/customer/wallet" element={<MyWallet />} />
  49          <Route path="/ground/review/add" element={<AddReview />} />
  50          <Route path="/admin/ground/all" element={<ViewAllGround />} />
  51          <Route path="/turf/all" element={<ViewAllTurf />} />
  52
  53        </Routes>
  54      </div>
  55    );
  56  }
  57
  58
  59  export default App;
```

Bottom status bar: In 59, Col 20, Spaces: 2, UTE-R, LF, JavaScript, Go Live

Figure 2 App.js

## 4.2. Home Page

Below is a snippet of the home page which shows a react component that is simply the Front page of the football pitch booking website. In the Component, we have what can be similar to a slide show. In the first slide theres a text saying ("Play your game on top-notch Grounds, with hassle-free

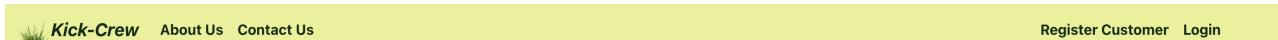


Booking"). In the future this can be used to shuffle between advertisements or football news.

Below that component is a Welcome to KickCrew, and just below it is a short description of what the application aims to do. And the benefits of booking through KickCrew rather than the old fashioned phone call.

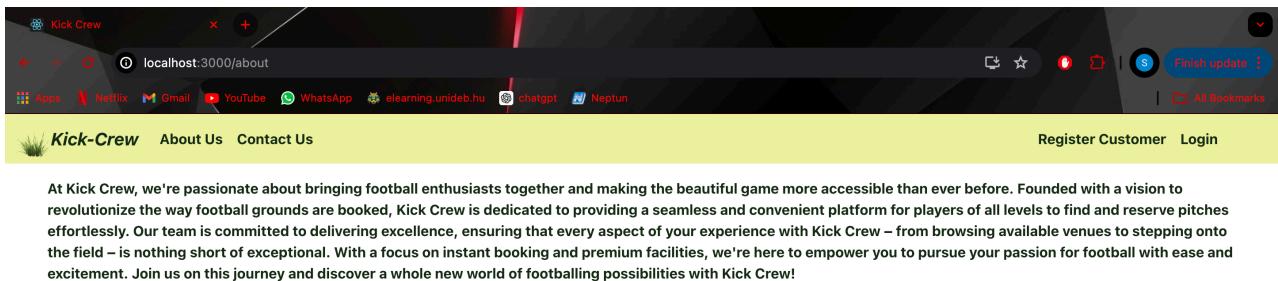


Figure 3 Home Page



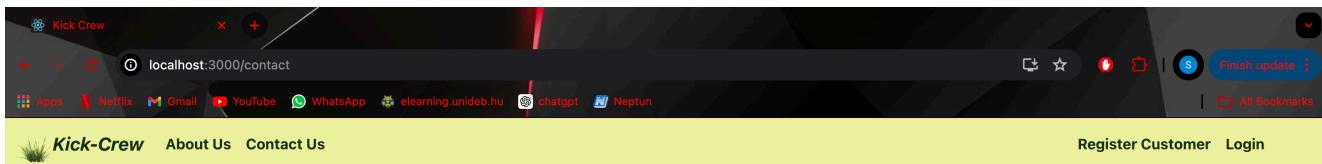
Figure

4 KickCrew HomePage links



Below  
the  
text,  
there  
is the  
Get

Figure 6 About Us



We're here to assist you every step of the way! Whether you have a question, feedback, or need support, our team at Kick Crew is ready to help. You can reach us through the following channels:

Email: [info@kickcrew.com](mailto:info@kickcrew.com)  
Phone: +1 (123) 456-7890  
Address: 123 Main Street, Cityville, State, Zip Code

Connect with us on social media:  
Facebook: [@kickcrewofficial](#)  
Twitter: [@kickcrew\\_tweets](#)  
Instagram: [@kickcrewpics](#)

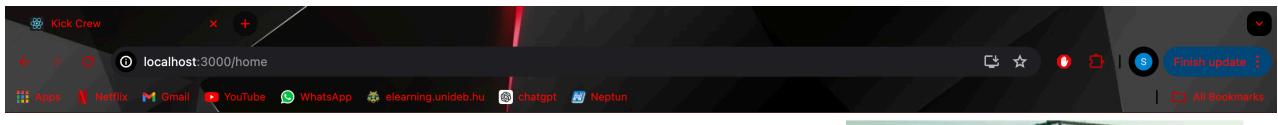
Don't hesitate to get in touch – we're always eager to hear from fellow football enthusiasts like you!

localhost:3000/contact

Figure 7 Contact Us.

Started button. Which takes you to the page that shows all the football grounds.

#### 4.3. Navbar



## Welcome to Kick Crew

Welcome to Kick Crew, where your passion for football meets the convenience of online booking! Whether you're a seasoned player or just looking to kick around with friends, Kick Crew is your go-to destination for securing the perfect pitch. With our user-friendly platform, finding and reserving football grounds has never been easier. Simply browse through our diverse selection of venues, check availability in real-time, and lock in your spot with just a few clicks. Get ready to lace up your boots and elevate your game with Kick Crew – because every match starts with the perfect pitch!

At Kick Crew, we pride ourselves on offering instant booking and premium facilities to enhance your football experience. Say goodbye to the hassle of lengthy reservation processes – with our platform, you can secure your preferred ground instantly, ensuring you never miss a match. Plus, our carefully curated venues boast top-notch facilities, from pristine pitches to state-of-the-art amenities, guaranteeing an unparalleled playing experience every time. Join Kick Crew today and elevate your game to new heights!



## Instant Booking & Premium Facilities

Gone are the days of tedious turf reservation processes! With our Instant Turf Booking feature, securing your favorite playing field is faster and easier than ever before. Simply log in to our platform, select your preferred location, date, and time, and with just a few taps, your reservation is confirmed. No more waiting on hold or dealing with last-minute disappointments.

At our Turf Booking System, we believe in providing nothing but the best for our players. That's why we've partnered with top-notch turf facilities that offer a range of premium amenities to elevate your sports experience. From impeccably maintained, lush green playing surfaces to well-equipped changing rooms and modern lighting systems for evening games, every aspect of our partnered turfs is carefully curated to cater to your needs.

[Get Started](#)

The Navigation bar below is the home navigation bar before any admin or customers log in. It has the website's name which is a link at any moment to go back to the home page. An about us page. And a Contact Us page with has the email, phone number, and the address of KickCrew. As well as, the accounts on social media on applications such as: FaceBook, Twitter and Instagram.

Figure 5 NavBar Home

```

const ContactUs = () => {
  return (
    <div className="text-color ms-5 me-5 mt-3">
      <b>We're here to assist you every step of the way! Whether you have a question, feedback, or need support, our team at Kick Crew is ready to help.</b>
      <br/>
      <br/>
      You can reach us through the following channels:<br/><br/>
      Email: info@kickcrew.com <br/> Phone: +1 (23) 456-7890 <br/>
      Address: 123 Main Street, Cityville, State, Zip Code<br/> <br/>
      Connect with us on social media:<br/><br/>
      Facebook: @kickcrewofficial(<">)
      <br/>
      Twitter: @kickcrew_tweets
      <br/>
      Instagram: @kickcrewpics<br/>
      <br/>
      Don't hesitate to get in touch <br/> we're always eager to hear from fellow football enthusiasts like you!
    </div>
  );
}

export default ContactUs;

```

Figure 8  
Header.jsx.

The Header in the NavBar Component is the general NavBar. It is used to its purpose generally. However, this changes depending on the user logged in. Depends on if it is a user or an admin.

```

const AboutUs = () => {
  return (
    <div className="text-color ms-5 me-5 mt-3">
      <b>At Kick Crew, we're passionate about bringing football enthusiasts together and making the beautiful game more accessible than ever before. Founded with a vision to revolutionize the way football grounds are booked, Kick Crew is dedicated to providing a seamless and convenient platform for players of all levels to find and reserve pitches effortlessly. Our team is committed to delivering excellence, ensuring that every aspect of your experience with Kick Crew <br/> is nothing short of exceptional. With a focus on instant booking and premium facilities, we're here to empower you to pursue your passion for football with ease and excitement! Join us on this journey and discover a whole new world of footballing possibilities with Kick Crew!</b>
    </div>
  );
}

export default AboutUs;

```

Figure 9  
AboutUs.jsx

The About Us page can be accessed using any NavBar and can be accessed at any moment excluding registration and login pages.



Figure 10 ContactUs.jsx

#### Figure 4.4. Log in Link

Shown below is the bottom of the homepage which includes various links. The Get Started link works as a way to view all the grounds. However, not much can be done since there is no user logged in. At the bottom the Log in link directs you directly to the user login.

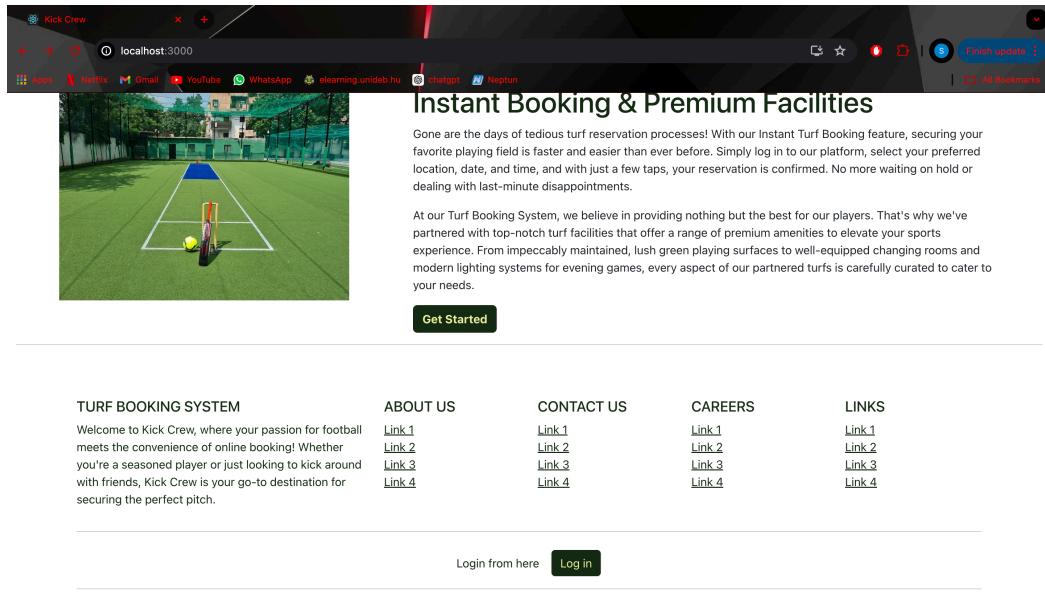


Figure 11  
Login link.

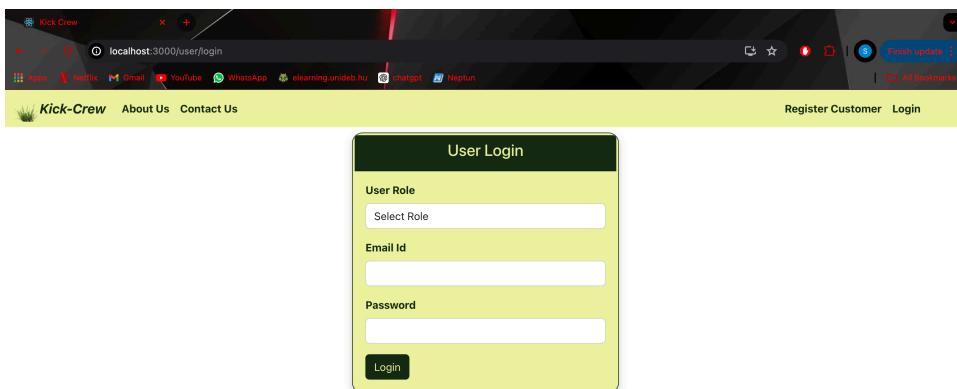


Figure 12 User Login.

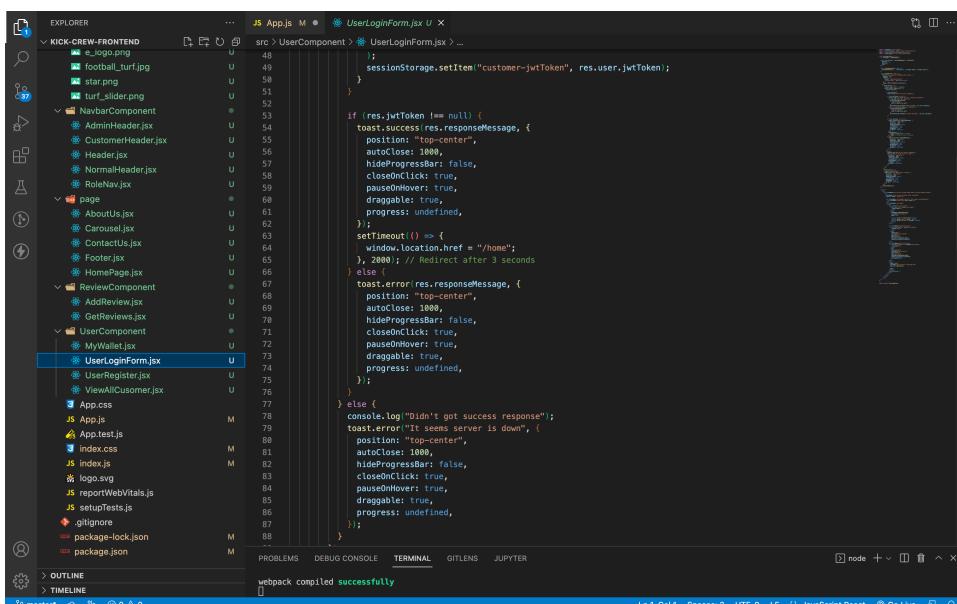


Figure 13  
UserLoginForm.jsx

The 'UserLoginForm' component is a React-based form designed for user authentication in a web application. It includes fields for selecting user role (admin or customer), entering email and password credentials. When users submit their login details, the form sends a POST request to '<http://localhost:8080/api/user/login>' using the 'fetch' API. If authentication succeeds (\*res.success\*), users are redirected to '/home' after storing role-specific session data ('active-admin' or 'active-customer') and JWT tokens ('admin-jwtToken' or 'customer-jwtToken') in 'sessionStorage'. The component also displays toast notifications using 'react-toastify' for success or error messages based on server responses.

Styled with Bootstrap classes for a responsive layout, the 'UserLoginForm' integrates seamlessly into the application's login page or authentication flow, offering a straightforward user experience. Developers should ensure the backend API endpoint is correctly implemented to handle authentication and provide responses in the expected format ('success', 'jwtToken', 'responseMessage').

## 4.5. Customer

The screenshot shows the 'Register customer' form. It includes fields for First Name, Last Name, Email Id, Password, User Gender (with a dropdown for 'Select Sex'), Contact No, Age, Street, City, and Pincode. A 'Register User' button is at the bottom.

Figure 14 Customer Registration

The customer register page asks for. First Name, Last Name, Email ID, Password, User gender, Contact No Age, Street, City, Code.

The screenshot shows the 'User Login' form. It includes fields for User Role (Customer), Email Id (selfthecustomer@gmail.com), and Password. A 'Login' button is at the bottom. A success message 'Logged in Successful!!!' is shown in a pop-up window.

Figure 15 Customer Login

After the customer registration is complete, the user is straight away directed to the login. In there once the credentials are verified the user is informed with a pop up that says "Logged in Successful"

Welcome to Kick Crew  
Welcome to Kick Crew, where your passion for football meets the convenience of online booking! Whether you're a seasoned player or just looking to kick around with friends, Kick Crew is your go-to destination for securing the perfect pitch. With our user-friendly platform, finding and reserving football grounds has never been easier. Simply browse through our diverse selection of venues, check availability in real-time, and lock in your spot with just a few clicks. Get ready to face up your boots and elevate your game with Kick Crew – because every match starts with the perfect pitch.

Figure 16 Customer HomePage

As mentioned previously the NavBar changes when the customer is logged in.



```
src= NavBarComponent > CustomerHeader.jsx > ...
17  );
18  storageService.retrieveItem("customerToken");
19  storageService.retrieveItem("customerJWTToken");
20
21  navigate("main");
22  window.location.reload(true);
23}
24
25 return (
26   <ul class="nav-item nav-item--auto mb-2 sb-lq-8 mb-4">
27     <Link
28       to="#color-selector"
29       className="nav-link active"
30       aria-current="page">
31       <b>My Wallet</b>
32     </Link>
33   <li class="nav-item">
34     <Link
35       to="#bookings"
36       className="nav-link">
37       Bookings
38     </Link>
39   <li class="nav-item">
40     <Link
41       to="#turf"
42       className="nav-link active"
43       aria-current="page">
44       Booked Turfs
45     </Link>
46   <li class="nav-item">
47     <Link
48       to="#logout"
49       onClick={userLogout}>
50         Logout
51       <b>Logout</b>
52     </Link>
53   </li>
54 </ul>
55 </div>
56 <HomePage.jsx > ...
57 <ReviewComponent > ...
58 <AddReview.jsx > ...
59 <GetReviews.jsx > ...
60
```

Figure 17 CustomerHeader.jsx

Figure 18 UserRegister.jsx

user-controller : User Controller		Show/Hide	List Operations	Expand Operations
POST	/api/user/add/wallet/money			Api to add wallet money
GET	/api/user/customer/all			getAllCustomers
GET	/api/user/customer/wallet/fetch			getCustomerWallet
GET	/api/user/gender			Api to get all user gender
GET	/api/user/id			Api to fetch the User using user Id
POST	/api/user/login			Api to login any User
POST	/api/user/register			Api to register any User
GET	/api/user/roles			Api to get all user roles

Figure 19 User APIs

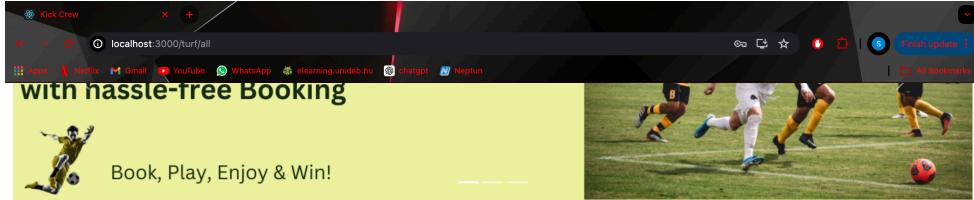


Figure 20 Customer All grounds

<b>Doberdo Ground</b> A beautiful and scenic 7v7 football pitch.  Ground Width (in Feet): 100 Ground Height (in Feet): 100 Ground Length (in Feet): 200  Price : HUF 9500 <a href="#">Book Now</a>	<b>Kassai football</b> A great football pitch for 11v11 games  Ground Width (in Feet): 100 Ground Height (in Feet): 100 Ground Length (in Feet): 100  Price : HUF 11500 <a href="#">Book Now</a>
--	--

Shows the customer all the available pitches. This is provided by the admin. And a vibrant Book Now button is presented with the price above.

```

src > GroundComponent > ViewAllGround.jsx U X
1 import { useState, useEffect } from "react";
2 import axios from "axios";
3 import React from "react";
4 import { ToastContainer, toast } from "react-toastify";
5
6 const ViewAllGround = () => {
7   const [allGround, setAllGround] = useState([]);
8
9   useEffect(() => {
10     const getAllGround = async () => {
11       const allGround = await retrieveAllGround();
12       if (allGround) {
13         setAllGround(allGround.grounds);
14       }
15     };
16     getAllGround();
17   }, []);
18
19   const retrieveAllGround = async () => {
20     const response = await axios.get("http://localhost:8080/api/ground/fetch");
21     console.log(response.data);
22     return response.data;
23   };
24
25   const deleteGround = (groundId) => {
26     fetch(`http://localhost:8080/api/ground/delete?groundId=${groundId}`, {
27       method: "DELETE",
28       headers: {
29         Accept: "application/json",
30         "Content-Type": "application/json",
31       },
32     });
33   };
34
35   .then(result) => {
36     result.json().then(res) => {
37       if (res.success) {
38         console.log("Got the success response");
39
40         toast.success(res.responseMessage, {
41           position: "top-center",
42           autoClose: 1000,
43         });
44       }
45     };
46   };
47 }
48
49 
```

Figure 21 ViewAllGround.jsx

## 4.6. Wallet

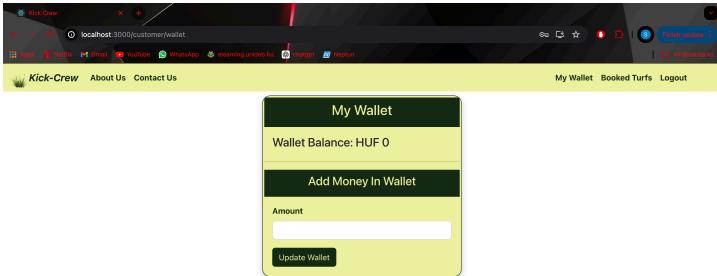


Figure 22 Wallet

Before the user can book. He needs to add money to the wallet by writing an amount in the 'amount' are and clicking update wallet.

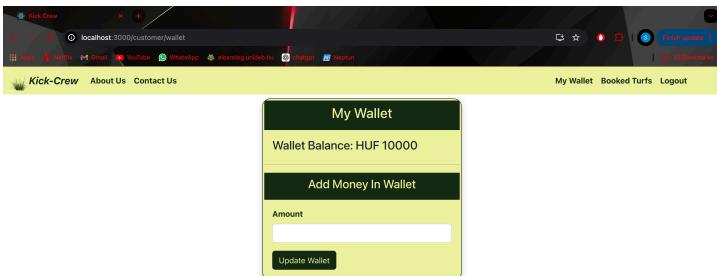


Figure 23 Wallet Balance

```

src > UserComponent > MyWallet.jsx > MyWallet > useEffect() callback
U 37   const addMoneyInWallet = () => {
U 38     const url = "http://localhost:8888/api/user/add/wallet/money";
U 39     const options = {
U 40       method: "POST",
U 41       headers: {
U 42         "Content-Type": "application/json",
U 43       },
U 44       body: JSON.stringify(walletRequest),
U 45     };
U 46     fetch(url, options)
U 47     .then(response => {
U 48       console.log("Response:", response);
U 49       if (response.ok) {
U 50         return response.json();
U 51       } else {
U 52         throw new Error(`Error: ${response.status}`);
U 53       }
U 54     })
U 55     .then(data => {
U 56       setSuccess(true);
U 57       toast.success(`You've added ${data.message}`);
U 58     })
U 59     .catch(error => {
U 60       toast.error(`An error occurred: ${error.message}`);
U 61     });
U 62   }
U 63   useEffect(addMoneyInWallet, []);
U 64   const [balance, setBalance] = useState(0);
U 65   const [success, setSuccess] = useState(false);
U 66   const [error, setError] = useState(null);
U 67   const [loading, setLoading] = useState(true);
U 68   const [progress, setProgress] = useState(0);
U 69   const [toastRef, {show} ] = useToast();
U 70   const [progressBarStyle] = useState({
U 71     width: progress + "%"
U 72   });
U 73   const [progressBarColor] = useState({
U 74     background: "#007bff",
U 75     color: "#fff"
U 76   });
U 77   const [progressBarWidth] = useState({
U 78     width: progress + "%"
U 79   });

setInterval(() => {
  window.location.reload();
}, 1000 / 2); // Reloads every 3 seconds
} else {
  console.log("Data's not yet received!");
  toast.error("It seems server is down!");
  position: "top-center",
  autoClose: 1000,
  hideProgressBar: false,
  closeOnClick: true,
  pauseOnHover: true,
  draggable: true,
  progress: undefined,
};

setTimeout(() => {
  window.location.reload();
}, 1000 / 2); // Reloads after 3 seconds
}

```

Figure 24 MyWallet.jsx

The MyWallet component manages the user's wallet, allowing them to view their current balance and add money. Upon mounting, it fetches the wallet balance from the backend using an API call and displays it. Users can input an amount to add to their wallet, which is then sent to the server when they submit the form. The response is handled with appropriate success or error messages using react-toastify, and the page refreshes to show the updated balance.

This component leverages React hooks for state management and lifecycle methods, and axios for making HTTP requests. It also handles user interactions and provides feedback through toast notifications, ensuring a smooth and responsive user experience.

## 4.7. Booking

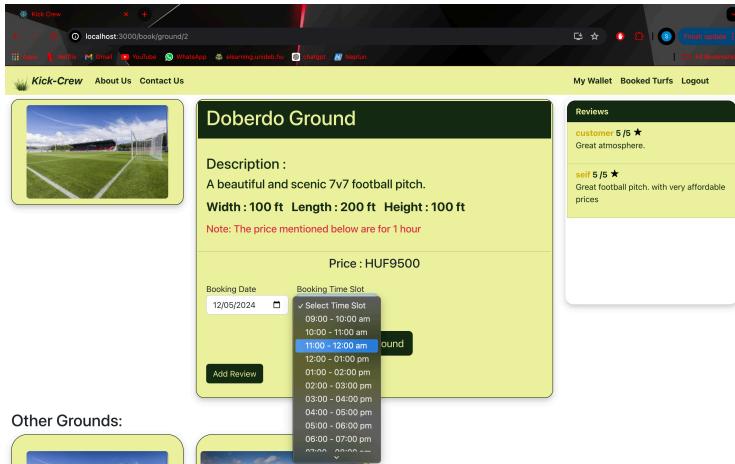


Figure 25 Booking Ground

```

App.js
import React from 'react';
import './App.css';
import Header from './components/Header';
import Footer from './components/Footer';
import GroundList from './components/GroundList';
import { useState } from 'react';

function App() {
  const [grounds, setGrounds] = useState([]);
  const [selectedGround, setSelectedGround] = useState(null);
  const [selectedSlot, setSelectedSlot] = useState(null);

  const handleBookingInput = (e) => {
    setSelectedSlot(e.target.value);
  };

  const retrieveAllSlots = async () => {
    const response = await axios.get('http://localhost:8080/api/book/ground/fetch/slots');
    return response.data;
  };

  const getALGrounds = async () => {
    const response = await axios.get('http://localhost:8080/api/ground/fetch');
    return response.data;
  };

  const retrieveAllGrounds = async () => {
    const response = await axios.get('http://localhost:8080/api/ground/fetch');
    return response.data;
  };

  const retrieveGround = async () => {
    const response = await axios.get(`http://localhost:8080/api/ground/id/${selectedGround}`);
    return response.data;
  };

  useEffect(() => {
    const retrieveAllGrounds = async () => {
      const response = await axios.get('http://localhost:8080/api/ground/fetch');
      if (response) {
        setGrounds(response);
      }
    };
    const getALGrounds = async () => {
      const response = await axios.get('http://localhost:8080/api/ground/fetch');
      if (response) {
        setGrounds(response);
      }
    };
    const retrieveAllSlots = async () => {
      const response = await axios.get('http://localhost:8080/api/book/ground/fetch/slots');
      if (response) {
        setSelectedSlot(response);
      }
    };
    const retrieveGround = async () => {
      const response = await axios.get(`http://localhost:8080/api/ground/id/${selectedGround}`);
      if (response) {
        setSelectedGround(response);
      }
    };
  }, []);

  return (
    <div>
      <Header />
      <GroundList grounds={grounds} selectedGround={selectedGround} />
      <div>
        {selectedGround ? (
          <div>
            <h3>{selectedGround.name}</h3>
            <p>Description : {selectedGround.description}</p>
            <p>Width : {selectedGround.width} ft Length : {selectedGround.length} ft Height : {selectedGround.height} ft</p>
            <p>Note: The price mentioned below are for 1 hour</p>
            <p>Price : HUF{selectedGround.price}</p>
            <div>
              <div>Booking Date:</div>
              <input type="date" value="2024-05-12" />
              <div>Booking Time Slot:</div>
              <div>Select Time Slot</div>
              <div>09:00 - 10:00 am</div>
              <div>10:00 - 11:00 am</div>
              <div>11:00 - 12:00 pm</div> (Selected)
              <div>12:00 - 01:00 pm</div>
              <div>01:00 - 02:00 pm</div>
              <div>02:00 - 03:00 pm</div>
              <div>03:00 - 04:00 pm</div>
              <div>04:00 - 05:00 pm</div>
              <div>05:00 - 06:00 pm</div>
              <div>06:00 - 07:00 pm</div>
            </div>
            <button>Add Review
          </div>
        ) : null}
      </div>
      <Footer />
    </div>
  );
}

export default App;
  
```

Figure 26 Ground.jsx

The Ground component showcases detailed information about a specific sports ground, allowing users to book it for a certain date and time and leave reviews. When the component loads, it fetches the ground details, available time slots, and other grounds from the server using axios. Users can select a date and time slot to book the ground, and the booking info is sent to the backend. The component handles responses with success or error messages using react-toastify. In addition to booking, the component displays the ground's image, name, description, and dimensions. Logged-in users see an option to add a review for the ground. There's also a section listing other available grounds for users to explore, making it easy to find and book sports grounds while also sharing their experiences.

Ground	Ground Name	Booking Id	Customer Name	Customer Contact	Booking Date	Booking Time Slot	Total Payable Amount	Booking Status	Action
	Dobredo Ground	DSDILMSQYQ	self customer	0036205930492	2024-02-19	01:00 - 02:00 pm	9500.0	Approved	<button>Cancel Booking</button>
	Kassai football	SCC72JDGK4	self customer	0036205930492	2024-02-12	01:00 - 02:00 pm	11500.0	Pending	<button>Cancel Booking</button>

## 4.8. Admin

Figure 27 My Bookings

After the customer submits a booking, they are automatically redirected to the "My Bookings" page. Here, they can view the ground name, a unique booking ID, customer details, and the booking information, including the price. The booking status is initially marked as "pending" until an admin logs in and

approves the request.

The UserRegister component simplifies the sign-up process for admins. To register, admins should visit: <http://localhost:3000/user/admin/register>. When accessed, the form automatically sets the role to 'admin' based on the URL. It collects essential details like name, email, password, contact info, and address. Additionally, gender options are dynamically fetched from the server to ensure up-to-date choices. After filling out the form, admins can submit their information, which is then sent to the server. If the registration is successful, a toast notification confirms it, and they are redirected to the login page.

```

// UserRegister.jsx
import React, { useState } from 'react';
import { toast } from 'react-toastify';
import { saveUser } from '../services/UserService';

const UserRegister = () => {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [address, setAddress] = useState('');
  const [gender, setGender] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    fetch(`http://localhost:8080/api/user/register`, {
      method: 'POST',
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ user }),
    }).then((result) => {
      console.log("result", result);
      result.json().then((res) => {
        console.log(res);

        if (!res.success) {
          console.log("Got the success response");
        }

        toast.success(res.responseMessage, {
          position: "top-center",
          autoClose: 1000,
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });

        setTimeout(() => {
          navigate("/user/login");
        }, 1000); // Redirect after 3 seconds
      }) else {
        console.log("Didn't get success response");
        toast.error("It seems server is down", {
          position: "top-center",
          autoClose: 1000,
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });
        setTimeout(() => {
          // ...
        }, 1000);
      }
    });
  };
}

export default UserRegister;

```

Figure 28  
UserRegister.jsx

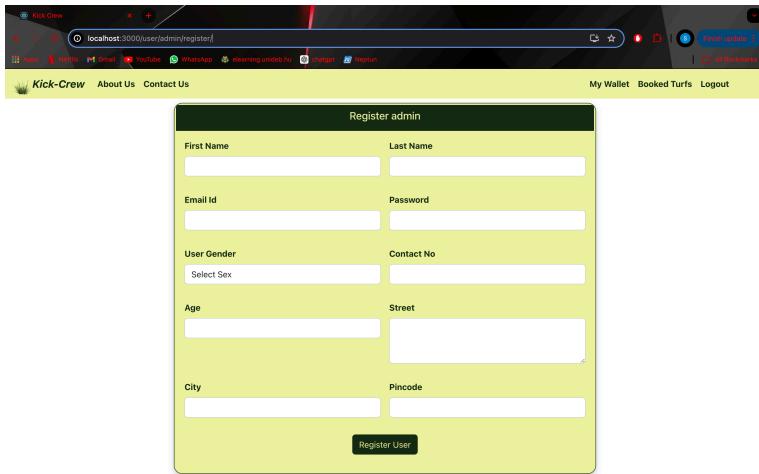


Figure 29 Admin Register

The UserLoginForm component handles user authentication and includes functionality specifically for admin users. Upon a successful login,

the system checks the user's role. If the user is an admin, their details are stored in the session storage under "active-admin" along with a JWT token labeled "admin-jwtToken". This allows the admin to have an authenticated session with appropriate access levels.

In the event of a successful login, the admin is greeted with a success toast notification and redirected to the home page. In order to register a new admin. The user/admin must go to the link : <http://localhost:3000/user/admin/register>

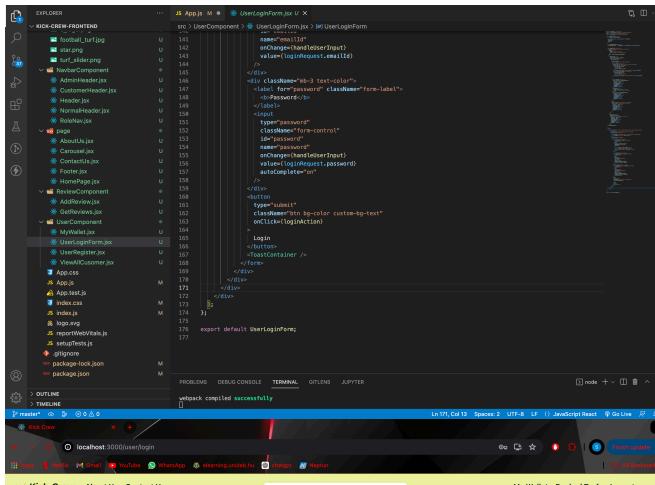


Figure 30 UserLoginForm.jsx



Figure 31 Admin Login

## 4.9. Admin NavBar

The AdminHeader component is designed to provide easy navigation for admins within the application. Upon logging in, admins can see options to add a new ground, view all grounds, view all customers, and view all bookings. These options are displayed as navigation links, making it simple for admins to manage the platform efficiently. The component also fetches the currently active admin from the session storage to personalize the experience.

When an admin decides to log out, they can simply click the "Logout" link. This triggers the adminLogout function, which displays a success toast notification, clears the admin session data, reloads the page, and navigates back to the home page. This ensures a smooth and user-friendly logout process. The ToastContainer is included to handle the display of toast notifications, providing instant feedback to the admin for actions taken.

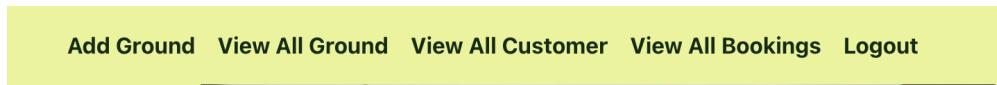


Figure 32 Admin Navbar

A screenshot of the VS Code IDE. The left sidebar shows the project structure under "EXPLORER". The "src" folder contains several components like BookingComponent, GroundComponent, and NavComponent. The "AddGroundForm.jsx" file is selected and shown in the main code editor area. The code is a functional component for adding a new ground, using useState for state management and axios for API calls. The right side of the interface shows the "PROBLEMS", "DEBUG CONSOLE", "TERMINAL", "GITLENS", and "JUPYTER" tabs, with the "TERMINAL" tab active. At the bottom, a message "webpack compiled successfully" is visible.

```
1 import { useState, useEffect } from "react";
2 import axios from "axios";
3 import { useNavigate } from "react-router-dom";
4 import { ToastContainer, toast } from "react-toastify";
5
6 const AddGroundForm = () => {
7   let navigate = useNavigate();
8
9   const [selectedImage, setSelectedImage] = useState(null);
10
11   const [ground, setGround] = useState({
12     name: "",
13     description: "",
14     width: "",
15     height: "",
16     price: "",
17     length: ""
18   });
19
20   const handleInput = (e) => {
21     setGround((...ground, {e.target.name}): e.target.value));
22   };
23
24   const saveGround = () => {
25     const formData = new FormData();
26     formData.append("image", selectedImage);
27     formData.append("name", ground.name);
28     formData.append("description", ground.description);
29     formData.append("width", ground.width);
30     formData.append("height", ground.height);
31     formData.append("price", ground.price);
32     formData.append("length", ground.length);
33
34     axios
35       .post("http://localhost:8888/api/ground/add", formData)
36       .then(result => {
37         console.log("result", result);
38         result.json().then(res => {
39           console.log(res);
40           if (res.success) {
41             toast("Ground added successfully!");
42             navigate("/");
43           }
44         });
45       })
46     );
47   };
48
49   return (
50     <div>
51       <form>
52         <input type="file" onChange={handleInput} />
53         <input type="text" name="name" value={ground.name} />
54         <input type="text" name="description" value={ground.description} />
55         <input type="text" name="width" value={ground.width} />
56         <input type="text" name="height" value={ground.height} />
57         <input type="text" name="price" value={ground.price} />
58         <input type="text" name="length" value={ground.length} />
59         <button type="button" onClick={saveGround}>Add Ground</button>
60       </form>
61     </div>
62   );
63 }
64
65 export default AddGroundForm;
```

Figure 33  
AddGroundForm.jsx

## 4.10. Add Ground

The screenshot shows a web browser window with the URL `localhost:3000/admin/ground/add`. The page title is "Add Ground". The form has the following fields:

- Ground Name (input field)
- Ground Description (input field)
- Ground width (input field)
- Ground Length (input field)
- Ground Height (input field)
- Price (input field)
- Select Ground Image (file input field showing "No file chosen")
- Add Ground (button)

Figure 34 Add Ground

The `AddGroundForm` component helps administrators add new grounds easily. It includes fields for the ground's name, description, width, height, length, and price. Admins can also upload an image of the ground. The form data is handled using state, and when submitted, it's sent to the server with axios. If the submission is successful, a toast notification pops up to confirm, and the admin is redirected to the home page.

Admins will find this form straightforward to use. It captures all the essential details needed for a new ground and provides immediate feedback through toast notifications. This makes it simple to add new grounds and manage the information effectively.

## 4.11 View All Grounds

Ground	Name	Description	Ground Width	Ground Height	Ground Length	Price	Action
	Doberdo Ground	A beautiful and scenic 7v7 football pitch.	100	100	200	9500	<button>Remove</button>
	Kassai football	A great football pitch for 11v11 games	100	100	100	11500	<button>Remove</button>

Figure 35 View All Ground

```

// App.js
import { useState, useEffect } from "react";
import axios from "axios";
import React from "react";
import { ToastContainer, toast } from "react-toastify";
import GroundComponent from "./GroundComponent";
import VerifyBooking.jsx
import ViewAllBooking.jsx
import AddGroundForm.jsx
import Ground.jsx
import GroundCard.jsx
import GroundCarousel.jsx
import ViewAllGround.jsx
import ViewAllTurf.jsx
import images
import cricket_turf.jpg
import e_logo.png
import football_turf.jpg
import star.png
import turf_slider.png
import NavbarComponent
import AdminHeader.jsx
import CustomerHeader.jsx
import Header.jsx
import NormalHeader.jsx
import RoleNav.jsx
import page
import AboutUs.jsx
import Carousel.jsx
import ContactUs.jsx
import Footer.jsx
import HomePage.jsx
import ReviewComponent
import AddReview.jsx
import GetReviews.jsx

```

```

const ViewAllGround = () => {
  const [allGround, setAllGround] = useState([]);
  useEffect(() => {
    const getAllGround = async () => {
      const allGround = await retrieveAllGround();
      if (allGround) {
        setAllGround(allGround.grounds);
      }
    };
    getAllGround();
  }, []);
  const retrieveAllGround = async () => {
    const response = await axios.get("http://localhost:8080/api/ground/fetch");
    console.log(response.data);
  };
  const deleteGround = (groundId) => {
    fetch(`http://localhost:8080/api/ground/delete?groundId=${groundId}`, {
      method: "DELETE",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json",
      },
    })
      .then(result) => {
        result.json().then(res => {
          if (res.success) {
            console.log("Got the success response");
            toast.success(res.responseMessage, {
              position: "top-center",
              autoClose: 1000,
            });
          }
        });
      }
  };
}

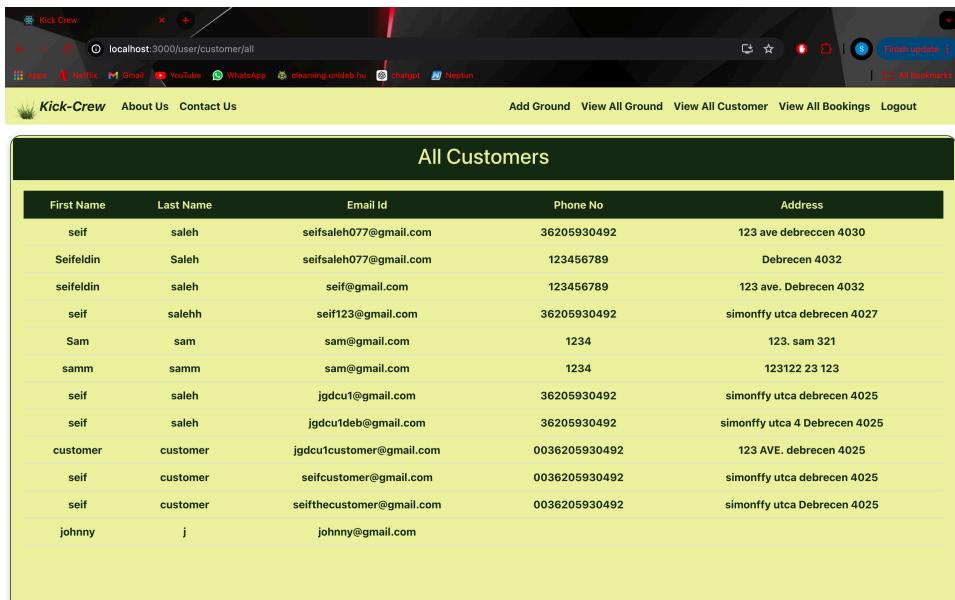
```

Figure 36. ViewAllGround.jsx

The ViewAllGround component displays a list of all grounds currently registered in the system. It fetches this data from the server upon loading using a call with axios. Each ground's details such as name, description, dimensions, and price are presented in a table format. For each ground entry, administrators have the option to delete it directly from the list. When deleting a ground, the system sends a request to the server with the ground's ID, and upon successful deletion, a toast notification confirms the action. If there are any issues during the

deletion process or if the server is unreachable, an error notification appears instead. The interface provides a seamless experience for administrators to manage grounds efficiently. This component ensures administrators can quickly view all grounds, perform necessary actions, and receive immediate feedback through notifications, enhancing their workflow management.

#### 4.12. View All Customers



## Figure 37 View All Customers

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- Explorer View:** Shows the project structure with files like `KICK-CREW-FRONTEND`, `src`, `page`, `ReviewComponent`, `UserComponent`, and various `.js` and `.css` files.
- Code Editor:** Displays the content of `ViewAllCustomer.jsx`. The code is a class-based component using functional state hooks (`useState`) and effects (`useEffect`). It includes asynchronous logic for fetching customer data from an API endpoint (`http://localhost:8080/api/user/customer/all`).
- Bottom Status Bar:** Shows the status "node" and other terminal-related information.

Figure 38  
ViewAllCustomer.jsx

The `ViewAllCustomer` component is designed to display a comprehensive list of all registered customers in a straightforward manner. It fetches customer data from the server using axios within the `useEffect` hook upon component rendering. This ensures that the displayed information, which includes essential details like first name, last name, email ID, phone number, and address for each customer, is always up-to-date.

The layout of the component utilizes a table format, making it easy for administrators to quickly scan and review customer details. This structured presentation enhances clarity and usability, facilitating efficient management of user accounts within the system. Overall, ViewAllCustomer aims to support administrators in effectively overseeing and managing customer information, promoting streamlined operations and user management workflows.

## 4.13. View All Bookings

Ground	Ground Name	Booking Id	Customer Name	Customer Contact	Booking Date	Booking Time Slot	Booking Status	Total Payable Amount	Verify Booking Status
	Doberdo Ground	PJ24VUEBXM	Sam sam	1234	2024-02-22	09:00 - 10:00 am	Cancel	9500.0	
	Doberdo Ground	AC3UXXVHUR	seif saleh	36205930492	2024-02-22	09:00 - 10:00 am	Cancel	9500.0	
	Doberdo Ground	FZVHUJ4VPU	seif saleh	36205930492	2024-02-22	09:00 - 10:00 am	Approved	9500.0	
	Doberdo Ground	A02H5BKAMR	customer customer	0036205930492	2024-02-22	11:00 - 12:00 am	Pending	9500.0	<button>Verify Booking</button>
	Doberdo Ground	ZJEVVVDVS2X	self customer	0036205930492	2024-06-20	02:00 - 03:00 pm	Pending	9500.0	<button>Verify Booking</button>
	Doberdo Ground	DSD1LMSQYG	self customer	0036205930492	2024-02-19	01:00 - 02:00 pm	Approved	9500.0	
	Kassai football	SCC72JDGK4	self customer	0036205930492	2024-02-12	01:00 - 02:00 pm	Pending	11500.0	<button>Verify Booking</button>

Figure 39 All bookings

```

src > BookingComponent > ViewAllBooking.jsx > <ViewAllBooking> > [t]e) retrieveAllBooking
  import { useState, useEffect } from "react";
  import axios from "axios";
  import React from "react";
  import { Link } from "react-router-dom";
  const ViewAllBooking = () => {
    const [allBookings, setAllBookings] = useState([]);
    let user = JSON.parse(sessionStorage.getItem("active-customer"));
    useEffect(() => {
      const getAllBooking = async () => {
        const allBooking = await retrieveAllBooking();
        if (allBooking) {
          setAllBookings(allBooking.bookings);
        }
      };
      getAllBooking();
    }, []);
    const retrieveAllBooking = async () => {
      const response = await axios.get(`http://localhost:8088/api/book/ground/fetch/all`);
      console.log(response.data);
      return response;
    };
    return (
      <div className="mt-3">
        <div className="card form-card ns-2 ne-2 mb-5 custom-bg border-color" style={{ height: "45rem", }}>
          <div className="card-header custom-bg text-center bg-color">
            <h2>All Bookings</h2>
          </div>
          <div>
            <table border="1">
              <thead>
                <tr>
                  <th>Ground</th>
                  <th>Ground Name</th>
                  <th>Booking Id</th>
                  <th>Customer Name</th>
                  <th>Customer Contact</th>
                  <th>Booking Date</th>
                  <th>Booking Time Slot</th>
                  <th>Booking Status</th>
                  <th>Total Payable Amount</th>
                  <th>Verify Booking Status</th>
                </tr>
              </thead>
              <tbody>
                <tr>
                  <td><img alt="Ground thumbnail 1"/></td>
                  <td>Doberdo Ground</td>
                  <td>PJ24VUEBXM</td>
                  <td>Sam sam</td>
                  <td>1234</td>
                  <td>2024-02-22</td>
                  <td>09:00 - 10:00 am</td>
                  <td>Cancel</td>
                  <td>9500.0</td>
                  <td></td>
                </tr>
                <tr>
                  <td><img alt="Ground thumbnail 2"/></td>
                  <td>Doberdo Ground</td>
                  <td>AC3UXXVHUR</td>
                  <td>seif saleh</td>
                  <td>36205930492</td>
                  <td>2024-02-22</td>
                  <td>09:00 - 10:00 am</td>
                  <td>Cancel</td>
                  <td>9500.0</td>
                  <td></td>
                </tr>
                <tr>
                  <td><img alt="Ground thumbnail 3"/></td>
                  <td>Doberdo Ground</td>
                  <td>FZVHUJ4VPU</td>
                  <td>seif saleh</td>
                  <td>36205930492</td>
                  <td>2024-02-22</td>
                  <td>09:00 - 10:00 am</td>
                  <td>Approved</td>
                  <td>9500.0</td>
                  <td></td>
                </tr>
                <tr>
                  <td><img alt="Ground thumbnail 4"/></td>
                  <td>Doberdo Ground</td>
                  <td>A02H5BKAMR</td>
                  <td>customer customer</td>
                  <td>0036205930492</td>
                  <td>2024-02-22</td>
                  <td>11:00 - 12:00 am</td>
                  <td>Pending</td>
                  <td>9500.0</td>
                  <td><button>Verify Booking</button></td>
                </tr>
                <tr>
                  <td><img alt="Ground thumbnail 5"/></td>
                  <td>Doberdo Ground</td>
                  <td>ZJEVVVDVS2X</td>
                  <td>self customer</td>
                  <td>0036205930492</td>
                  <td>2024-06-20</td>
                  <td>02:00 - 03:00 pm</td>
                  <td>Pending</td>
                  <td>9500.0</td>
                  <td><button>Verify Booking</button></td>
                </tr>
                <tr>
                  <td><img alt="Ground thumbnail 6"/></td>
                  <td>Doberdo Ground</td>
                  <td>DSD1LMSQYG</td>
                  <td>self customer</td>
                  <td>0036205930492</td>
                  <td>2024-02-19</td>
                  <td>01:00 - 02:00 pm</td>
                  <td>Approved</td>
                  <td>9500.0</td>
                  <td></td>
                </tr>
                <tr>
                  <td><img alt="Ground thumbnail 7"/></td>
                  <td>Kassai football</td>
                  <td>SCC72JDGK4</td>
                  <td>self customer</td>
                  <td>0036205930492</td>
                  <td>2024-02-12</td>
                  <td>01:00 - 02:00 pm</td>
                  <td>Pending</td>
                  <td>11500.0</td>
                  <td><button>Verify Booking</button></td>
                </tr>
              </tbody>
            </table>
          </div>
        </div>
      </div>
    );
  };
}

```

Figure 40 ViewAllBooking.jsx

The ViewAllBooking component is designed to present a comprehensive list of all bookings made within the system. It fetches this data using an asynchronous call to the server upon component initialization, ensuring the displayed information is always current. Each booking entry includes essential details such as the ground image, ground name, booking ID, customer name, contact information, booking date, time slot, booking status, total payable amount, and an option to verify the booking status if it's pending. When an admin approves a booking, the money is then deducted

```

src > main > java > com > kickcrew > dto > AddWalletMoneyRequestDTO.java
package com.kickcrew.dto;
import java.math.BigDecimal;
public class AddWalletMoneyRequestDTO {
    private int userId;
    private double walletAmount;
    // Getter for userId
    public int getUserId() {
        return userId;
    }
    // Setter for userId
    public void setUserId(int userId) {
        this.userId = userId;
    }
    // Getter for walletAmount
    public double getWalletAmount() {
        return walletAmount;
    }
    // Setter for walletAmount
    public void setWalletAmount(double walletAmount) {
        this.walletAmount = walletAmount;
    }
}

```

from the customer's wallet. This structured presentation in a table format allows administrators to efficiently monitor and manage booking activities, facilitating streamlined operational oversight and customer service management.

```

src > main > java > com > kickcrew > entity > Ground.java
package com.kickcrew.entity;
import java.persistence.Entity;
import java.persistence.GeneratedValue;
import java.persistence.GenerationType;
import java.persistence.Id;
@Entity
public class Ground {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private String description;
    private double width;
    private double height;
    private double length;
    private String image;
    private double price;
    private int status;
    // Getter for id
    public int getId() {
        return id;
    }
    // Setter for id
    public void setId(int id) {
        this.id = id;
    }
    // Getter for name
    public String getName() {
        return name;
    }
    // Setter for name
    public void setName(String name) {
        this.name = name;
    }
    // Getter for description
    public String getDescription() {
        return description;
    }
}

```

Figure 41  
AddWalletMoneyRequestDTO.java

Figure 42 Ground.java

The Ground class in Java serves as a fundamental component for managing football pitch bookings within a web application. It encapsulates crucial attributes like name, description, and physical dimensions (width, height, length) that define each football ground. This class is annotated with `@Entity`, marking it as a persistent entity in a database, typically managed through JPA (Java Persistence API). This setup ensures that instances of Ground can be seamlessly stored and retrieved from a relational database, facilitating efficient data management and retrieval for booking operations.