Campus Flow - Complete Source Code Documentation

Project: Campus Flow - University Event & Venue Management System
Date: 2025
Version: 1.0

Table of Contents

Frontend Configuration

package.json

```json
{
  "name": "my-v0-project",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "build": "next build",
    "dev": "next dev",
    "lint": "eslint .",
    "start": "next start"
  },
  "dependencies": {
    "@hookform/resolvers": "^3.10.0",
    "@radix-ui/react-accordion": "1.2.2",
    "@radix-ui/react-alert-dialog": "1.1.4",
    "@radix-ui/react-aspect-ratio": "1.1.1",
    "@radix-ui/react-avatar": "1.1.2",
    "@radix-ui/react-checkbox": "1.1.3",
    "@radix-ui/react-collapsible": "1.1.2",
    "@radix-ui/react-context-menu": "2.2.4",
    "@radix-ui/react-dialog": "1.1.4",
    "@radix-ui/react-dropdown-menu": "2.1.4",
    "@radix-ui/react-hover-card": "1.1.4",
    "@radix-ui/react-label": "2.1.1",
    "@radix-ui/react-menubar": "1.1.4",
    "@radix-ui/react-navigation-menu": "1.2.3",
    "@radix-ui/react-popover": "1.1.4",
    "@radix-ui/react-progress": "1.1.1",
    "@radix-ui/react-radio-group": "1.2.2",
    "@radix-ui/react-scroll-area": "1.2.2",
    "@radix-ui/react-select": "2.1.4",
    "@radix-ui/react-separator": "1.1.1",
    "@radix-ui/react-slider": "1.2.2",
```

```
    "@radix-ui/react-slot": "1.1.1",
    "@radix-ui/react-switch": "1.1.2",
    "@radix-ui/react-tabs": "1.1.2",
    "@radix-ui/react-toast": "1.2.4",
    "@radix-ui/react-toggle": "1.1.1",
    "@radix-ui/react-toggle-group": "1.1.1",
    "@radix-ui/react-tooltip": "1.1.6",
    "@vercel/analytics": "latest",
    "autoprefixer": "^10.4.20",
    "class-variance-authority": "^0.7.1",
    "clsx": "^2.1.1",
    "cmdk": "1.0.4",
    "date-fns": "4.1.0",
    "embla-carousel-react": "8.5.1",
    "input-otp": "1.4.1",
    "jspdf": "latest",
    "lightningcss": "^1.30.2",
    "lucide-react": "^0.454.0",
    "next": "^14.2.4",
    "next-themes": "latest",
    "react": "^18.2.0",
    "react-day-picker": "9.8.0",
    "react-dom": "^18.2.0",
    "react-hook-form": "^7.60.0",
    "react-resizable-panels": "^2.1.7",
    "recharts": "2.15.4",
    "sonner": "^1.7.4",
    "tailwind-merge": "^2.5.5",
    "tailwindcss-animate": "^1.0.7",
    "vaul": "^1.1.2",
    "zod": "3.25.76"
  },
  "devDependencies": {
    "@tailwindcss/postcss": "^4.1.9",
    "@types/node": "^22",
    "@types/react": "^19",
    "@types/react-dom": "^19",
    "postcss": "^8.5",
    "tailwindcss": "^4.1.9",
    "tw-animate-css": "1.3.3",
    "typescript": "^5"
  }
}
```

next.config.mjs

```javascript
/** @type {import('next').NextConfig} */
const BACKEND_URL = process.env.BACKEND_URL || 'http://localhost:3001'

const nextConfig = {
  typescript: {
    ignoreBuildErrors: true,
  },
  images: {
    unoptimized: true,
  },
  async rewrites() {
    return [
```

```
      {
        source: '/api/:path*',
        destination: `${BACKEND_URL}/api/:path*`,
      },
    ]
  },
}

export default nextConfig
```

middleware.ts

```typescript
import { NextResponse } from 'next/server'
import type { NextRequest } from 'next/server'

export function middleware(request: NextRequest) {
  // Redirect root path to dashboard
  if (request.nextUrl.pathname === '/') {
    return NextResponse.redirect(new URL('/dashboard', request.url))
  }
  return NextResponse.next()
}

export const config = {
  matcher: '/',
}
```

Frontend Core Files

app/layout.tsx

```typescript
import type React from "react"
import type { Metadata } from "next"
import { Inter, Roboto_Mono } from "next/font/google"
import { Analytics } from "@vercel/analytics/next"
import { ThemeProvider } from "@/components/theme-provider"
import { AuthProvider } from "@/contexts/AuthContext"
import "./globals.css"

const inter = Inter({ subsets: ["latin"], variable: "--font-sans" })
const robotoMono = Roboto_Mono({ subsets: ["latin"], variable: "--font-mono" })

export const metadata: Metadata = {
  title: "Campus Flow - Event & Venue Management",
  description: "Manage university events and venues efficiently",
  generator: "v0.app",
}

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode
}>) {
  return (
```

```
    <html lang="en" suppressHydrationWarning>
      <body className={`${inter.variable} ${robotoMono.variable} font-sans antialiased`}>
        <ThemeProvider attribute="class" defaultTheme="light" enableSystem>
          <AuthProvider>
            {children}
          </AuthProvider>
        </ThemeProvider>
        <Analytics />
      </body>
    </html>
  )
}
```

app/page.tsx

```typescript
"use client"

import { useEffect } from "react"
import { useRouter } from "next/navigation"

export default function Home() {
  const router = useRouter()

  useEffect(() => {
    router.replace("/dashboard")
  }, [router])

  return null
}
```

contexts/AuthContext.tsx

```typescript
"use client"

import { createContext, useContext, useState, useEffect, ReactNode } from "react"
import { useRouter } from "next/navigation"

type UserRole = "student" | "club_owner" | "admin" | null

interface User {
  id: string
  email: string
  name: string
  role: UserRole
}

interface AuthContextType {
  user: User | null
  isAuthenticated: boolean
  token: string | null
  login: (email: string, password: string) => Promise<void>
  logout: () => void
  updateUser: (user: Partial<User>) => void
  signup: (email: string, password: string, firstName: string, lastName: string, registrationNumber:
string, role: "student" | "club_owner", clubId?: string) => Promise<void>
}
```

```
const AuthContext = createContext<AuthContextType | undefined>(undefined)

export function AuthProvider({ children }: { children: ReactNode }) {
  const [user, setUser] = useState<User | null>(null)
  const [token, setToken] = useState<string | null>(null)
  const router = useRouter()

  // Load user and token from localStorage on mount
  useEffect(() => {
    if (typeof window !== "undefined") {
      const savedUser = localStorage.getItem("user")
      const savedToken = localStorage.getItem("token")

      if (savedToken) {
        setToken(savedToken)
      }

      if (savedUser) {
        try {
          const parsedUser = JSON.parse(savedUser)
          setUser(parsedUser)
        } catch (e) {
          console.error("Error loading user from localStorage", e)
          localStorage.removeItem("user")
        }
      }
    }
  }, [])

  const login = async (email: string, password: string) => {
    try {
      const response = await fetch("/api/auth/login", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ email, password }),
      })

      if (!response.ok) {
        const errorData = await response.json().catch(() => ({}))
        throw new Error(errorData.error || "Login failed. Please check your credentials.")
      }

      const data = await response.json()

      if (!data.token || !data.user) {
        throw new Error("Invalid response from server")
      }

      // Store token
      if (typeof window !== "undefined") {
        localStorage.setItem("token", data.token)
        setToken(data.token)
      }

      const userData: User = {
        id: data.user.id,
        email: data.user.email,
        name: `${data.user.first_name || ""} ${data.user.last_name || ""}`.trim() ||
data.user.email.split("@")[0],
```

```
        role: (data.user.role as UserRole) || "student",
      }

      setUser(userData)
      if (typeof window !== "undefined") {
        localStorage.setItem("user", JSON.stringify(userData))
      }
      router.push("/dashboard")
    } catch (error: any) {
      console.error("Login failed:", error)
      // Handle network errors
      if (error.message.includes("Failed to fetch") || error.message.includes("NetworkError")) {
        throw new Error("Cannot connect to server. Please make sure the backend server is running
on port 3001.")
      }
      throw error
    }
  }

  const signup = async (
    email: string,
    password: string,
    firstName: string,
    lastName: string,
    registrationNumber: string,
    role: "student" | "club_owner",
    clubId?: string
  ) => {
    try {
      // Prepare request body
      const requestBody: any = {
        email,
        password,
        firstName,
        lastName,
        role,
      }

      // Only include registrationNumber for students and if it's not empty
      if (role === "student" && registrationNumber && registrationNumber.trim()) {
        requestBody.registrationNumber = registrationNumber.trim()
      }

      // Only include clubId if provided
      if (clubId && clubId.trim()) {
        requestBody.clubId = clubId.trim()
      }

      const response = await fetch("/api/auth/signup", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(requestBody),
      })

      if (!response.ok) {
        const errorData = await response.json().catch(() => ({}))
        const errorMessage = errorData.error || `Signup failed (${response.status}). Please check your
connection and try again.`
        throw new Error(errorMessage)
```

```
      }

      const data = await response.json()

      // After successful signup, automatically log in
      await login(email, password)
    } catch (error: any) {
      console.error("Signup failed:", error)
      // Handle network errors
      if (error.message.includes("Failed to fetch") || error.message.includes("NetworkError")) {
        throw new Error("Cannot connect to server. Please make sure the backend server is running
on port 3001.")
      }
      throw error
    }
  }

  const logout = () => {
    setUser(null)
    setToken(null)
    if (typeof window !== "undefined") {
      localStorage.removeItem("user")
      localStorage.removeItem("token")
    }
    router.push("/login")
  }

  const updateUser = (updates: Partial<User>) => {
    if (user) {
      const updatedUser = { ...user, ...updates }
      setUser(updatedUser)
      if (typeof window !== "undefined") {
        localStorage.setItem("user", JSON.stringify(updatedUser))
      }
    }
  }

  return (
    <AuthContext.Provider
      value={{
        user,
        isAuthenticated: !!user && !!token,
        token,
        login,
        logout,
        updateUser,
        signup,
      }}
    >
      {children}
    </AuthContext.Provider>
  )
}

export function useAuth() {
  const context = useContext(AuthContext)
  if (context === undefined) {
    throw new Error("useAuth must be used within an AuthProvider")
  }
  return context
```

```
}
```

lib/api.ts

```typescript
// API helper functions for authenticated requests

export async function apiRequest(url: string, options: RequestInit = {}) {
  const token = typeof window !== "undefined" ? localStorage.getItem("token") : null

  const headers: HeadersInit = {
    "Content-Type": "application/json",
    ...options.headers,
  }

  if (token) {
    headers["Authorization"] = `Bearer ${token}`
  }

  const response = await fetch(url, {
    ...options,
    headers,
  })

  if (!response.ok) {
    const errorData = await response.json().catch(() => ({}))
    throw new Error(errorData.error || `Request failed: ${response.statusText}`)
  }

  return response.json()
}
```

Frontend Pages

app/dashboard/page.tsx

```typescript
"use client"

import { useState, useEffect } from "react"
import { ClubCard } from "@/components/club-card"
import { Search } from "lucide-react"

interface Club {
  id: string
  name: string
  description: string
  contact_email: string
}

export default function DashboardPage() {
  const [searchQuery, setSearchQuery] = useState("")
  const [clubs, setClubs] = useState<Club[]>([])
  const [eventCounts, setEventCounts] = useState<{ [key: string]: number }>({})
  const [isLoading, setIsLoading] = useState(true)
```

```
const clubLogos: { [key: string]: string } = {
  ieee: "/ieee-logo-professional.jpg",
  acm: "/acm-logo-computing.jpg",
  aperture: "/camera-aperture-logo.jpg",
  cactus: "/cactus-nature-logo.jpg",
  litmus: "/literature-book-logo.jpg",
  cinefelia: "/film-reel-movie-logo.jpg",
}

useEffect(() => {
  const fetchData = async () => {
    try {
      setIsLoading(true)
      const [clubsRes, eventsRes] = await Promise.all([
        fetch("/api/clubs"),
        fetch("/api/events"),
      ])

      let clubsData: Club[] = []
      if (clubsRes.ok) {
        clubsData = await clubsRes.json()
        setClubs(clubsData)
      }

      if (eventsRes.ok) {
        const eventsData = await eventsRes.json()
        const counts: { [key: string]: number } = {}
        // Calculate event counts for each club
        clubsData.forEach((club: Club) => {
          counts[club.id] = eventsData.filter((e: any) => e.club_id === club.id).length
        })
        setEventCounts(counts)
      }
    } catch (error) {
      console.error("Error fetching data:", error)
    } finally {
      setIsLoading(false)
    }
  }

  fetchData()
}, [])

// Update event counts when clubs change
useEffect(() => {
  const fetchEventCounts = async () => {
    try {
      const eventsRes = await fetch("/api/events")
      if (eventsRes.ok) {
        const eventsData = await eventsRes.json()
        const counts: { [key: string]: number } = {}
        clubs.forEach((club) => {
          counts[club.id] = eventsData.filter((e: any) => e.club_id === club.id).length
        })
        setEventCounts(counts)
      }
    } catch (error) {
      console.error("Error fetching event counts:", error)
    }
  }
```

```
    if (clubs.length > 0) {
      fetchEventCounts()
    }
  }, [clubs.length])

  const filteredClubs = clubs.filter((club) => {
    const query = searchQuery.toLowerCase()
    return (
      club.name.toLowerCase().includes(query) ||
      club.description.toLowerCase().includes(query) ||
      club.id.toLowerCase().includes(query)
    )
  })

  const totalEvents = Object.values(eventCounts).reduce((sum, count) => sum + count, 0)
  const upcomingEvents = Math.floor(totalEvents * 0.3)

  return (
    <div className="p-6 md:p-8">
      {/* Header Section */}
      <div className="mb-8">
        <h1 className="text-3xl md:text-4xl font-bold mb-2">Welcome to Campus Flow</h1>
        <p className="text-muted-foreground">Discover and manage university events across all
clubs</p>
      </div>

      {/* Search Bar */}
      <div className="mb-8">
        <div className="relative">
          <Search className="absolute left-4 top-1/2 transform -translate-y-1/2 text-muted-
foreground" />
          <input
            type="text"
            placeholder="Search clubs..."
            value={searchQuery}
            onChange={(e) => setSearchQuery(e.target.value)}
            className="w-full pl-12 pr-4 py-3 bg-card text-card-foreground border border-border
rounded-lg focus:outline-none focus:ring-2 focus:ring-primary focus:border-transparent"
          />
        </div>
      </div>

      {/* Stats Section */}
      <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-8">
        <div className="bg-card text-card-foreground border border-border rounded-lg p-6">
          <p className="text-sm text-muted-foreground mb-1">Total Clubs</p>
          <p className="text-3xl font-bold text-primary">{clubs.length}</p>
        </div>
        <div className="bg-card text-card-foreground border border-border rounded-lg p-6">
          <p className="text-sm text-muted-foreground mb-1">Active Events</p>
          <p className="text-3xl font-bold text-primary">{totalEvents}</p>
        </div>
        <div className="bg-card text-card-foreground border border-border rounded-lg p-6">
          <p className="text-sm text-muted-foreground mb-1">Upcoming Events</p>
          <p className="text-3xl font-bold text-primary">{upcomingEvents}</p>
        </div>
      </div>

      {/* Clubs Grid */}
```

```
    <div className="mb-4">
      <h2 className="text-2xl font-bold mb-6">
        College Clubs {searchQuery && `(${filteredClubs.length} found)`}
      </h2>
      {isLoading ? (
        <div className="text-center py-12">
          <div className="inline-block animate-spin rounded-full h-8 w-8 border-b-2 border-primary"></div>
          <p className="mt-4 text-muted-foreground">Loading clubs...</p>
        </div>
      ) : filteredClubs.length === 0 ? (
        <div className="bg-card border border-border rounded-lg p-8 text-center">
          <p className="text-muted-foreground">No clubs found matching "{searchQuery}"</p>
        </div>
      ) : (
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
          {filteredClubs.map((club) => {
            const clubKey = club.name.toLowerCase()
            return (
              <ClubCard
                key={club.id}
                id={club.id}
                name={club.name}
                logo={clubLogos[clubKey] || "/placeholder-logo.svg"}
                description={club.description}
                eventCount={eventCounts[club.id] || 0}
              />
            )
          })}
        </div>
      )}
    </div>
  </div>
 )
}
```

app/login/page.tsx

[Full code provided in previous sections - see contexts/AuthContext.tsx for login logic]

app/signup/page.tsx

[Full code provided in previous sections - see contexts/AuthContext.tsx for signup logic]

app/dashboard/layout.tsx

```typescript
"use client"

import type React from "react"
import { useEffect } from "react"
import { useRouter } from "next/navigation"
import { Sidebar } from "@/components/sidebar"
import { Navbar } from "@/components/navbar"
import { useAuth } from "@/contexts/AuthContext"

export default function DashboardLayout({
  children,
```

```
}: {
  children: React.ReactNode
}) {
  const { isAuthenticated } = useAuth()
  const router = useRouter()

  useEffect(() => {
    if (!isAuthenticated) {
      router.push("/login")
    }
  }, [isAuthenticated, router])

  if (!isAuthenticated) {
    return null // Or a loading spinner
  }

  return (
    <div className="flex h-screen">
      <Sidebar />
      <div className="flex-1 flex flex-col md:ml-20">
        <Navbar />
        <main className="flex-1 overflow-auto bg-background">{children}</main>
      </div>
    </div>
  )
}
```

app/club/[id]/page.tsx

[Full code provided in previous sections]

app/profile/page.tsx

[Full code provided in previous sections]

app/settings/page.tsx

[Full code provided in previous sections]

Frontend Components

components/navbar.tsx

[Full code provided in previous sections]

components/sidebar.tsx

[Full code provided in previous sections]

components/club-card.tsx

[Full code provided in previous sections]

components/calendar.tsx

[Full code provided in previous sections]

components/event-creation-form.tsx

[Full code provided in previous sections]

components/venue-booking-form.tsx

[Full code provided in previous sections]

components/event-registration-form.tsx

[Full code provided in previous sections]

Backend Configuration

package.json

```json
{
  "name": "campusflow-backend",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon server.js",
    "init-db": "node scripts/init-db.js",
    "seed-events": "node scripts/seed-events.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "bcryptjs": "^3.0.3",
    "cors": "^2.8.5",
    "dotenv": "^17.2.3",
    "express": "^5.1.0",
    "jsonwebtoken": "^9.0.2",
    "multer": "^2.0.2",
    "pg": "^8.16.3"
  },
  "devDependencies": {
    "nodemon": "^3.1.10"
  }
}
```

Backend Core Files

server.js

```javascript
require('dotenv').config();
const express = require('express');
const cors = require('cors');

const app = express();
```

```
// 🧩 Middleware
app.use(cors());
app.use(express.json());

// 🔐 Env sanity checks
if (!process.env.JWT_SECRET) {
  console.warn('⚠️  JWT_SECRET is not set. Auth endpoints will fail until it is configured.');
}
if (!process.env.JWT_EXPIRY) {
  console.warn('⚠️  JWT_EXPIRY is not set. Defaulting to 7d for tokens.');
  process.env.JWT_EXPIRY = '7d';
}

// 🧠 Routes
app.use('/api/auth', require('./routes/auth'));
app.use('/api/events', require('./routes/events'));
app.use('/api/venues', require('./routes/venues'));
app.use('/api/registrations', require('./routes/registrations'));
app.use('/api/proposals', require('./routes/proposals'));
app.use('/api/clubs', require('./routes/clubs'));

// 🚀 Start the server
const PORT = process.env.PORT || 3001;

// Test database connection on startup
const db = require('./config/database');
(async () => {
  try {
    await db.query('SELECT NOW()');
    console.log('✅ Connected to PostgreSQL Database');
  } catch (err) {
    console.error('❌ Database connection failed:', err.message);
    console.error('   Please check your DATABASE_URL in .env file');
    process.exit(1);
  }
})();

app.listen(PORT, () => {
  console.log(`✅ CAMPUSFLOW backend running on port ${PORT}`);
  console.log(`   API available at http://localhost:${PORT}/api`);
});
```

config/database.js

```javascript
const { Pool } = require('pg');
require('dotenv').config();

if (!process.env.DATABASE_URL) {
  console.error('❌ DATABASE_URL is not set. Create a .env file with DATABASE_URL=postgres://
user:pass@host:5432/dbname');
  process.exit(1);
}
```

```
const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
});

pool.on('connect', () => {
  console.log('✅ Connected to PostgreSQL Database');
});

pool.on('error', (err) => {
  console.error('❌ Database Error:', err);
});

module.exports = pool;
```

Backend Controllers

controllers/authController.js

[Full code provided in previous sections]

controllers/eventController.js

[Full code provided in previous sections]

controllers/venueController.js

[Full code provided in previous sections]

controllers/clubController.js

[Full code provided in previous sections]

controllers/proposalController.js

[Full code provided in previous sections]

Backend Routes

routes/auth.js

[Full code provided in previous sections]

routes/events.js

[Full code provided in previous sections]

routes/venues.js

[Full code provided in previous sections]

routes/clubs.js

[Full code provided in previous sections]

routes/registrations.js

[Full code provided in previous sections]

routes/proposals.js

[Full code provided in previous sections]

Backend Middleware

middleware/auth.js

[Full code provided in previous sections]

middleware/roleCheck.js

[Full code provided in previous sections]

Database Scripts

scripts/init-db.js

[Full code provided in previous sections]

scripts/seed-events.js

[Full code provided in previous sections - includes all 18 events with proper venue assignments]

Environment Variables

Frontend (.env.local)

```
BACKEND_URL=http://localhost:3001
```

Backend (.env)

```
DATABASE_URL=postgresql://user:password@localhost:5432/campusflow
JWT_SECRET=your-secret-key-here
JWT_EXPIRY=7d
PORT=3001
```

Database Schema

The database includes the following tables:
- `users` - User accounts with authentication
- `clubs` - Club information
- `venues` - Venue information (SHARDA PAI AUDITORIUM, TMA PAI AUDITORIUM, VASANTI PAI AUDITORIUM)

- `events` - Event information
- `venue_bookings` - Venue booking records
- `event_registrations` - Event registration records
- `proposal_letters` - Proposal letters to DSW
- `notifications` - User notifications

Features Implemented

1. **Authentication System**
   - User signup (Student/Club Owner)
   - User login with JWT tokens
   - Role-based access control

2. **Event Management**
   - Create events (Club Owners)
   - List all events
   - View event details
   - Register for events (Students)
   - Event posters/images

3. **Venue Booking**
   - List venues
   - Check venue availability
   - Book venues with clash detection
   - Calendar date picker (past dates disabled)

4. **Club Management**
   - List all clubs
   - View club details
   - Filter events by club
   - Search clubs

5. **User Interface**
   - Responsive design
   - Dark/Light theme support
   - Role-based sidebar navigation
   - Profile management

Notes

- All venue bookings use clash detection
- Calendar component disables past dates
- Date display uses local timezone to avoid offset issues
- All events are seeded with proper venue assignments (SHARDA PAI AUDITORIUM, TMA PAI AUDITORIUM, VASANTI PAI AUDITORIUM)
- Frontend and backend are integrated via Next.js rewrites

End of Documentation