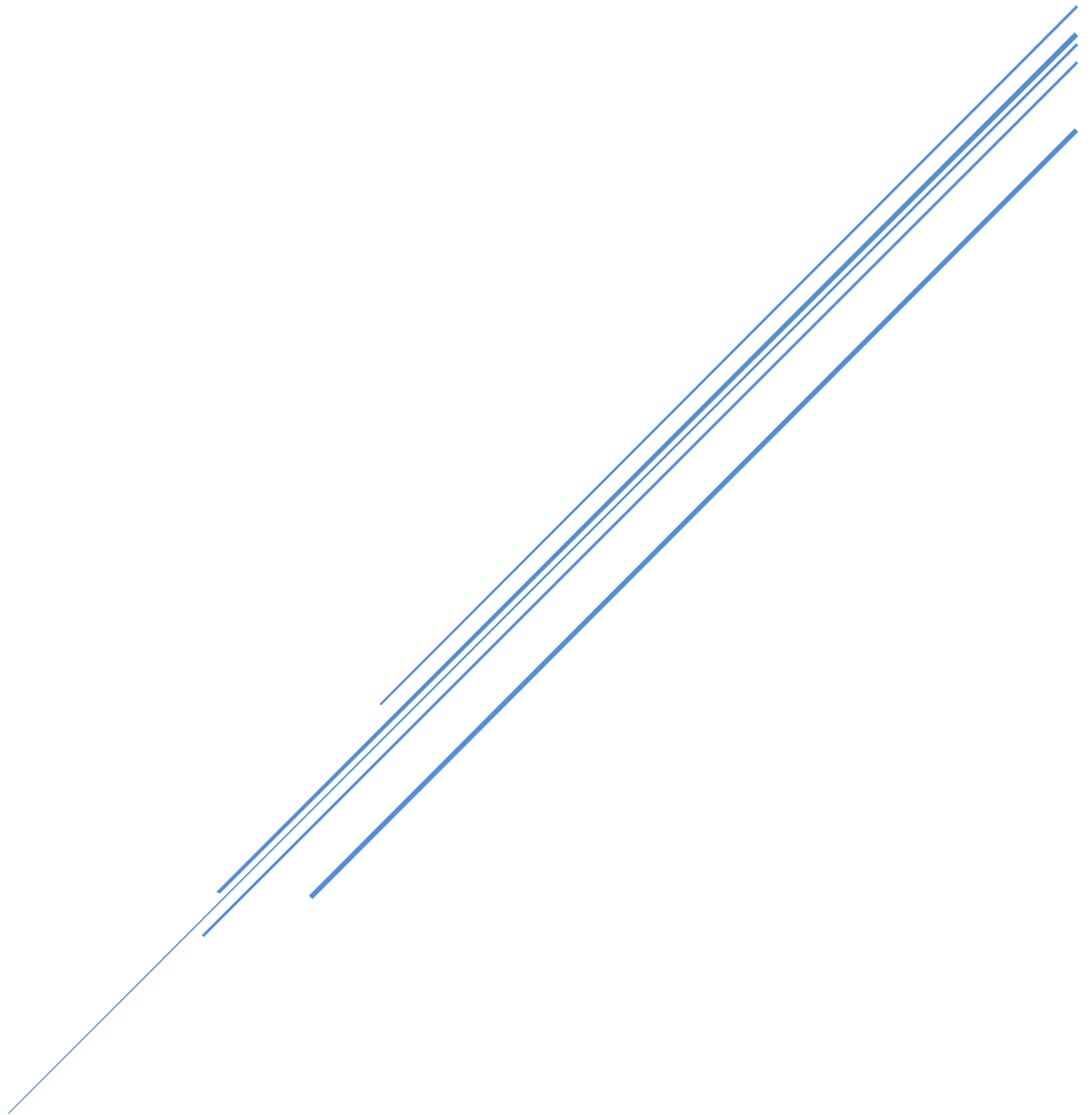


NWU Sports Manager

User Guide



CMPG 323

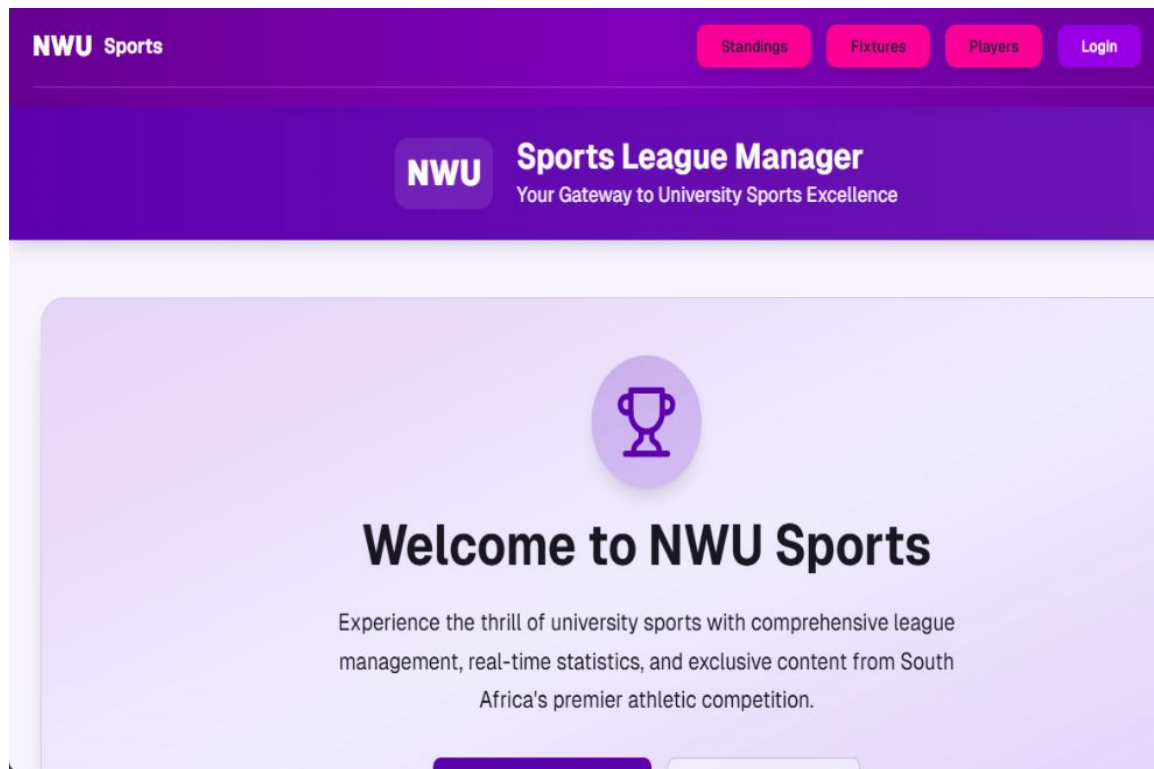
Table of Contents

1. Welcome page (Home Page)	___ 2
2. Team of the Week	___ 3
3. Coach of the Week	___ 3
4. Top Scorer (Home Page)	___ 4
5. Sign-In Page	___ 5
6. Admin Dashboard	___ 6
7. User management	___ 7
8. Coach Dashboard	___ 9
9. Standings	___ 10
10. Fixtures management	___ 12
11. Create fixtures	___ 14
12. Player Profiles	___ 15
13. Scout dashboard	___ 16
14. Player dashboard	___ 17

Home page

This section provides a visual and functional overview of the **NWU Sports Manager** user interface. Each page serves a unique purpose within the system, providing essential tools for administrators, coaches, and players. The interface has been designed with **ease of use, accessibility, and responsiveness** in mind, ensuring seamless navigation across desktop and mobile devices.

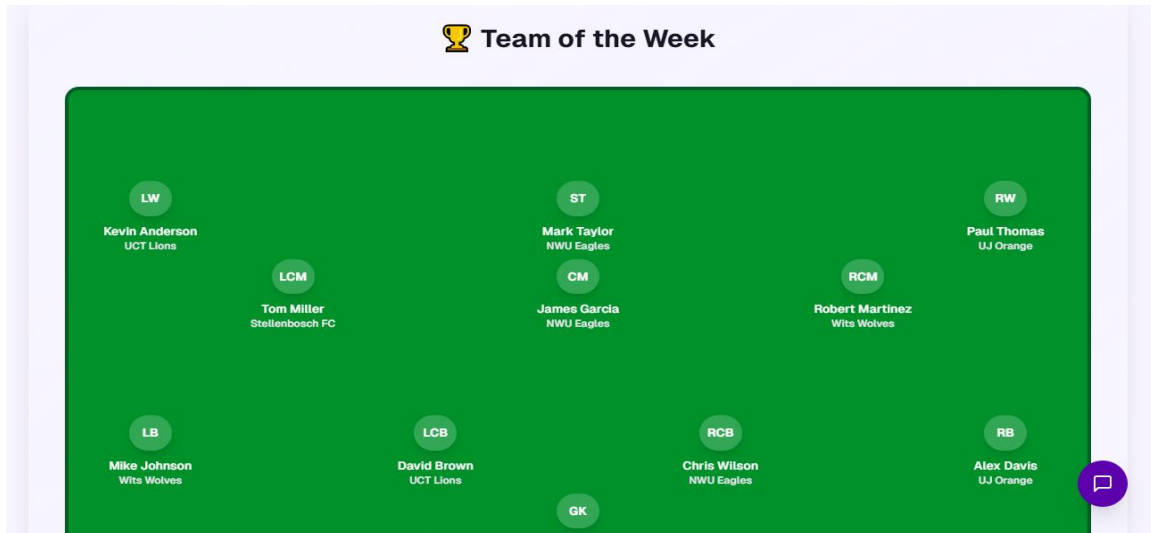
Users are presented with a **clean dashboard layout** that highlights key performance indicators, quick access links, and recent updates. Every module — from team management to fixture scheduling — is organized intuitively to minimize user effort and enhance productivity. Visual cues such as icons, color coding, and labels are used consistently to guide users through different tasks.



Team of the Week

The **Team of the Week** section showcases a curated lineup of players who delivered **outstanding performances** during the current week's fixtures. This feature recognizes top individual contributions across all teams, providing visibility and motivation for consistent high performance within the league.

Each selected player is displayed with key performance metrics, including **position, goals scored, assists, match ratings, and overall impact**. The lineup is automatically generated from the system's match data, ensuring that selections are **objective, data-driven, and based on verified results** recorded throughout the week.



Coach of the Week

The **Coach of the Week** section celebrates exceptional coaching performance across all teams for the current week. This recognition is based on a combination of **match results, team form, tactical decisions, and adherence to fair play principles**. The feature highlights coaches who have demonstrated strong leadership, strategic planning, and consistent improvement in their teams' overall performance.

Selection is determined through the system's performance analytics, which evaluate metrics such as **win ratios, goals scored, goals conceded, and disciplinary records**. The coach with the most impactful performance over the week is automatically featured, accompanied by relevant statistics and team achievements.

Coach of the Week

Sarah Mitchell (NWU Eagles)

Led team to crucial 3-2 victory with tactical masterclass

Featured Matches

NWU Eagles

VS

Wits Wolves

Upcoming

2024-01-15 • 15:00 • NWU Stadium

UCT Lions

VS

UP Tuks

Upcoming

2024-01-15 • 17:30 • UCT Grounds

League Standings

1

NWU Eagles

14W • 3D • 1L

45

points

2

Wits Wolves

13W • 3D • 2L

42

points

3

UCT Lions

12W • 2D • 4L

38

points

4

UP Tuks

10W • 4D • 4L

34

points

Top Scorer (Home Page)

The **Top Scorer** section, located on the **Home Page**, highlights the most outstanding players based on the number of goals scored across ongoing and completed leagues. This feature provides a **real-time leaderboard** that dynamically updates after every recorded fixture, ensuring that the rankings reflect the latest match outcomes and player performances.

Top Scorers

1

John Doe

NWU Eagles

18

goals

2

Aiden Clark

NWU Eagles

16

goals

3

Mike Smith

Wits Wolves

15

goals

4

Nathan White

Wits Wolves

14

goals

5

Lucas Taylor

UCT Lions

14

goals

Latest News

League Championship Race Heats Up

With only 5 matches remaining, the top 3 teams are separated by just 7 points.

2024-01-13

New Player Safety Protocols Introduced

Enhanced medical support and concussion protocols now in effect across all matches.

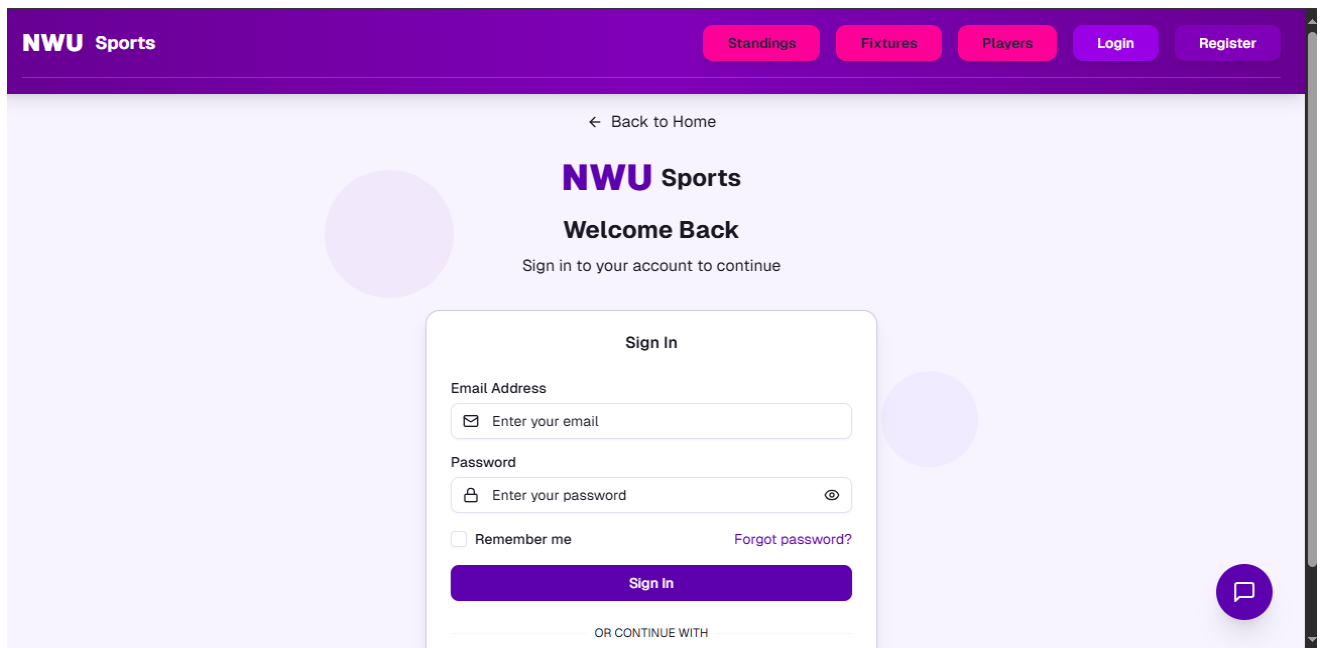
2024-01-10

4 | Page

Sign-In Page

The **Sign-In Page** provides secure authentication using verified **email and password** credentials. It serves as the main entry point into the system for administrators, coaches, and players. The authentication process is powered by **JWT (JSON Web Token)**-based session management, ensuring that user data and active sessions remain secure and tamper-proof throughout system usage.

The page is designed with a **clean and intuitive interface**, featuring labeled input fields, validation messages, and responsive feedback for incorrect login attempts. Upon successful authentication, users are **redirected to role-specific dashboards** — for instance, administrators are taken to the system management panel, while coaches and players access their respective modules for team or fixture management.



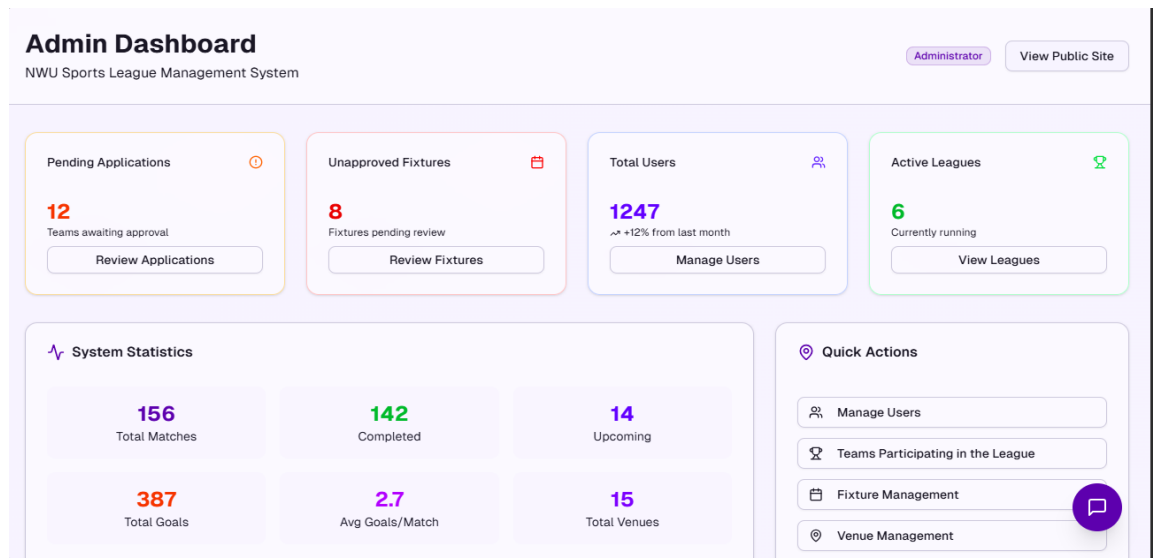
The screenshot displays the NWU Sports Sign-In page. At the top, a purple navigation bar contains the 'NWU Sports' logo on the left and five buttons ('Standings', 'Fixtures', 'Players', 'Login', 'Register') on the right. Below the navigation bar, a light purple background features a 'Back to Home' link, the 'NWU Sports' logo, and a 'Welcome Back' message with the instruction 'Sign in to your account to continue'. A central white sign-in form is overlaid, containing fields for 'Email Address' and 'Password', a 'Remember me' checkbox, a 'Forgot password?' link, and a 'Sign In' button. At the bottom of the form, it says 'OR CONTINUE WITH'. A purple chat bubble icon is located in the bottom right corner of the page.

Admin Dashboard

The **Admin Dashboard** serves as the **central hub** for system administrators, providing a comprehensive overview of all activities within the **NWU Sports Manager System**. From

this interface, administrators can **manage users, approve fixtures, monitor league statistics, and oversee system performance** in real time.

The dashboard displays key **summary metrics** such as the total number of registered users, active teams, upcoming fixtures, and accumulated goals. It also features dedicated panels for **pending approvals**, enabling quick review and validation of new team or fixture submissions.



User management

In the User Management section, administrators can view a list of all registered users, including their roles, status, and contact information. You can add new users, update existing user details, assign or change roles, and deactivate or delete accounts as needed. All changes are saved by clicking the “Save” or “Update” button, ensuring users have the appropriate access and permissions within the system.

← Back to Dashboard

User Management

Manage system users and permissions

+ Add New User

Q Search users by name, email, or phone...

All Roles

All Status

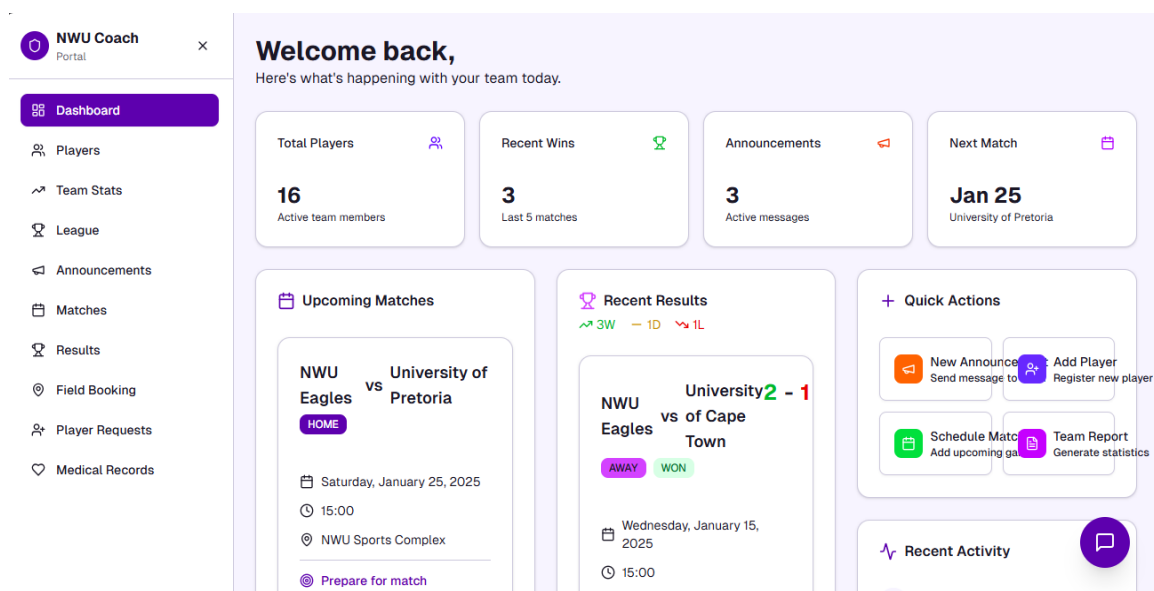
Users (34)

User	Contact	Roles	Organizations	Status	Joined	Actions
 Forget Nukeri	 forgetnukeri55@gmail.com  +27898765432		No organizations	 Active	2025/10/23	...
 Shaun Malesa	 shaun@gmail.com  +6243651345236		No organizations	 Active	2025/10/23	...
 Tinyiko Ndaba	 tinyiko@gmail.com  +27 897654321		No organizations	 Active	2025/10/23	...
 Dumazile Nkuna	 nkuna@gmail.com  +27 546457341		No organizations	 Active	2025/10/23	...
 Duma Zile	 duma@gmail.com  +23 243143256		No organizations	 Active	2025/10/23	...
 Sfiso Maluleke	 sfiso@gmail.com  +23 123456789		No organizations	 Active	2025/10/23	...
 Forget Nukeri	 nukeri@gmail.com  +27 762852622		No organizations	 Active	2025/10/23	...
 John Doe	 john@gmail.com  +27 78 25 27 26 5		No organizations	 Active	2025/10/23	...
 Tumi Doe	 tumi@gmail.com  +27 829090 163		No organizations	 Active	2025/10/23	...
 Forget Nukeri	 nhlamulofttee@gmail.com  +27762852631		No organizations	 Active	2025/10/23	...
 Forget Nukeri	 forgetnukeri@gmail.com  +27762852630		No organizations	 Active	2025/10/23	...
 Dudu Malesa	 dudu@gmail.com  0792876525		No organizations	 Active	2025/10/22	...
 Black Blue	 black@gmail.com  1234567890		No organizations	 Active	2025/10/22	...
 Mandla Khumalo  2001/12/15	 mandla@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Andile Mthethwa  2002/04/10	 andile@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Thulani Ndlovu  2001/06/25	 thulani@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Lwandile Shange  2002/01/18	 lwandile@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Nikanyiso Dube  2001/10/05	 nkanyiso@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Vuyo Mabena  2002/08/22	 vuyo@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Bongani Ngcobo  2001/11/11	 bongani@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Sanele Mkhize  2002/02/28	 sanele@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Siyabonga Zuma  2001/09/12	 siyabonga@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Jabulani Ndaba  2002/03/30	 jabulani@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Sipho Dlamini  2002/11/10	 sipho@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Thabo Mokoena  2002/07/19	 thabo@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Lethabo Motsepe  2002/07/19	 lethabo@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Kagiso Mahlangu  2001/02/05	 kagiso@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Blessing Nkosi  2002/05/12	 scout_sarah@gmail.com  +27 72 114 9090		No organizations	 Active	2025/10/22	...
 Tinyiko Ndou	 tinyikondou@gmail.com  +27 785 7 5582		No organizations	 Active	2025/10/22	...
 Forget Maluleke	 forgetmaluleke@gmail.com  +27 28526399		No organizations	 Active	2025/10/22	...
 Tumiso Maela	 coachnwu@gmail.com  +27 762852630		No organizations	 Active	2025/10/22	...
 Blessing Masemola	 bless@gmail.com  +27 76 285 2630		No organizations	 Active	2025/10/22	...
 Admin User	 forget@nwu.ac.za		No organizations	 Active	2025/10/22	...
 Admin User  1990/01/01	 admin@example.com		No organizations	 Active	2025/10/22	...

Coach Dashboard

The **Coach Dashboard** provides coaches with a centralized workspace to monitor team performance, manage player activities, and plan upcoming fixtures. Designed with **usability and strategic insight** in mind, it presents key team information at a glance, including **upcoming matches, recent results, win records, and player statistics**.

Coaches can access **quick-action tools** such as scheduling new matches, updating player performance data, or adding new players to the roster. The dashboard also integrates **real-time announcements and notifications**, allowing coaches to stay informed about league updates, fixture changes, and administrative notices.



Standings

The **Standings** section provides a **real-time league table** that ranks all participating teams based on their performance in ongoing competitions. Teams are evaluated using

key metrics such as **wins, losses, draws, goal difference, and total points**, ensuring a transparent and objective representation of league positions.

The table updates **automatically** whenever match results are recorded, reflecting the most current outcomes without requiring manual intervention. Each row in the standings includes team-specific details such as **matches played, goals scored, goals conceded, and overall ranking**, allowing administrators, coaches, and players to quickly assess performance at a glance.

[← Back](#)

NWU

League Standings

2024 University Sports League

Season 2024

University Sports League Table

18 Matches Played

#	Team	P	W	D	L	GF	GA	GD	Pts	Form
1	NWU Eagles	18	14	3	1	42	12	+30	45	W W W D W
2	Wits Wolves	18	13	3	2	38	15	+23	42	W L W W D
3	UCT Lions	18	12	2	4	35	20	+15	38	W W L W W
4	UP Tuks	18	10	4	4	32	22	+10	34	D W L D W
5	UJ Orange	18	9	5	4	28	25	+3	32	L D W D L
6	Stellenbosch FC	18	8	3	7	26	28	-2	27	L W L W L

Fixtures management

The **Fixtures Page** provides a comprehensive overview of all **upcoming and completed matches** for every team in the league. It displays essential details such as **match dates, times, venues, participating teams, and results**, enabling users to track schedules and performance at a glance.

For administrators and coaches, this page serves as a **management hub** for fixtures. Users can **schedule new matches, modify existing fixtures, approve proposed games**, and ensure that all scheduling conflicts are resolved efficiently. The interface includes interactive features such as **filters by team, date, or venue** to quickly locate specific matches, as well as search functionality to streamline navigation.

Fixture Management

Managing 4 fixtures

 Refresh

[+ Create Fixture](#)

[Back to Dashboard](#)

 Search fixtures by teams or venue...

All Leagues 

All Statuses 

All (4)

Pending (0)

Upcoming (2)

Needs Results (2) 

Completed (0)

NWU Hawks VS NWU Lions

 2025-11-06 at 17:00

NWU Stadium -
• NWU Stadium ground A

 NWU Senior League

 Admin User

 APPROVED

 Details

 Add Results

NWU Eagles VS NWU Hawks

 2025-10-30 at 15:30

NWU Stadium -
• NWU Stadium ground A

 NWU Senior League

 Admin User

 APPROVED

 Details

 Add Results

NWU Hawks VS NWU Lions

 2025-10-23 at 16:38

NWU Stadium -
• NWU Stadium ground A

 NWU Senior League

 Admin User

 Needs Results

 APPROVED

 Details

 Add Results

NWU Lions VS NWU Eagles

 2025-10-23 at 16:10

NWU Stadium -
• NWU Stadium ground A

 NWU Senior League

 Admin User

 Needs Results

 APPROVED

 Details

 Add Results

Create Fixture

To create a fixture, navigate to the “Create Fixture” page and fill out the required details, including the match date, time, venue, and the two teams participating. Ensure that the selected league and season are correct, as the fixture will be linked to them for scheduling and standings updates. Once all information is entered, click the “Save” or “Create” button to schedule the match, after which it will appear on the Fixture page and be available for all users to view.

The screenshot displays the 'Fixture Management' interface. At the top, there's a header with 'Managing 4 fixtures', a 'Refresh' button, a '+ Create Fixture' button, and a 'Back to Dashboard' button. Below the header, a search bar is present with the text 'Search fixtures by teams or'. A modal form titled 'Create New Fixture' is open in the center. The form contains the following fields: 'Home Team *' (dropdown), 'Away Team *' (dropdown), 'League *' (dropdown), 'Date *' (text input with a calendar icon), 'Time *' (text input with a clock icon), and 'Venue' (dropdown). At the bottom of the modal are 'Cancel' and 'Create Fixture' buttons. In the background, a list of fixtures is visible. The first fixture is 'NWU Hawks VS NWU Eagles' scheduled for '2025-11-06 at 17:00' at 'NWU Stadium - ground A', with a status of 'APPROVED'. The second fixture is 'NWU Hawks VS NWU Lions' scheduled for '2025-10-23 at 16:38' at 'NWU Stadium - ground A', with a status of 'Needs Results' and 'APPROVED'. The third fixture is 'NWU Lions VS NWU Eagles' scheduled for '2025-10-23 at 16:10' at 'NWU Stadium - ground A', also with a status of 'Needs Results' and 'APPROVED'. Each fixture entry includes a 'Details' button and an 'Add Results' button.

Fixture Management
Managing 4 fixtures

Refresh + Create Fixture Back to Dashboard

Search fixtures by teams or

All (4)

Create New Fixture

Home Team *
Select home team

Away Team *
Select away team

League *
Select league

Date *
yyyy/mm/dd

Time *
--:--

Venue
Select venue

Cancel Create Fixture

NWU Hawks VS NWU Eagles
2025-11-06 at 17:00
NWU Stadium - ground A
NWU Senior League
Admin User
APPROVED
Details Add Results

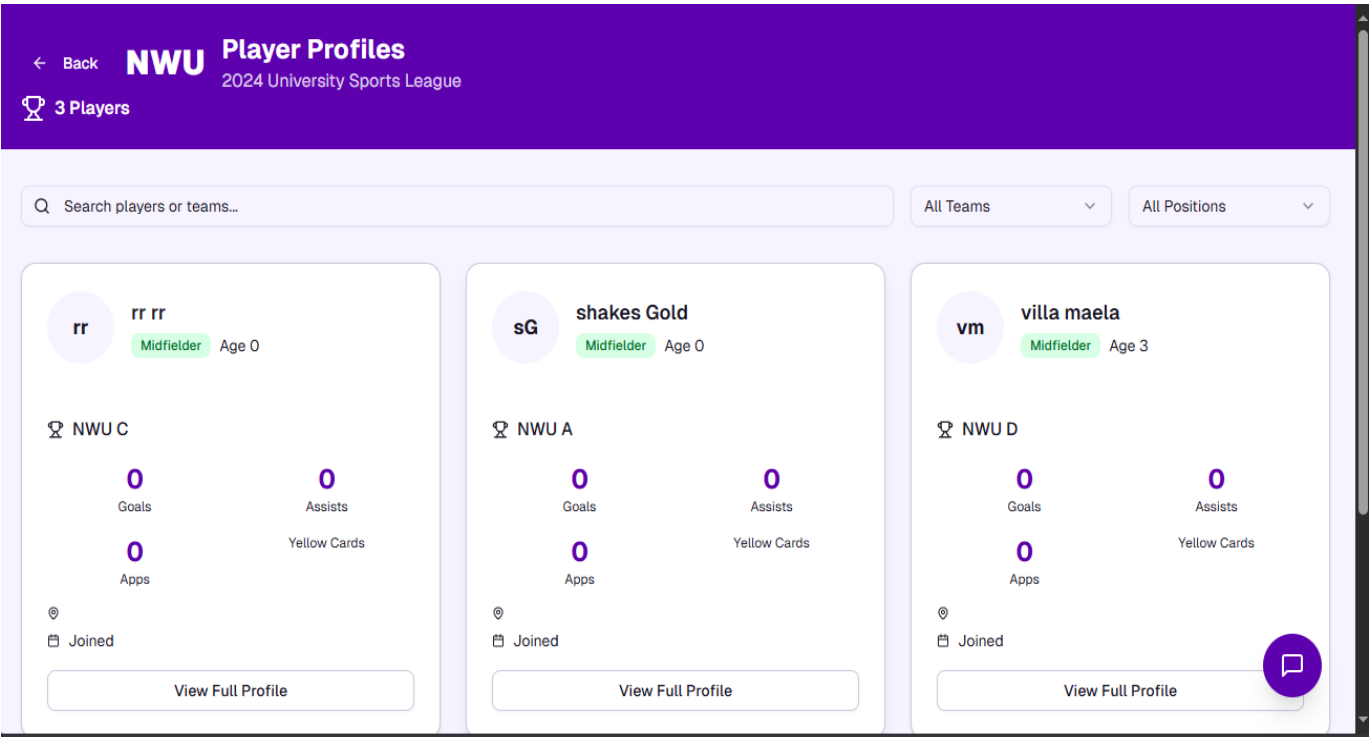
NWU Hawks VS NWU Lions
2025-10-23 at 16:38
NWU Stadium - ground A
NWU Senior League
Admin User
Needs Results APPROVED
Details Add Results

NWU Lions VS NWU Eagles
2025-10-23 at 16:10
NWU Stadium - ground A
NWU Senior League
Admin User
Needs Results APPROVED
Details Add Results

Player Profiles

The **Player Profiles** section offers a detailed overview of individual players’ performance and career statistics within the NWU Sports Manager System. Each profile displays **key metrics** such as **appearances, goals scored, assists, minutes played, and disciplinary records** (yellow/red cards), providing a comprehensive insight into a player’s contribution to their team.

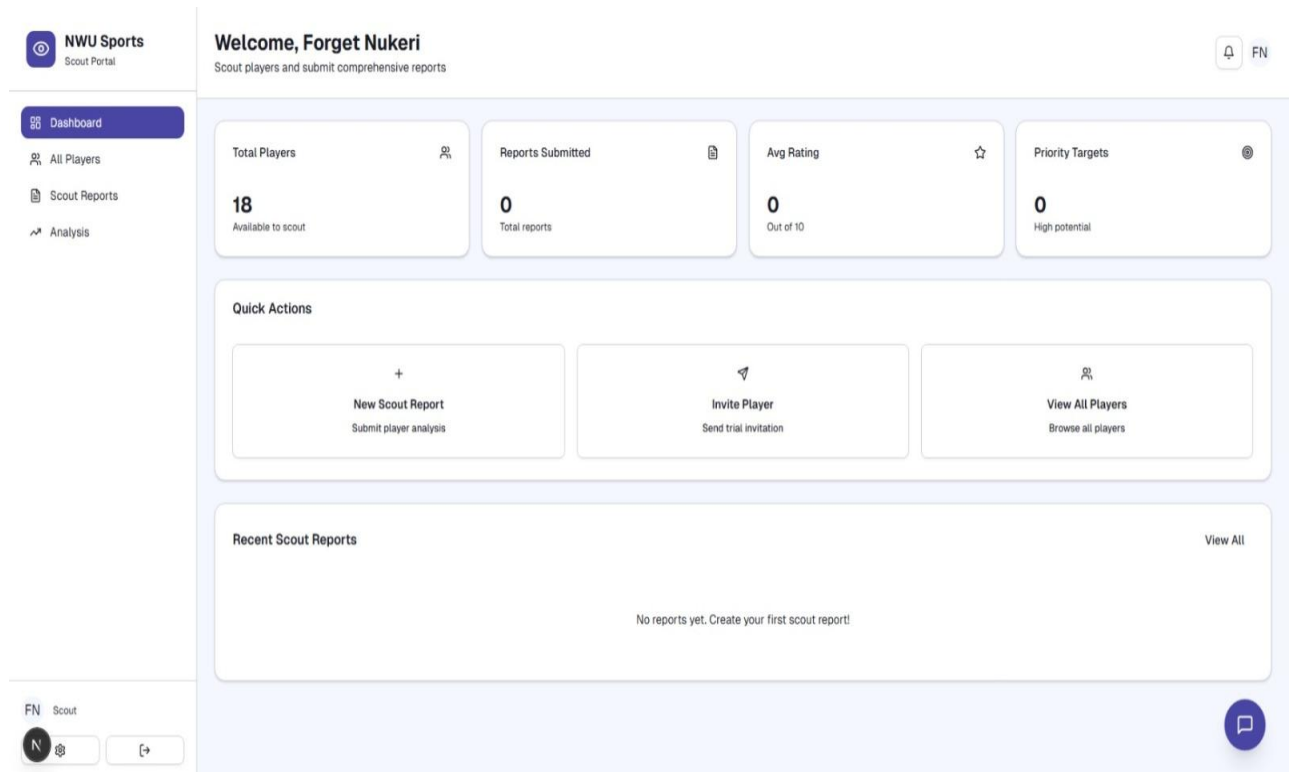
Profiles also include **personal information**, such as player name, team affiliation, position, and jersey number, alongside historical performance data that tracks improvement over time. Interactive elements allow users to **filter statistics by season, competition, or match type**, enabling a deeper understanding of performance trends and player consistency.



Scout dashboard

The Scout Dashboard provides an interface for users with the "scout" role to access league information and relevant scouting data. It ensures that only authenticated users with the proper role can view the dashboard, redirecting unauthorized users to the home or general

dashboard pages. While loading or verifying access, a loading indicator is displayed to enhance user experience and prevent unauthorized access.



Player dashboard

The Player Dashboard is designed for users with the "player" role, providing access to personalized league information, match schedules, and performance stats. It verifies that only authenticated players can access the dashboard, redirecting other users or unauthenticated visitors to the home or general dashboard pages. While the authentication

and role check are in progress, a loading indicator is displayed to ensure a smooth user experience.

NWU Player Portal

Nathan Mitchell
NWU Hawks

Welcome, Nathan

Track your performance and stay updated

Matches Played

1

This season

Goals

1

Total scored

Assists

0

Team contributions

Performance

100%

Overall rating

Team Announcements

View All

No announcements at this time

Upcoming Matches

NWU Hawks vs NWU Eagles

NWU Premier League

2025-10-28 • 05:23

Details

NWU Hawks vs PFC United

NWU Premier League 2025

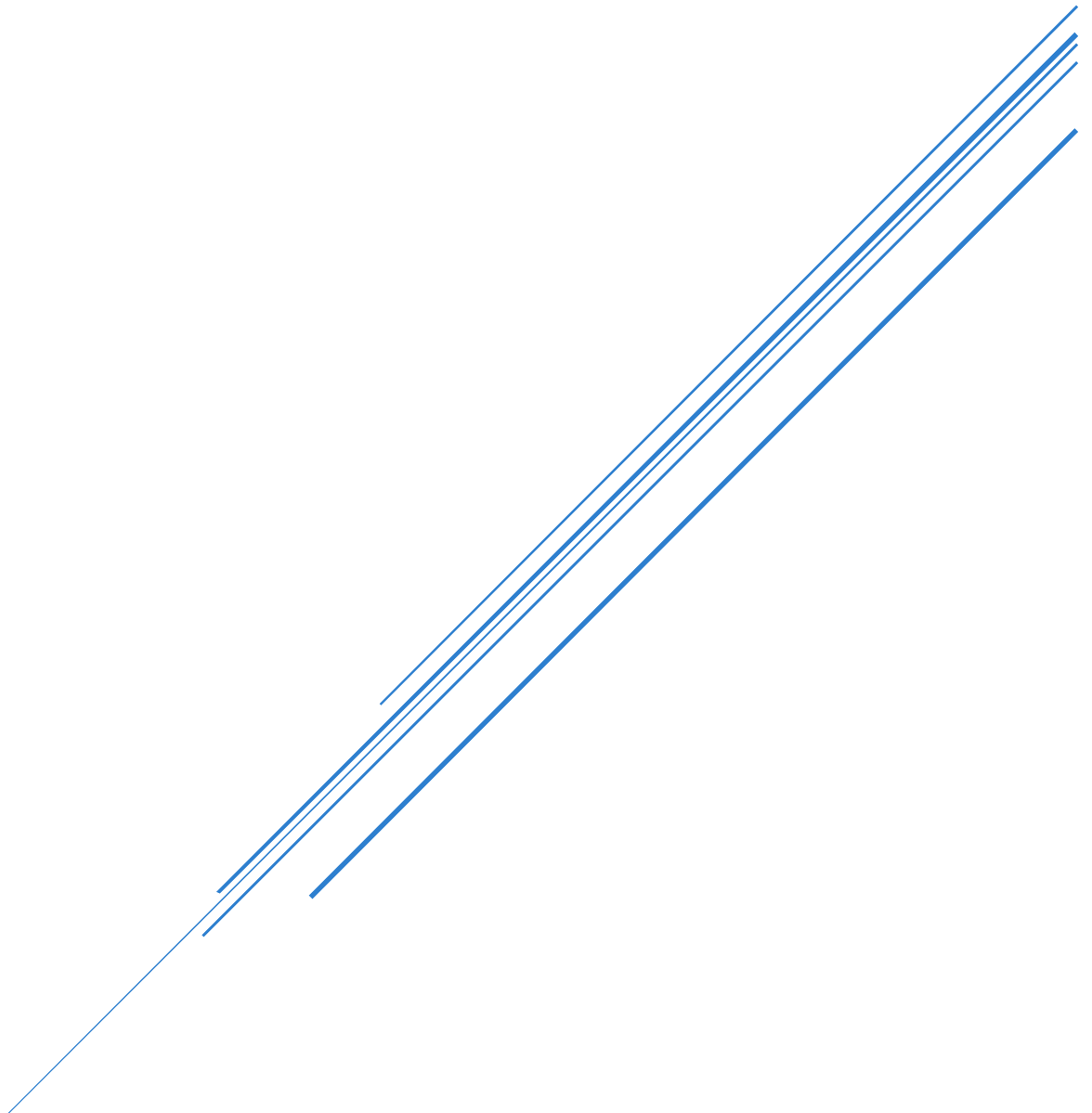
2025-10-29 • 06:30

Details

Trial Invitations

NWU Sports Manager

Developer Manual



CMPG 323

Table of Contents

1. Mysql database connection	___ 2
2. Tech stack	___ 2
3. UI components	___ 3
4. API Integration	___ 3
5. Authentication flow	___ 3
6. Routing	___ 3
7. State management	___ 3
8. Development setup	___ 3
9. Environment variables	
Building for production	___ 4
Testing	___ 4
Best practices	___ 4
10. Admin Dashboard component	___ 4-8)
11. Coach Dashboard	___ 9-12)
12. Fixtures	___ 13-15)
13. Forget Password page	___ 15-19)
14. Home page	___ 19-22)
15. Login	___ 22-23)
16. Player	___ 23-24)
17. Scout Dashboard	___ 24-25)
18. Standings	___ 25-26)
19. Support page	___ 26-27)
20. Terms page	___ 27-28)
21. Rules and regulations	___ 28-29)
22. Privacy and policy	___ 29-30)

MySQL database connection

This code defines a **MySQL database connection utility** using the `mysql2/promise` library, enabling asynchronous queries with promises. It creates a **connection pool** that allows multiple simultaneous database connections, improving performance and resource management. The connection details—host, user, password, and database name—are read from environment variables, with default values provided for local development. The pool configuration includes `waitForConnections`, `connectionLimit`, and `queueLimit` to control connection behavior under high load. Finally, the pool is **exported** as a module, making it reusable across the backend for executing SQL queries safely and efficiently.

```
1 // MySQL connection utility
2 const mysql = require('mysql2/promise');
3
4 const pool = mysql.createPool({
5   host: process.env.DB_HOST || 'localhost',
6   user: process.env.DB_USER || 'root',
7   password: process.env.DB_PASSWORD || '',
8   database: process.env.DB_NAME || 'NWU_Sports_League',
9   waitForConnections: true,
10  connectionLimit: 10,
11  queueLimit: 0
12 });
13
14 module.exports = pool;
```

Tech Stack

Project Structure

app/

- |— admin/
- |— coach/
- |— players/
- |— auth/
- |— components/
- |— lib/
- |— styles/

Authentication Flow

Login via /api/auth/login

JWT token stored in HTTP-only cookies

Next.js middleware for route protection

Session management with token refresh

Routing

/, /login, /register, /admin/*, /coach/*, /players/*, /fixtures, /standings

State Management

Local: useState

Global: React Context API (auth, UI, cached API data)

UI Components

Layout: MainLayout, DashboardLayout

Navigation: Sidebar, TopNav

Data Display: DataTable, Card, Badge

Forms: Input, Select, DatePicker

Feedback: Toast, Alert, Dialog

Data Visualization: StatsCard,

API Integration

Fetch API with SWR

JWT token via cookies

Error boundaries for API errors

Development Setup

Prerequisites: Node.js 18+, npm/yarn, backend server

Install dependencies: npm install

.env.local with API and NextAuth configs

Run: npm run dev (frontend) + backend server

Environment Variables

Building for Production

npm run build

npm start

Ensure environment variables are set

Testing

Unit: Jest + React Testing Library

Component: Storybook + Chromatic

E2E: Cypress

Best Practices

Component design: Atomic Design

State management: lift state, context for global state

Performance: lazy load, code splitting

Admin Dashboard Component

1. Imports and Setup

This section imports all required **UI components, icons, routing, and React hooks**. It uses "use client" to enable client-side rendering in Next.js. The imported cards, badges, and buttons structure the dashboard layout. Icons from lucide-react provide visual cues for different metrics and actions. React hooks useState and useEffect handle **state management and side effects** for fetching dashboard data.

```
'use client'

import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card"
import { Badge } from "@components/ui/badge"
import { Button } from "@components/ui/button"
import { Users, Calendar, Trophy, MapPin, AlertCircle, CheckCircle, Clock, Activity, TrendingUp,
import Link from "next/link"
import { useState, useEffect } from "react"
```

2. TypeScript Interfaces

These interfaces define the **shape of the data** used in the dashboard. DashboardMetrics holds main metrics and statistics for the admin. ActivityItem represents individual recent activities

with action type, timestamp, and user. `SystemStats` provides aggregate match and goal statistics. Using TypeScript interfaces ensures **type safety** and **better developer experience** when accessing or updating state.

```
interface DashboardMetrics { ... }  
interface ActivityItem { ... }  
interface SystemStats { ... }
```

3. State Initialization

Here, two pieces of state are defined using React's `useState`. `metrics` stores all **dashboard data**, including team applications, fixtures, users, and system stats. `loading` manages the **loading state** to display a spinner while data is fetched. Initial values are set to **default zero or empty arrays** to prevent errors. This state setup allows the dashboard to **reactively update** when new data arrives.

```
const [metrics, setMetrics] = useState<DashboardMetrics>({ ... })  
const [loading, setLoading] = useState(true)
```

4. Data Fetching with `useEffect`

This effect fetches **all dashboard metrics** when the component mounts. Multiple endpoints are called in parallel using `Promise.all` to improve efficiency. Responses are converted to JSON and then stored in the `metrics` state. Error handling ensures that **failed requests do not break the UI**. Finally, `loading` is set to `false` to render the dashboard once data is available.

```
useEffect(() => {  
  const fetchDashboardData = async () => { ... }  
  fetchDashboardData()  
}, [])
```

5. Helper Functions

These helper functions dynamically return **icons and badge colors** based on activity type. They improve **UI clarity** by visually distinguishing pending, approved, or completed activities. The functions ensure **consistent styling** throughout the dashboard. By abstracting this logic, code is more maintainable and reusable. They also simplify JSX in the main return block, keeping it **clean and readable**.

```
const getActivityIcon = (type: string) => { ... }
const getActivityBadgeColor = (type: string) => { ... }
```

6. Loading State

This section displays a **loading spinner** while the dashboard data is being fetched. It ensures users do not see incomplete or empty metrics. Styling includes a centered spinner and muted text for clarity. Conditional rendering with `if (loading)` prevents the main dashboard from loading prematurely. This improves **user experience and perceived performance**.

```
if (loading) {
  return (
    <div className="min-h-screen ..." >
      <div className="animate-spin ..." ></div>
      <p>Loading Admin Dashboard...</p>
    </div>
  )
}
```

7. Dashboard Layout and Metrics Cards

This block displays **key metrics in responsive cards** for pending applications, unapproved fixtures, total users, and active leagues. Each card uses **CardHeader, CardContent, and icons** for consistent design. Metrics are populated dynamically from the `metrics` state. Cards include buttons and links for **quick access to detailed management pages**. The grid layout adapts across devices, making the dashboard **responsive and user-friendly**.

```
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8">
  <Card> ... </Card>
  <Card> ... </Card>
  <Card> ... </Card>
  <Card> ... </Card>
</div>
```

8. System Statistics and Quick Actions

These sections provide a **summary of system-wide stats** like total matches, completed matches, goals, and venues. Quick Actions give administrators **fast navigation** to important pages like Live Scores, Users, Teams, Fixtures, and Venues. Data is dynamically pulled from the `metrics` state for real-time accuracy. UI cards use **color coding and icons** for visual clarity. This section enhances **dashboard efficiency and workflow management**.

```
<Card> ... System Statistics ... </Card>
<Card> ... Quick Actions ... </Card>
```

9. Recent Activity

The Recent Activity card lists all **user actions and system events** in a chronological feed. Each activity shows an icon, badge, description, user, and timestamp. The card dynamically handles **empty states** with a friendly message. Conditional styling and icon selection make it easy to **distinguish activity types**. This feature provides administrators with **real-time monitoring** of system usage and changes.

```
<Card>
  {metrics.recentActivity.map((activity) => ( ... ))}
</Card>
```

DashboardStats Component (Coach Dashboard)

1. Imports and Setup

This section imports React hooks `useState`, `useEffect`, and `useMemo` for state management and memoization. It also imports **TypeScript types** (`Player`, `Match`, etc.) for type safety. UI

components like Card, CardHeader, and CardContent are used to build the dashboard layout. Icons from lucide-react visually represent each metric. "use client" ensures this component is rendered on the client side in Next.js.

```
"use client"

import { useEffect, useState, useMemo } from "react"
import type { Player, Match, Announcement, Fixture } from "@lib/mockData"
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card"
import { Users, Trophy, Megaphone, Calendar } from "lucide-react"
```

2. Props and Interfaces

This interface defines the **data the component expects** from its parent. Optional arrays include players, recent matches, announcements, and upcoming fixtures. Using TypeScript ensures the component **handles undefined or empty data safely**. It enforces **strong typing** for all dashboard metrics. This structure improves maintainability and reduces runtime errors.

```
interface DashboardStatsProps {
  players?: Player[]
  recentMatches?: Match[]
  announcements?: Announcement[]
  upcomingMatches?: Fixture[]
}
```

3. State Initialization

The state currentTeam stores the **team associated with the logged-in coach**. It is initially null until a valid user is retrieved. useState allows the component to **reactively update metrics** based on the coach's team. This enables filtering of players, matches, and statistics dynamically. It ensures the dashboard only shows **relevant team data**.

```
const [currentTeam, setCurrentTeam] = useState<string | null>(null)
```

4. Fetching Current Team

This effect retrieves the **logged-in coach's information** from `localStorage`. It checks the user role and ensures the team is a valid string. `setCurrentTeam` updates the state with the coach's assigned team. This logic ensures the dashboard **filters data specific to the coach's team**. By using `useEffect`, it runs only once when the component mounts.

```
useEffect(() => {
  const storedUser = localStorage.getItem("currentUser")
  if (storedUser) {
    const user = JSON.parse(storedUser)
    if (user.role === "coach" && typeof user.team === "string") {
      setCurrentTeam(user.team)
    }
  }
}, [])
```

5. Filtering Team Data (useMemo)

`useMemo` optimizes performance by **caching filtered results** for team players, recent matches, and upcoming fixtures. It filters only the data relevant to the `currentTeam`. Recent matches are **sorted by date** and limited to the latest five. Upcoming matches are filtered to include **only future fixtures**. This ensures metrics displayed are **accurate, up-to-date, and specific to the coach's team**.

```
const teamPlayers = useMemo(() => { ... }, [players, currentTeam])
const teamRecentMatches = useMemo(() => { ... }, [recentMatches, currentTeam])
const teamUpcomingMatches = useMemo(() => { ... }, [upcomingMatches, currentTeam])
```

6. Calculating Metrics

Here, key dashboard metrics are computed for display. `totalPlayers` counts all active players on the coach's team. `recentWins` counts victories in the last five matches. `totalAnnouncements` tracks system messages relevant to all users. `nextMatch` identifies the upcoming match or sets a fallback of `null`. These calculations feed into the **cards displayed on the dashboard**.

```
const totalPlayers = teamPlayers.length
const recentWins = teamRecentMatches.filter(m => m.result?.status === "won").length
const totalAnnouncements = announcements.length
const nextMatch = teamUpcomingMatches[0] ?? null
```

7. Stats Array for Rendering

This array organizes **all the metrics for display** in dashboard cards. Each object contains a title, value, icon, description, and color. The nextMatch object includes a formatted date and matchup details. This abstraction allows **mapping over the array to render UI dynamically**. It makes the dashboard component **scalable and easy to maintain**.

```
const stats = [
  { title: "Total Players", value: totalPlayers, icon: Users, description: "Active team members",
  { title: "Recent Wins", value: recentWins, icon: Trophy, description: "Last 5 matches", color: "
  { title: "Announcements", value: totalAnnouncements, icon: Megaphone, description: "Active messa
  { title: "Next Match", value: nextMatch ? ... : "TBD", icon: Calendar, description: nextMatch ?
]
```

8. Rendering Dashboard Cards

The component maps over the stats array to **render a responsive grid of cards**. Each card displays the metric title, value, icon, and description. Tailwind classes ensure **responsive layout across devices**. Using dynamic mapping keeps the JSX **clean and reusable**. The result is a visually consistent, data-driven **coach dashboard overview**.

```
return (
  <div className="grid gap-4 md:grid-cols-2 lg:grid-cols-4">
    {stats.map(stat => {
      const Icon = stat.icon
      return (
        <Card key={stat.title}>
          <CardHeader> ... </CardHeader>
          <CardContent> ... </CardContent>
        </Card>
      )
    })}
  </div>
)
```

FIXTURES

The FixturesPage component is a client-side React/Next.js page that displays upcoming fixtures, recent results, and key match statistics for the NWU Sports League. It fetches data from multiple API endpoints, including upcoming fixtures, finalized match results, and quick stats, and stores them in local state variables (upcomingFixtures, recentResults, stats) while showing a

loading spinner during asynchronous requests. The component uses Tabs to toggle between upcoming fixtures and recent results, rendering each fixture or result as a responsive card with team names, scores, venue, date, time, and status badges. Helper functions like `formatDate`, `formatTime`, `getRound`, `getStatusBadge`, and `generateHighlights` handle date formatting, week calculation, status visualization, and automatic generation of match highlights based on scores. Additionally, the page includes a quick stats section displaying matches scheduled this week, completed last week, and average goals per match, providing users with an at-a-glance overview of league activity.

```
"use client"

import { useState, useEffect } from "react"
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card"
import { Badge } from "@components/ui/badge"
import { Button } from "@components/ui/button"
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@components/ui/tabs"
import { Calendar, Clock, MapPin, ArrowLeft, Filter, Loader2 } from "lucide-react"
import Link from "next/link"

// Types based on your database schema
interface Fixture {
  fixture_id: number
  league_id: number
  home_team_id: number
  away_team_id: number
  scheduled_at: string
  venue_field_id?: number
  booking_id?: number
  created_by: number
  approval_status: string
  approved_by?: number
  approved_at?: string
  home_team_name: string
  away_team_name: string
  venue_name?: string
  field_name?: string
  home_score?: number
  away_score?: number
}
```

```

interface MatchResult extends Fixture {
  match_id: number
  started_at?: string
  ended_at?: string
  highlights?: string[]
  attendance?: number
}

const getStatusBadge = (status: string) => {
  switch (status) {
    case "SCHEDULED":
      return <Badge variant="default">Upcoming</Badge>
    case "IN_PROGRESS":
      return (
        <Badge variant="destructive" className="animate-pulse">
          Live
        </Badge>
      )
    case "FINAL":
      return <Badge variant="secondary">Final</Badge>
    case "POSTPONED":
      return <Badge variant="outline">Postponed</Badge>
    case "CANCELLED":
      return <Badge variant="outline">Cancelled</Badge>
    default:

```

ForgotPasswordPage Component

1. Imports and Setup

- Imports React, state hooks, UI components (Card, Button, Input, Label) and icons.
- `useState` is used for managing form input, loading, and submission state.
- Next.js `Link` is used for client-side navigation.

```
"use client"
import type React from "react"
import { useState } from "react"
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card"
import { Button } from "@components/ui/button"
import { Input } from "@components/ui/input"
import { Label } from "@components/ui/label"
import { ArrowLeft, Mail, CheckCircle } from "lucide-react"
import Link from "next/link"
```

2. Component State

- email: stores the user's input.
- isLoading: tracks if the reset process is in progress.
- isSubmitted: tracks if the reset link has been sent successfully

```
const [email, setEmail] = useState("")
const [isLoading, setIsLoading] = useState(false)
const [isSubmitted, setIsSubmitted] = useState(false)
```

3. Form Submission Handler

- Prevents default form submission behavior.
- Simulates sending a reset link with a 1.5-second delay.
- Updates isSubmitted to show the confirmation screen.

```
const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault()
  setIsLoading(true)
  setTimeout(() => {
    setIsLoading(false)
    setIsSubmitted(true)
  }, 1500)
}
```

4. Confirmation Screen (After Submission)

- Shows a success message after submitting the email.
- Provides buttons to go back to login or try another email.

```
if (isSubmitted) {
  return (
    <div className="min-h-screen ..." >
      <Card className="shadow-lg text-center">
        <CardContent>
          <CheckCircle className="w-16 h-16 text-green-500 mx-auto mb-4" />
          <h1>Check Your Email</h1>
          <p>We've sent a password reset link to {email}</p>
          <Button asChild><Link href="/login">Back to Sign In</Link></Button>
          <Button variant="outline" onClick={() => setIsSubmitted(false)}>Try Different Email</Button>
        </CardContent>
      </Card>
    </div>
  )
}
```

5. Forgot Password Form

- Displays the form to enter an email address.
- Includes a submit button and link back to the login page.
- Uses state to disable the button and show a loading message while sending.

```

<div className="w-full max-w-md relative z-10">
  <Card className="shadow-lg">
    <CardHeader><CardTitle>Forgot Password</CardTitle></CardHeader>
    <CardContent>
      <form onSubmit={handleSubmit} className="space-y-4">
        <Label htmlFor="email">Email Address</Label>
        <Input id="email" type="email" placeholder="Enter your email" value={email} onChange={(e)
        <Button type="submit" disabled={isLoading}>{isLoading ? "Sending reset link..." : "Send R
      </form>
      <p>Remember your password? <Link href="/login">Sign in</Link></p>
    </CardContent>
  </Card>
</div>

```

HOME

Imports, Types, and Data Fetching Functions

This part sets up all the libraries, icons, and components needed for the homepage, including React, Next.js, and UI components like cards, buttons, and badges. It also defines TypeScript interfaces to strictly type the data fetched from APIs, such as matches, standings, top scorers, and news. Finally, it contains functions to fetch live matches, upcoming fixtures, and calculate the current match minute, handling errors and formatting data for the UI.

```

"use client"

import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card"
import { Badge } from "@components/ui/badge"
import { Button } from "@components/ui/button"
import { MessageSquare } from "lucide-react"
import {
  Trophy, Calendar, Users, Play, ImageIcon, Mail, Phone, MapPin,
  Facebook, Twitter, Instagram, Youtube, Star, Award, Target, Clock, Zap
} from "lucide-react"
import Link from "next/link"
import { useState, useEffect } from "react"
import TeamOfTheWeek from "@components/ui/TeamOfTheWeek"

// Types based on your database schema
interface Fixture {
  fixture_id: number

```

```

  fixture_id: number
  home_team_id: number
  away_team_id: number
  scheduled_at: string
  venue_field_id?: number
  home_team_name: string
  away_team_name: string
  venue_name?: string
  field_name?: string
  home_score?: number
  away_score?: number
  status_key: string
}

interface LiveMatch extends Fixture {
  homeScore: number
  awayScore: number
  minute: number
  status: string

```

HomePage Component State and Data Fetching Hooks

This section initializes React states for live matches, fixtures, standings, top scorers, news, and team-of-the-week data, along with loading indicators and selected league. `useEffect` hooks are

used to fetch leagues on page load, update all home page data when a league is selected, refresh live matches every minute, and fetch the next upcoming match. These hooks ensure the UI always displays current, accurate data while maintaining responsiveness and loading feedback.

```
export default function HomePage() {
  const [liveMatches, setLiveMatches] = useState<LiveMatch[]>([])
  const [fixtures, setFixtures] = useState<Fixture[]>([])
  const [standings, setStandings] = useState<Standing[]>([])
  const [topScorers, setTopScorers] = useState<TopScorer[]>([])
  const [news, setNews] = useState<NewsArticle[]>([])
  const [teamOfTheWeek, setTeamOfTheWeek] = useState<TeamOfTheWeekData | null>(null)
  const [leagues, setLeagues] = useState<League[]>([])
  const [selectedLeague, setSelectedLeague] = useState<string>("")
  const [nextMatch, setNextMatch] = useState<Fixture | null>(null)
  const [loading, setLoading] = useState(true)

  // Fetch leagues first
  useEffect(() => {
    const fetchLeagues = async () => {
```

HomePage JSX Layout (Header, Main, Footer)

This part defines the visual structure of the homepage, including a header with navigation, a main content area with hero, live matches, standings, top scorers, news, and team-of-the-week sections. It also includes a footer with contact information, social links, and quick stats, all styled with TailwindCSS and custom UI components. Essentially, this part translates all fetched and processed data into a user-friendly, interactive interface.

```

return (
  <div className="min-h-screen bg-gradient-to-br from-background via-muted/5 to-background">
    <div className="fixed inset-0 bg-grid-pattern opacity-[0.02] pointer-events-none"></div>

    {/* Header */}
    <header className="relative bg-gradient-to-r from-primary via-primary/95 to-primary/90 text-white">
      <div className="container mx-auto px-6">
        <div className="flex items-center justify-between py-6">
          <div className="flex items-center space-x-6">
            <div className="bg-primary-foreground/10 backdrop-blur-sm rounded-xl p-3">
              <div className="text-3xl font-black text-primary-foreground">NWU</div>
            </div>
            <div>
              <h1 className="text-3xl font-bold text-balance">Sports League Manager</h1>
              <p className="text-primary-foreground/90 text-lg font-medium">
                Your Gateway to University Sports Excellence
              </p>
            </div>
          </div>
        </div>
      </div>
    </header>
  </div>
)

```

LOGIN

React client component for the login page, using `useState` to manage email, password, visibility, "remember me," loading state, and error messages. It includes a `handleSubmit` function that performs client-side validation, sends login credentials to the backend API, handles the response, stores minimal user data in `localStorage`, and redirects users based on their primary role. The JSX renders a styled login form with email/password fields, a show/hide password toggle, remember me checkbox, error messages, and links for registration and password recovery, all styled with TailwindCSS and custom UI components.

```

"use client"

import type React from "react"
import { useState } from "react"
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card"
import { Button } from "@components/ui/button"
import { Input } from "@components/ui/input"
import { Label } from "@components/ui/label"
import { Checkbox } from "@components/ui/checkbox"
import { ArrowLeft, Eye, EyeOff, Mail, Lock } from "lucide-react"
import Link from "next/link"
import { useRouter } from "next/navigation"

export default function LoginPage() {
  const [email, setEmail] = useState("")
  const [password, setPassword] = useState("")
  const [showPassword, setShowPassword] = useState(false)
  const [rememberMe, setRememberMe] = useState(false)
  const [isLoading, setIsLoading] = useState(false)
  const [error, setError] = useState("")
  const router = useRouter()

```

PLAYER

This component is a protected page for players, using the `useAuth` hook to get the current user and loading state, and `useRouter` to handle navigation. It first ensures the component is mounted with `useEffect`, then checks if the user is authenticated and has the "player" role, redirecting non-players to the generic dashboard or unauthenticated users to the home page. While loading or if the user is invalid, it shows a loading message; otherwise, it renders the `PlayerDashboard` component.

```

"use client"

import { useEffect, useState } from "react"
import { useRouter } from "next/navigation"
import { useAuth } from "@lib/auth-context"
import PlayerDashboard from "@app/player/dashboard/page"

export default function PlayerDashboardPage() {
  const { user, isLoading } = useAuth()
  const router = useRouter()
  const [mounted, setMounted] = useState(false)

  useEffect(() => {
    setMounted(true)
  }, [])

  useEffect(() => {
    if (!isLoading && mounted) {
      if (!user) {
        router.push("/")
      } else if (user.role !== "player") {
        router.push("/dashboard")
      }
    }
  }, [user, isLoading, mounted, router])

  if (!mounted || isLoading || !user || user.role !== "player") {
    return (
      <div className="flex items-center justify-center min-h-screen">

```

SCOUT Dashboard

This component is a protected page for scouts, using the `useAuth` hook to get the current user and loading state, and `useRouter` for navigation. It checks that the component is mounted and verifies that the logged-in user has the "scout" role, redirecting non-scout users to the generic dashboard and unauthenticated users to the home page. While loading or if the user is unauthorized, it shows a loading message; otherwise, it renders the `ScoutDashboard` component.

```

"use client"

import { useEffect, useState } from "react"
import { useRouter } from "next/navigation"
import { useAuth } from "@lib/auth-context"
import ScoutDashboard from "@components/ui/scout-dashboard"

✓ export default function ScoutDashboardPage() {
  const { user, isLoading } = useAuth()
  const router = useRouter()
  const [mounted, setMounted] = useState(false)

  // Ensure component is mounted before rendering
  useEffect(() => {
    setMounted(true)
  }, [])

  // Compute if the user has the Scout role
  const userRole = user?.roles?.[0]?.toLowerCase() // take first role

  // Redirect if user is not authorized
  useEffect(() => {
    if (!isLoading && mounted) {

```

Standings

This component displays the league standings for a university sports league, fetching seasons, leagues, and team standings from your API and allowing users to filter by season and league. It shows a loading state while fetching data and renders a dynamic table with team positions, points, wins, losses, draws, form badges, and trend icons, along with a total matches count. Additionally, it includes a legend and abbreviation guide to help users understand the table symbols and highlights champions and playoff positions visually.

```

"use client"

import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card"
import { Badge } from "@components/ui/badge"
import { Button } from "@components/ui/button"
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@components/ui/select"
import { Trophy, TrendingUp, TrendingDown, Minus, ArrowLeft, Loader2 } from "lucide-react"
import Link from "next/link"
import { useState, useEffect } from "react"

// Types based on your existing API response
✓ interface TeamStanding {
  team_id: number
  team_name: string
  position: number
  points: number
  wins: number
  draws: number
  losses: number
  goals_scored: number
  goals_conceded: number
  goal_difference: number
  matches_played: number
  form: string[]
  trend: "up" | "down" | "same"
}

```

Support Page

The SupportPage provides users with contact information, including the NWU Sports Complex address, phone number, and email, making it easy to reach the league support team. It includes sections for general help, frequently asked questions, and feedback, guiding users to resources and encouraging communication. The page also features a back button to navigate to the home page and uses clear visual hierarchy for readability.

```

"use client"

import Link from "next/link"
import { Button } from "@/components/ui/button"
import { ArrowLeft, Mail, Phone, MapPin } from "lucide-react"

export default function SupportPage() {
  return (
    <div className="min-h-screen bg-background">
      { /* Header */ }
      <header className="bg-primary text-primary-foreground py-6">
        <div className="container mx-auto px-4 flex items-center space-x-4">
          <Link href="/home">
            <Button variant="ghost" size="sm" className="text-primary-foreground hover:bg-primary-foreground/20">
              <ArrowLeft className="w-4 h-4 mr-2" />
              Back
            </Button>
          </Link>
          <h1 className="text-2xl font-bold">Support</h1>
        </div>
      </header>
    </div>
  )
}

```

Terms Page

The TermsPage outlines the rules and regulations for using the NWU Sports League website and participating in league activities, covering areas such as eligibility, user conduct, and intellectual property. It informs users about disclaimers, limitations of liability, and how changes to terms will be communicated, ensuring transparency and legal clarity. The page also provides guidance on governing law and contact information for questions, helping users understand their rights and responsibilities.

```

"use client"

import Link from "next/link"
import { Button } from "@components/ui/button"
import { ArrowLeft } from "lucide-react"

export default function TermsPage() {
  return (
    <div className="min-h-screen bg-background">
      { /* Header */ }
      <header className="bg-primary text-primary-foreground py-6">
        <div className="container mx-auto px-4 flex items-center space-x-4">
          <Link href="/home">
            <Button variant="ghost" size="sm" className="text-primary-foreground hover:bg-primary-foreground/20">
              <ArrowLeft className="w-4 h-4 mr-2" />
              Back
            </Button>
          </Link>
          <h1 className="text-2xl font-bold">Terms of Service</h1>
        </div>
      </header>
    </div>
  )
}

```

Rules and Regulations

The RulesPage provides a structured overview of the league's rules and regulations, covering general guidelines, match rules, and disciplinary procedures. It ensures that teams, players, and supporters understand expectations for sportsmanship, eligibility, and adherence to match standards. Additional notes clarify league rights, safety requirements, and equipment standards to promote fair play and accountability.

```
"use client"

import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card"
import { Button } from "@components/ui/button"
import { ArrowLeft } from "lucide-react"
import Link from "next/link"

export default function RulesPage() {
  return (
    <div className="min-h-screen bg-background">
      { /* Header */ }
      <header className="bg-primary text-primary-foreground py-6">
        <div className="container mx-auto px-4 flex items-center space-x-4">
          <Link href="/home">
            <Button
              variant="ghost"
              size="sm"
              className="text-primary-foreground hover:bg-primary-foreground/20"
            >
              <ArrowLeft className="w-4 h-4 mr-2" />
              Back
            </Button>
          </Link>
          <h1 className="text-2xl font-bold">League Rules & Regulations</h1>
        </div>
      </header>
    </div>
  )
}
```

Privacy and policy

The PrivacyPage outlines how the NWU Sports League collects, uses, and protects personal information from users. It details the types of data collected, the purposes for which it is used, and the security measures in place to safeguard it. The page also provides guidance on contacting the league administrators for any privacy-related inquiries.

```
"use client"

import Link from "next/link"
import { Button } from "@components/ui/button"
import { ArrowLeft } from "lucide-react"

export default function PrivacyPage() {
  return (
    <div className="min-h-screen bg-background">
      { /* Header */ }
      <header className="bg-primary text-primary-foreground py-6">
        <div className="container mx-auto px-4 flex items-center space-x-4">
          <Link href="/home">
            <Button variant="ghost" size="sm" className="text-primary-foreground hover:bg-primary-foreground/20">
              <ArrowLeft className="w-4 h-4 mr-2" />
              Back
            </Button>
          </Link>
          <h1 className="text-2xl font-bold">Privacy Policy</h1>
        </div>
      </header>
      <main className="container mx-auto px-4 py-8 space-y-6">
        <section>
          <h2 className="text-xl font-semibold mb-2">Introduction</h2>
          <p>
            NWU Sports League is committed to protecting your privacy. This policy explains how we collect, use, and
          </p>
        </section>
      </main>
    </div>
  )
}
```