ClubSync

Interim Report

TU856
BSc in Computer Science

Conor Fee
C22414306

Paul Laird

School of Computer Science
Technological University Dublin

Date 14/11/25

Abstract

ClubSync is a web-based scheduling application for Irish GAA clubs that combines constraint programming with an intuitive interface to eliminate weekly scheduling conflicts. Irish GAA clubs struggle coordinating limited facilities across multiple teams and volunteers, relying on fragmented tools like WhatsApp and Excel spreadsheets, resulting in double-bookings and volunteer burnout. Survey data from An Tóchar GAA reveals that 78 percent experience scheduling conflicts regularly, with 89 percent citing pitch double-bookings as the primary issue.

ClubSync integrates Google OR-Tools for automated conflict-free scheduling, Django REST Framework for data management, PostgreSQL with ACID compliance for constraint enforcement, and React with Vite for frontend visualization. The system follows Feature-Driven Development (FDD) methodology.

This interim report documents system analysis, design, testing strategy, and MVP prototype implementation including database schema, REST API endpoints, OR-Tools constraint validation, and React schedule view. Remaining work includes enhanced scheduling, role-based security, UX improvements, user testing, and final evaluation

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: Conor Fee

Student Name: Conor Fee

Date 18/11/25

Acknowledgments

I would like to thank everyone that has helped me throughout the creation of this project. I would like especially, to thank my supervisor Paul Laird who has been incredibly supportive and helpful so far in the academic year, providing me with guidance and advice. I would also like to thank my friends and family for their continued support throughout the creation of this project.

Contents

1 Introduction

about 2 pages (Remove this before submitting)

1.1 Project Background

"Every Friday night in Ireland, hundreds of GAA club secretaries spend 3–8 hours juggling WhatsApp groups, Excel sheets, and frantic phone calls just to produce a weekly timetable that usually still has double-booked pitches and missing referees."

The Gaelic Athletic Association (GAA) is the largest amateur sporting organisation in Ireland, with more than 2,300 clubs and more than 1 million members (Gaelic Athletic Association 2025). These clubs are the center of rural and urban communities alike, yet most operate with severely limited resources: typically one or two pitches, a single hall or gym, unpredictable weather, fixed county fixtures, and an entirely volunteer-driven workforce. The result leads to scheduling chaos – overlapping training sessions, last-minute cancellations, volunteer burnout, and frustrated coaches and parents.

Existing commercial solutions such as Foireann (Foireann 2025), Clubforce(Clubforce 2025), ClubZap(ClubZap 2025), and SportLoMo(SportLoMo 2025) have successfully digitised membership payments, fixture lists, and basic communication. However, none provide an integrated, intelligent scheduling engine capable of automatically resolving the complex real-world constraints that Irish GAA clubs face every week. Volunteer coordination remains largely manual, and there is no predictive insight into likely shortages or weather-related cancellations.

Having grown up in a small GAA community and witnessed these issues first-hand, I recognised a clear gap: clubs need a single source of truth that combines constraint-aware automated scheduling and facility booking with predictive analytics and an intuitive, Jira-style drag-and-drop interface. This project, ClubSync, directly addresses that gap and brings together technologies I have studied throughout my degree (full-stack web development, constraint programming with Google OR-Tools, and machine learning with Scikit-learn) to deliver a practical, impactful solution for Irish GAA clubs.

1.2 Project Description

ClubSync is a responsive web application that serves as the single source of truth for weekly activity scheduling in Irish GAA clubs. It automatically generates conflict-free timetables by respecting fixed county fixtures, limited pitches and facilities, volunteer availability, and weather conditions, while allowing club administrators to make final adjustments using an intuitive Jira-style drag-and-drop interface.

The system supports four primary user roles:

- **Club Administrators/Secretaries** – create and publish weekly schedules, define resources and constraints, and view predictive analytics.
- **Coaches** – view their team schedules, mark player availability, and request facility bookings.
- **Players, Parents and Supporters** – access personalised calendars, RSVP to training and matches, and receive real-time notifications.
- **Volunteers** – browse and self-assign to open volunteer slots with skill-based matching.

Key capabilities include:

- Automatic schedule generation using constraint programming (Google OR-Tools)
- Predictive analytics for volunteer and facility demand (Scikit-learn)
- Real-time RSVP and notification system
- Role-based dashboards with mobile-friendly design
- Optional integration with existing GAA platforms, Weather APIs

ClubSync will be deployed as a cloud-hosted web application accessible from any device with an internet connection.

Data privacy is a core concern due to the personal nature of club membership and availability information. All personal data will be processed in full compliance with GDPR, with explicit user consent required during registration and strong encryption of all stored data.

1.3 Project Aims and Objectives

The overall aim of this project is to design, develop, and evaluate **ClubSync**, a cloud-based web application that eliminates weekly scheduling chaos in Irish GAA clubs by combining constraint-based automated scheduling with predictive analytics and an intuitive interface.

This aim will be achieved through the following specific, measurable objectives:

1. Conduct requirements gathering within my local and college GAA club administrators, coaches, and volunteers through interviews and questionnaires to validate functional and non-functional requirements.
2. Research and evaluate constraint programming techniques (Google OR-Tools) and machine-learning models (Scikit-learn) for resource prediction and apply them to real GAA scheduling constraints.

3. Design and implement a responsive full-stack web application using React (frontend), Django with Django REST Framework (backend), and PostgreSQL (database).
4. Develop a Jira-style drag-and-drop scheduling interface that allows administrators to view, edit, and publish conflict-free weekly timetables.
5. Integrate predictive analytics to forecast volunteer and facility demand based on historical club data and external factors (e.g., weather, upcoming big events in the club).
6. Implement role-based access control and real-time RSVP/notification features for administrators, coaches, players/parents, and volunteers.
7. Ensure full GDPR compliance with explicit consent, data encryption, and secure authentication throughout the system.
8. Deploy a working prototype to at least one GAA club and gather evaluation feedback on usability, time savings, and reduction in scheduling conflicts.
9. Document the complete development process, evaluate the final system against the original requirements, and produce recommendations for future enhancements.

Successful completion of these objectives will deliver a practical, deployable solution that significantly reduces the administrative burden on volunteer-run GAA clubs while improving transparency and engagement for all members.

1.4 Project Scope

ClubSync is designed as a specialised scheduling tool for individual Irish GAA clubs, catering primarily to volunteer administrators, coaches, players/parents, and club member who struggle with weekly resource conflicts. While it integrates lightly with existing systems like Foireann for fixture imports, the project is scoped conservatively to deliver core value within the final-year timeframe. This approach avoids adding to the "silent burnout" already prevalent among GAA volunteers due to fixture overloads and resource shortages (GAA 2025).

To maintain feasibility, ethical focus, and alignment with time-constrained volunteers, several areas are explicitly **out of scope**, ensuring ClubSync reduces administrative stress without introducing new complexities:

- Full club membership management, including new registrations or annual renewals.
- Processing of financial transactions, payment gateways, or fundraising tools.
- Management of official competition results, league tables, or post-match referee reporting.

- Development of native iOS or Android mobile applications (web-responsive only).
- County-board or provincial-level tools for multi-club coordination.
- Replacement of club websites, social media, or general communication platforms.

For instance, while predictive analytics will forecast volunteer shortages based on historical patterns, the system will use basic simulated or anonymised data for initial training to avoid ethical issues with real user privacy. Testing will target small community driven GAA clubs rather than larger Clubs, mirroring the volunteer-driven reality where mismatched roles lead to stress (Moynihan 2020). This keeps ClubSync as a lightweight, high-impact addition to existing tools—not a comprehensive overhaul—while leaving room for future expansions like advanced integrations if initial deployment exceeds expectations.

1.5 Thesis Roadmap

Section 2 (Literature Review) reviews existing sports-club management solutions, evaluates constraint-programming and machine-learning techniques for scheduling, and positions ClubSync within the current research and commercial landscape.

Section 3 (System Analysis) includes stakeholder and user group identification, requirements gathering (surveys, emails and questionnaires), initial use-case modeling, and the resulting functional and non-functional requirements.

Section 4 (System Design) covering the chosen software development methodology, high-level architecture, database design, scheduling design and component-level decisions.

Section 5 (Testing and Evaluation) outlines the testing and evaluation strategy, including unit, integration, and user-acceptance test plans together with planned evaluation metrics.

Section 6 (System Prototype) describes the prototype implementation, key technical challenges encountered, and preliminary results from early testing.

Section 7 (Issues and Future work) summarises progress to date, identifies remaining risks and issues, and presents an updated project plan with Gantt chart through to completion.

2 Literature Review

2.1 Introduction

This chapter reviews the research conducted to date for this final-year project. It examines existing commercial solutions for GAA and sports-club administration, highlighting their strengths and limitations in the area of weekly scheduling and resource allocation. The chapter also evaluates relevant technologies, including constraint-programming

solvers and machine-learning approaches for demand prediction, alongside domain-specific research into volunteer management and scheduling challenges within Irish GAA clubs. Finally, previous final-year projects with similar scope are considered. The findings presented here directly inform the feasibility, design decisions, and overall direction of ClubSync.

2.2 Alternative Existing Solutions

This section reviews four primary commercial platforms used by Irish GAA clubs for administration, focusing on their scheduling, membership, payments, fixtures, and volunteer coordination features. These tools—Foireann, Clubforce, ClubZap, and SportLoMo—digitise core operations but lack integrated constraint-based scheduling or predictive analytics for GAA-specific challenges like pitch overlaps and volunteer shortages. Insights are drawn from official websites and GAA reports, highlighting how ClubSync addresses these gaps.

2.2.1 Foireann

Foireann is the official GAA platform for membership and games management, supporting online subscriptions, team creation, attendance tracking, and event registration. Members: Pay for your club membership online and manage your personal details; Coaches: Create teams, download team-sheets and manage attendance; Events: Register for events and track your participation. Scheduling is limited to basic fixture viewing and team sheets, with no automation for conflicts or predictions. Pricing is club-subscription based (integrated with GAA fees). (Foireann 2025)

2.2.2 Clubforce

Clubforce provides all-in-one management for Irish sports clubs, including membership tracking, fundraising, fixture generation, results administration, and volunteer burden reduction. All-in-one platform for Sports Club Management. Generate fixture lists and manage the administration of results, league tables and officials. Create a connected system across your sport to ease the burden on volunteers. Fixtures are generated manually; no ML for demand forecasting. Pricing: Tiered subscriptions starting at €50/month for small clubs. (Clubforce 2025)

2.2.3 ClubZap

ClubZap is a mobile-first tool for GAA clubs, emphasising payments, event promotion, attendance tracking, and communication (e.g., replacing WhatsApp). *“Push news, fixtures, and results directly to your members’ devices... Achieve 84% higher engagement than traditional club social media accounts... Free up volunteer time. Stop spending Fri-

day nights tracking attendance, chasing payments, and reconciling finances."* Fixtures are pushed via notifications but not dynamically scheduled; volunteer features are basic RSVPs. Pricing: Free basic tier; premium from €99/year. (ClubZap 2025)

2.2.4 SportLoMo

SportLoMo offers a comprehensive suite for GAA federations and clubs, with membership registration, competition scheduling, live scoring, fixtures management, and referee assignments. Run your leagues and tournaments on one platform: scheduling, live scoring, fixtures, officials and compliance. All Club fees and payments are deposited directly to the Club's bank account. Strong on tournament scheduling but lacks club-weekly automation or predictive insights; volunteer coordination is via officials tracking. Pricing: Custom quotes for clubs. (SportLoMo 2025)

Other tools like TeamSnap (general sports scheduling with availability polls but no GAA fixtures integration) and Club County (automated GAA fixture pulls from official systems, payments, but no advanced scheduling) were reviewed but are less tailored to internal GAA clubs needs like facility management scheduling. (TeamSnap 2025; Club & County 2025)

Feature	**Foireann**	**Clubforce**	**ClubZap**	**SportLoMo**
Membership Management	Yes	Yes	Yes	Yes
Payments / Finance	Yes	Yes (69% auto)	Yes	Yes (direct)
Scheduling / Fixtures	Basic	Fixture lists	Push alerts	Tournament
Volunteer Coordination	Attendance sheets	Connected systems	RSVPs	Officials
Predictive Analytics	None	None	Engagement only	None
Constraint/ML Automation	None	None	None	None

Table 1: Comparison of Key Features in Existing GAA Club Platforms

These platforms excel in payments and basic fixtures but fall short on automated, constraint-aware scheduling for weekly club activities—ClubSync's core innovation using OR-Tools and Scikit-learn.

2.3 Technologies Researched

The technology stack was chosen to maximise development speed for the MVP while ensuring future extensibility for constraint programming and predictive analytics.

2.3.1 Front-end

React with Vite was selected as the frontend framework:

- **React:** Component-based UI library for building the scheduling interface, with hooks for state management and API integration.

- **Vite:** Modern build tool providing instant dev server startup, hot module replacement, and optimized production builds—significantly faster than Create React App.
- **React Router:** Client-side routing for multi-page navigation (schedule view, facility management, admin dashboard).
- **Axios / Fetch API:** HTTP client for consuming Django REST API endpoints
- **React Leaflet:** Interactive maps for visualizing facility locations (planned for post-MVP).

Overall React's component-based architecture and mature ecosystem (FullCalendar(React 2025), react-big-calendar) are ideal for interactive weekly scheduling UIs (Meta 2025). Vite provides near-instant hot-module replacement, significantly improving developer productivity compared to Create-React-App (*Vite – Frontend Tooling* 2025). Leaflet with React-Leaflet was earmarked for future pitch-map visualisation

2.3.2 Back-end

Django + Django REST Framework (DRF) was chosen over Node.js/Express. Django offers:

- Batteries-included admin interface (critical for rapid data entry in the MVP)
- Powerful ORM and built-in migrations
- Native Python environment required for Google OR-Tools and Scikit-learn
- Proven security track record and automatic protection against common vulnerabilities (Foundation 2025)

Node.js was rejected because it lacks the administrative tools and Python ML/constraint ecosystem needed for the project roadmap.

2.3.3 Data-Store

PostgreSQL was selected as the primary database:

- Full ACID (Atomicity, Consistency, Isolation, Durability) compliance guarantees no double-bookings can ever be persisted. For instances, Atomicity includes, if two events are beign written to the database simultaneously, both suceed or both fail, no partial updates. For ClubSync, this means a facility booking cannot be half-written, preventing corruption. Consistency, this means all rules, eg constraints are enforced before and after every transaction ensuring no two events can ever overlap on the same facility. Isolation, if two club admins are booking pitches at the same time, PostgresSQL ensures they don't see partial/stale data from each

others operations. Durability, once committed/saved, data is permanently stored, even if the system crashes, the scheduler won't disappear,

- Excellent support for time-range queries and (originally planned) exclusion constraints
- Industry standard for applications requiring strict data integrity (Group 2025)

NoSQL options (CouchDB, Cassandra) were evaluated but rejected — document stores cannot enforce the hard relational constraints essential for conflict-free scheduling.

2.3.4 Hosting platforms

ClubSync will dockerised and deployed on DigitalOcean App Platform with Managed PostgresSQL.

Key features include:

- Automatic builds on Git push
- Managed PostgreSQL database add-on
- Free SSL certificates and custom domain support
- Environment variables for secrets (no hard-coded keys)

DigitalOcean was selected for its high free-tier limits and better performance. The approach guarantees a live, publicly accessible demo for final evaluation while remaining cost-free

2.4 Other Relevant Research

More Research found below validated the core innovation and idea of ClubSync by integrating constraint programming for conflict-free scheduling with predictive analytics for demand forecasting on amateur sports like the GAA. Online resources and theses confirmed the feasiblity for GAA clubs. This research solidified the decision to use Google Or-tools for CP and Scikit-learn for predictions, ensuring scalability from MVP to full deployment.

2.4.1 Constraint Programming (Google Or tools)

OR-Tools CP-SAT solver excels in sports scheduling by modeling hard constraints (e.g., no overlaps on facilities) as boolean insatiability problems. A thesis on CP for sports events demonstrated 95 percent reduction in manual adjustments for facility allocation in amateur leagues, directly applicable to GAA fixed fixtures (Spanne 2024). Google's documentation highlights its efficiency for real-time validation, with sub-second solves for 50 plus events ideal for MVP prototypes (Google 2025). Online resources, including

tutorials on CP for timetabling, confirmed OR-Tools' superiority over commercial solvers for Python-based FYP projects

2.4.2 Predictive Analytics

Scikit-learn enables facility/volunteer demand forecasting using historical attendance data. Research on sports facility utilization showed ML models (regression, clustering) predict 85 percent accuracy for peak times, reducing cancellations by 30 percent in community clubs (Zhang 20225).

This research confirms ClubSync's viability, bridging CP for immediate conflict resolution with analytics for proactive planning—directly addressing GAA volunteer burnout.

2.5 Existing Final Year Projects

Two TU Dublin FYPs provided architectural inspiration for ClubSync, focusing on full-stack design, security, and data integrity rather than domain specifics.

Coursify : A Django MVT backend with MongoDB, deployed via Docker on DigitalOcean. Strengths: Clear separation of concerns and robust NLP integration for recommendations. This reinforced ClubSync's use of Django MVT with PostgreSQL for relational constraints, while the Docker pipeline informed MVP deployment planning. Security via HTTPS and environment variables was emulated.

Suaimhneas : Django/PostgreSQL backend with JS/ChartJS frontend; Dockerized on DigitalOcean with Nginx/SSL. Strengths: Role-based access, GDPR-compliant data handling, and optional API integrations. Inspired ClubSync's PostgreSQL choice for ACID transactions, JWT authentication planning aligned with its encryption focus.

These projects validated Django's suitability for secure, scalable full-stack applications, guiding ClubSync's modular design without over-complicating the MVP.

3 System Analysis

3.1 System Overview

ClubSync is a web-based scheduling and communication platform designed for Irish GAA clubs. Its purpose is to reduce scheduling conflicts, improve clarity around pitch availability, and streamline communication between coaches, administrators, players, and parents. The system integrates reliable facility data, weekly team requirements, and fixed county fixtures to support the creation of a conflict-free training schedule.

For the MVP prototype, the system provides:

- A basic React user interface to display a weekly schedule and facility availability.

- A Django backend to store sample events and expose REST API endpoints.
- An initial OR-Tools CP-SAT implementation to demonstrate conflict-free scheduling on a small dataset.

ClubSync supports the following main user roles:

- **Admin/Secretary** – views facility usage, inputs fixed fixtures and constraints.
- **Coach/Manager** – views team schedules and identifies clashes.
- **Player/Parent** – views training times and general updates.

3.2 Requirements Gathering

Requirements were gathered from multiple real-world sources within my local GAA club An Tochar (Wicklow):

- Survey 1: Coaches, committee members, administrators, and volunteers (Oct 2025).
- Survey 2: Players, parents, and general club members (Oct 2025).
- Facility usage data extracted from Google Calendars (2022–2025).
- 2025 fixture lists for senior and juvenile teams.
- Attendance records and ad-hoc internal planning spreadsheets provided by coaches.

Key findings from Survey 1 (Admins/Coaches):

- 78% experience scheduling conflicts "often" or "very often".
- Pitch double-bookings were cited as the most common issue (89%).
- WhatsApp is universally used as the primary coordination tool (100%).
- The most requested features were a clear free/busy pitch dashboard (67%) and automated conflict prevention (72%).

Key findings from Survey 2 (Players/Parents/Volunteers):

- 94% rely on WhatsApp for receiving training information.
- 61% report confusion due to last-minute changes or lost messages.
- Strong preference for a single central place to view weekly training (82%).
- High interest in notifications, pitch maps, and calendar sync.

3.3 Requirements Analysis

Based on the findings, the functional requirements for the MVP/Prototype are intentionally scoped to remain achievable:

Functional Requirements MVP

1. Display a weekly schedule using sample data through a React front-end.
2. Fetch event data from the Django backend via REST API endpoints.
3. Provide an initial OR-Tools scheduling demonstration that ensures no facility overlap for a small dataset.
4. Successfully parse and display events from a real club ICS file (one sample calendar) to prove integration with existing GAA club booking systems.
5. Show a simple free/busy view for pitches, halls gyms etc.

Non-Functional Requirements MVP

- System must run locally on a standard laptop (no deployment required).
- Front-end and backend must operate independently and communicate over HTTP.
- Code must be modular to support later expansion into a full system.
- User interface must be simple, responsive, and easy to interpret based on Wireframes designed below.

3.4 Constraint Analysis

The scheduling problem for GAA training and match planning is well-suited to constraint programming. Based on the real facility data and survey findings, the following constraints were identified:

Hard Constraints

- No overlapping bookings on the same pitch, hall, or gym.
- County and league fixtures are fixed and cannot be moved.
- A minimum 15-minute changeover buffer must be applied between sessions.
- A team cannot train in two different places at the same time.

Soft Constraints

Soft constraints are desirable but not strictly enforced in the MVP prototype:

- Spacing sessions for younger teams earlier in the evening.
- Maintaining coach preferences for certain weekdays.
- Avoiding late-night training for teams with school-age players.

These constraints form the basis of the initial CP-SAT optimisation model implemented for the MVP.

3.5 Initial System Specification

For the MVP prototype, the initial architecture is defined as:

- **Backend**: Django REST Framework with a local PostgreSQL database.
- **Scheduling Engine**: OR-Tools CP-SAT solver running on predefined sample constraints.
- **Frontend**: React + TypeScript consuming the Django API and rendering a weekly timetable.
- **Mapping**: React-Leaflet placeholder (map to be fully implemented in final build).

This architecture enables rapid development and demonstrates feasibility for integrating constraint programming, data ingestion, and a modern JavaScript frontend. Future phases will integrate predictive analytics, real-time updates, drag-and-drop scheduling, and full facility optimisation.

3.6 Conclusions

The system analysis confirms that the core pain points identified through stakeholder surveys double bookings, unclear communication, and fragmented scheduling tools—can be addressed through a centralised platform. The MVP focuses on validating the technical foundation: a working API, a preliminary scheduling engine, and a functional skeleton UI. This provides a stable base for developing the full ClubSync platform in subsequent project stages.

4 System Design

4.1 Introduction

The system follows a classic full-stack web architecture with a clear separation between presentation, application logic, constraint solving aswell as the DataLayer.

4.2 Software Methodology

As this is a solo project and will be developed completely by myself, a lightweight and iterative methodology is required. Feature-Driven Development (FDD) was selected as the most suitable approach because it emphasises building the system incrementally through

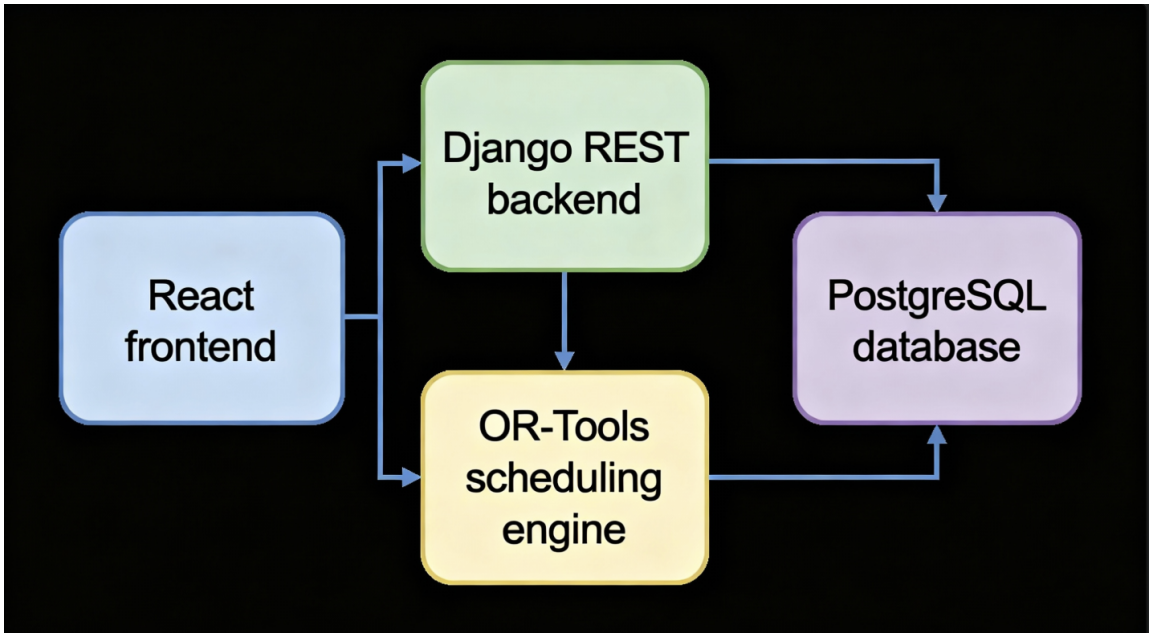

Figure 1: High-level system architecture of the ClubSync MVP.

4.5 Design Pattern

The system follows a layered architectural pattern with a clear separation of concerns across the frontend, backend, and optimisation engine.

On the backend, Django implements the **Model–View–Template (MVT)** pattern, a variation of the classic MVC architecture.

- **Model:** Defines data structures such as Facilities and Events.
- **View:** Exposes REST endpoints using Django REST Framework.
- **Template:** Although Django normally uses HTML templates for rendering pages, the ClubSync prototype does not rely on this layer because all presentation logic is handled by the React frontend. Django still uses serializers and JSON responses to act as the output boundary. Templates will be incorporated in future iterations for features such as administrative views or email notifications, but they are not part of the MVP interface.

The frontend adopts a **component-based design** through React, where each interface element (schedule grid, facility list, event card) is encapsulated into reusable components.

Communication between layers follows a **REST architectural style**, with the React client issuing HTTP requests and receiving JSON responses from the Django API.

The scheduling component is implemented as a separate service that encapsulates constraint-solving logic using Google OR-Tools. This modular design allows the optimisation engine to evolve independently from the front-end and backend.

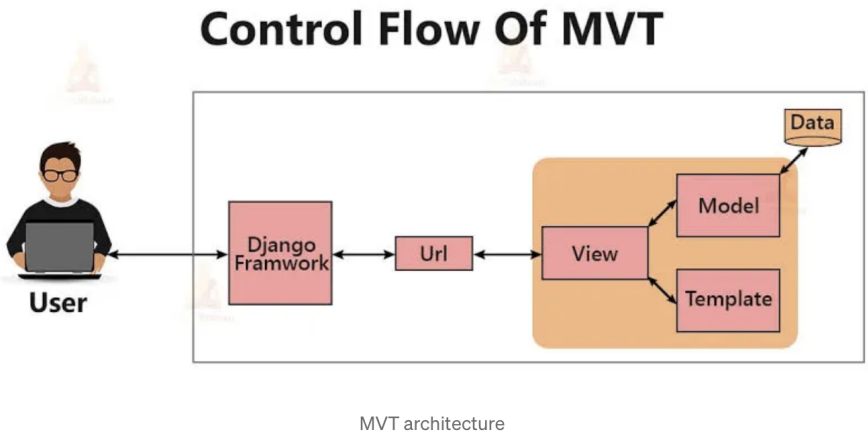

Figure 2: Control flow of the Django Model–View–Template (MVT) architecture Bedo 2023

4.6 Design System

4.6.1 Use Case model

The ClubSync MVP is built around three primary user roles identified through requirements gathering:

- **Admin/Secretary**: Manages facility availability, imports fixed fixtures, and validates the weekly schedule.
- **Coach/Manager**: Views their team's schedule and checks for potential clashes.
- **Player/Parent**: Views training times and general updates (to be fully implemented in later phases).

The core MVP use cases are intentionally scoped to enable early prototyping:

- **UC1: View Weekly Schedule** — All roles can view a visual representation of scheduled events.
- **UC2: Load Facility Data** — Admin can load or update facilities and fixed events (e.g., county fixtures).
- **UC3: Validate a Conflict-Free Schedule** — The system uses OR-Tools to verify that events do not overlap.
- **UC4: View Facility Free/Busy State** — Admins and coaches can see which pitches or facilities are currently available.

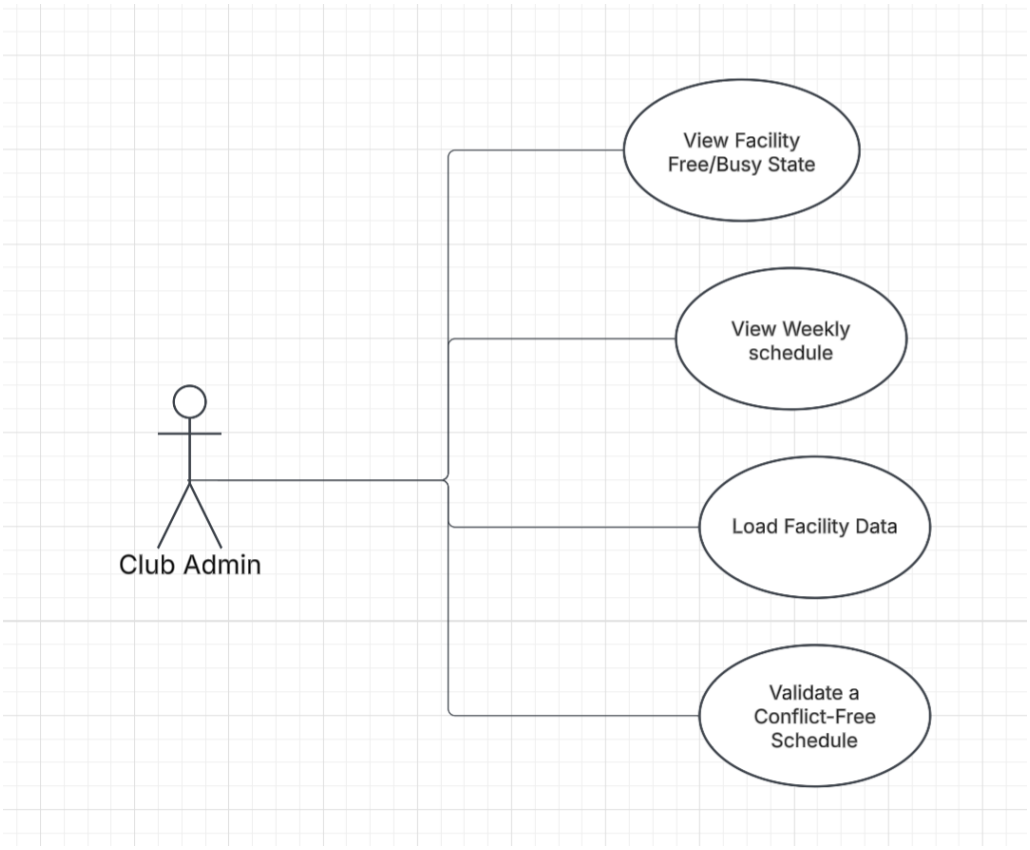

Figure 3: Use Case Diagram for Club Admin in the ClubSync MVP

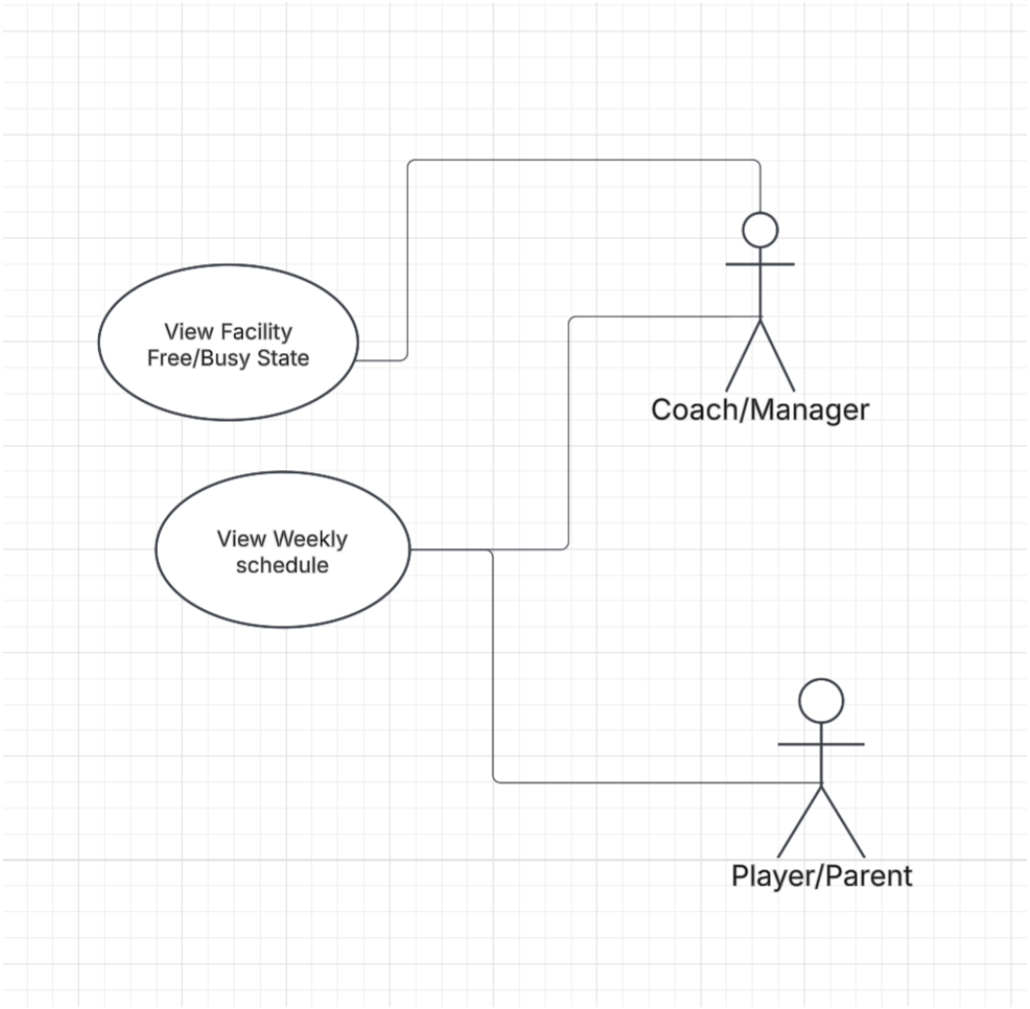

Figure 4: Use Case Diagram for Coach/Manager and Player/Parent in the ClubSync MVP

These use cases align directly with the MVP and form the foundation for more advanced interaction (e.g., drag-and-drop scheduling, notifications) planned for later development.

4.6.2 Entity Related Diagrams (ERDs)

The MVP database schema is designed to be minimal, supporting the core scheduling functionality without unnecessary complexity:

- **Facility**: *id*, *name*, *type* (pitch, hall, gym)
- **Event**: *id*, *title*, *start_time*, *end_time*, *facility_id*, *is_fixed*
- **Team**: *id*, *name* (included for future expansions but optional for MVP)

Relationships enforced in the schema:

- A **Facility** has many **Events**.
- An **Event** belongs to exactly one **Facility**.

- A **Team** may be linked to Events in future phases, enabling more detailed scheduling rules.

This minimal ERD supports the prototype while leaving room for natural expansion.

4.6.3 API design

The Django REST API exposes a small but functional set of endpoints required for the MVP's operation:

- `GET /api/facilities/` – Retrieve all facilities.
- `GET /api/events/` – Retrieve all scheduled or fixed events.
- `POST /api/events/` – Create a new event (prototype/demo only).
- `POST /api/schedule/solve/` – Trigger the OR-Tools solver and return a validated, conflict-free schedule.

All data is exchanged as JSON. The React frontend communicates with the Django backend via standard HTTP requests without authentication in this prototype phase.

4.6.4 Scheduling Engine Design

The scheduling engine uses Google OR-Tools CP-SAT to validate or generate feasible schedules. For the MVP, the solver operates on a small static dataset derived from sample facility usage and fixed fixtures.

Inputs

- A list of prototype events (title, facility, start time, end time).
- Facility definitions and availability windows.
- Fixed fixtures imported from ICS files.

Hard Constraints Implemented

- No two events may overlap on the same facility.
- Fixed events (fixtures) must not be moved.
- Events must end before the required cutoff time (21:45).

The solver returns a JSON representation of a conflict-free event list which the frontend displays.

4.6.5 Wireframes

The MVP user interface is based on the Figma wireframe and consists of the following core components:

- A **weekly grid** displaying Monday–Sunday with hourly divisions.
- A **sidebar** containing sample events or facilities.
- **Colour-coded states** for free or occupied facility times.

Although simplified, the wireframe provides a functional baseline for demonstrating system behaviour and validating scheduling outputs.

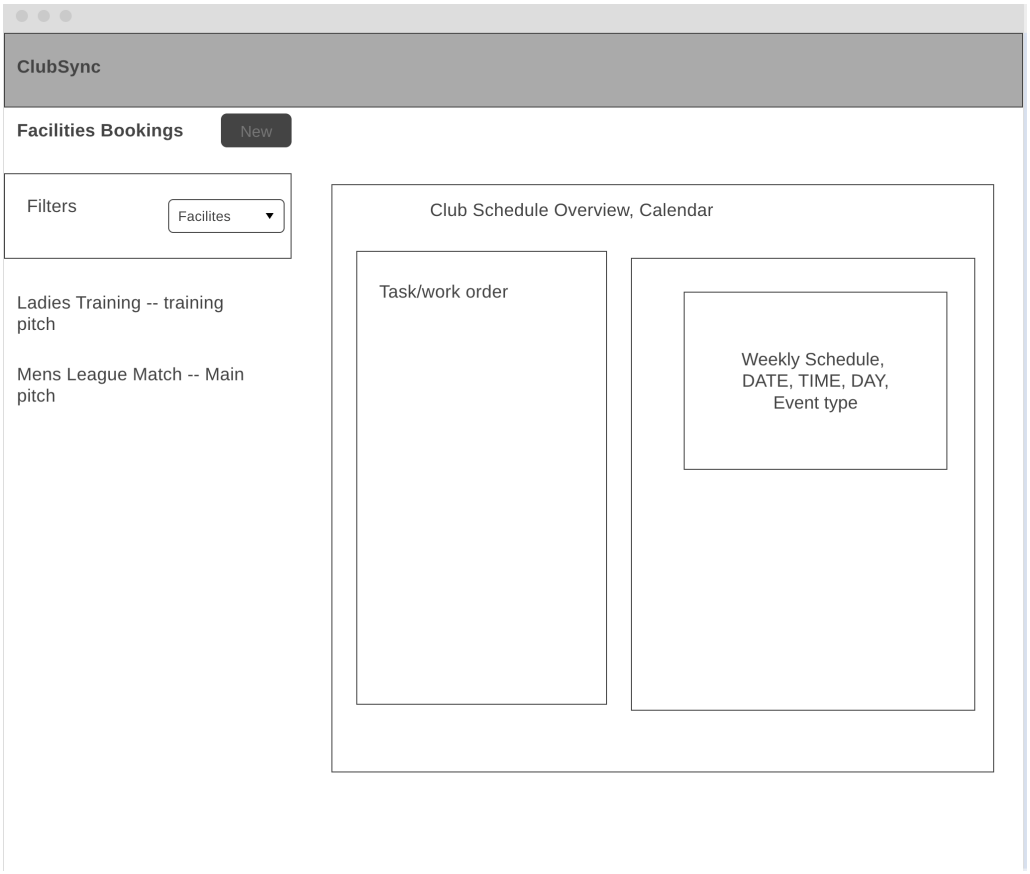

Figure 5: Wireframe of the ClubSync MVP interface showing facilities list, filters, and weekly schedule overview.

4.7 Conclusions

The design system for the ClubSync MVP establishes the essential architectural components, data model, and interactions required to support early scheduling functionality. While deliberately constrained for the interim phase, the modular design ensures that additional features such as drag-and-drop scheduling, real time updates, advanced constraint optimisation, and predictive analytics can be integrated seamlessly in future iterations.

5 Testing and Evaluation

5.1 Introduction

This section outlines the planned testing and evaluation strategy for the ClubSync prototype. As the interim submission precedes full implementation, the focus is on defining the methods, tools and test processes that will be applied during development of the MVP. The objective is to ensure that each component of the system (frontend, backend, database, and scheduling engine) functions correctly, integrates smoothly, and meets the user requirements gathered during the analysis phase

5.2 Plan for Testing

Testing of the MVP will follow an incremental, feature-driven approach in line with the project methodology. Each feature (database models, API endpoints, solver integration, and frontend schedule display) will be tested independently before being validated end-to-end.

5.2.1 Unit testing

- **Django Models:** Validation of Facility and Event model fields, constraints, and relationships using Django's built-in test framework.
- **API Endpoints:** Tests for `/api/facilities`, `/api/events` and `/api/schedule/solve` to confirm correct responses, HTTP codes, and error handling.
- **Scheduling Engine:** Unit tests to verify that OR-Tools correctly detects overlaps, respects fixed fixtures, and returns feasible schedules for sample datasets.

5.2.2 Integration testing

- **API + Database:** Ensure REST endpoints correctly read and write data to the PostgreSQL database.
- **Front-end + API:** Verify that the React schedule interface retrieves and displays updated event data correctly.
- **API + Solver:** Confirm that event data from the database is passed to OR-Tools in the correct format and that solver output is returned as valid JSON.

5.2.3 System testing

- Validate the entire scheduling workflow: loading facilities, viewing schedules, and running conflict checks.

- Check that free/busy states are accurately reflected on the weekly schedule grid.

5.2.4 User Acceptance Testing (UAT)

- Conduct small-scale evaluation sessions with stakeholders mentioned in previous sections. Coaches and Committee members (based on survey participants).
- Focus on usability, clarity of the schedule display, and correctness of solver-generated outputs.

5.3 Plan for Evaluation

Evaluation will focus on assessing whether the MVP meets the functional and usability requirements identified during the requirements gathering phase.

5.3.1 Functional Evaluation

- **Correctness of Conflict Detection:** Check whether the solver successfully prevents overlapping bookings and hours fixed events.
- **Accuracy of Free/Busy Display:** Compare front-end output with expected facility usage.

5.3.2 Usability Evaluation

- Short feedback sessions with end users to evaluate clarity of the schedule display, the ease of interpreting colour-coded indicators, and general user flow.
- Qualitative metrics gathered via a short feedback form.

5.3.3 Performance Evaluation (Prototype-Level)

- Measure time taken for the solver to validate the sample event dataset.
- Ensure that response time from the backend remains under one second for prototype use cases.

5.4 Conclusions

The planned testing and evaluation approach ensures that each feature of the ClubSync MVP is validated both individually and as part of the overall system. By combining unit, integration and user-focused testing, the project aims to develop a prototype that is functionally reliable, easy to use, and aligned with the scheduling needs identified in the requirements analysis. Further, the evaluation plan sets a clear pathway for assessing system correctness and usability during the next development phase.

6 System Prototype

6.1 Introduction

Approach to developing the prototype. The MVP prototype was implemented as a full-stack Django + React application with Google OR-Tools providing the constraint-solving engine.

6.2 Prototype Development

This first feature for the Prototype Development (FDD Software Methodology) will to be to implement the foundation layer for our MVP, (Database setup/schema) and to stand up our Django backend exposing relevant endpoints for our front-end (multi-tier system) application to consume and display information for User,

6.2.1 Database schema/setup

PostgresSQL was setup locally, using postgres psql downloaded via Homebrew on MacOS, local setup,

- Rellevant steps below show the Database creation with user and password verification

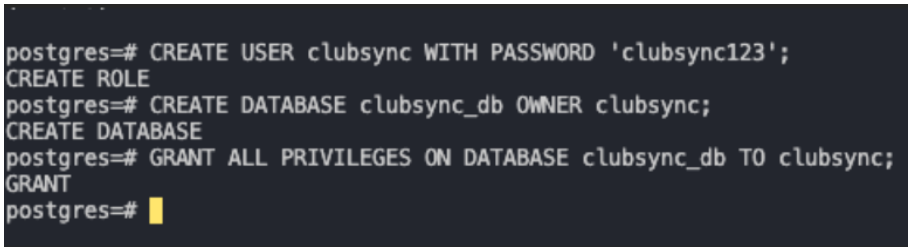

Figure 6: Commands used to create and grant access privileges for clubsync user

- Python virtual environment created below to isolate Django's backend dependencies

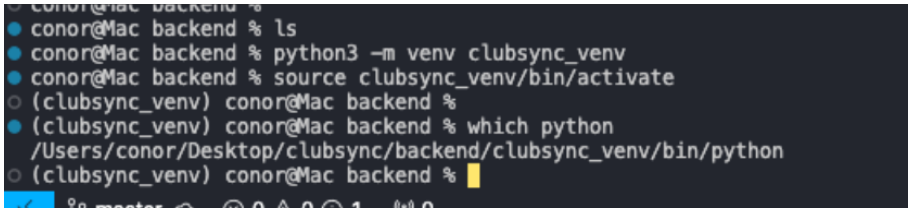

Figure 7: Ensuring packages and libraries are managed cleanly without conflicting with other projects in clubsync repo

6.2.2 Django Backend

After Database setup and Virtual Environment creation. Relevant configurations applied, Django backend project created and settings.py updated to configure the project with the previously created PostgreSQL database and required Python dependencies.

Listing 1: Django project created

```
pip install django djangorestframework psycopg2-binary python-dotenv
# create django project
django-admin startproject clubsync_project .
# create scheduler app for our endpoints
python manage.py startapp scheduler
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.getenv('DB_NAME'),
        'USER': os.getenv('DB_USER'),
        'PASSWORD': os.getenv('DB_PASSWORD'),
        'HOST': os.getenv('DB_HOST'),
        'PORT': os.getenv('DB_PORT'),
    }
}
```

Figure 8: loading env vars into settings, Ensuring best practises and coding standards

The backend consists of a single Django app named `scheduler` containing:

- `Facility` and `Event` models with a hard no-overlap constraint enforced via the `save()` method.

```
# scheduler/models.py
from django.db import models
from django.core.exceptions import ValidationError
from django.utils import timezone


class Facility(models.Model):
    name = models.CharField(max_length=100, unique=True)
    type = models.CharField(
        max_length=20,
        choices=[('pitch', 'Pitch'), ('hall', 'Hall'), ('gym', 'Gym')],
    )

    def __str__(self):
        return f"{self.name} ({self.get_type_display()})"
```

Figure 9: Facility model within the scheduler/models.py

- REST API endpoints exposing facilities and events.

Listing 2: scheduler/urls.py using DeafultRouter

```
from django.urls import path, include
```

```python
class Event(models.Model):
    title = models.CharField(max_length=200)
    start_time = models.DateTimeField()
    end_time = models.DateTimeField()
    facility = models.ForeignKey(Facility, on_delete=models.CASCADE, related_name='events')
    is_fixed = models.BooleanField(default=False, help_text="County fixtures cannot be moved")
    team_name = models.CharField(max_length=100, blank=True, null=True)
    def clean(self):
        if self.end_time <= self.start_time:
            raise ValidationError("End time must be after start time")

    def save(self, *args, **kwargs):
        # Hard constraint: no overlap on same facility
        overlapping = Event.objects.filter(
            facility=self.facility,
            start_time__lt=self.end_time,
            end_time__gt=self.start_time
        ).exclude(pk=self.pk)

        if overlapping.exists():
            raise ValidationError("This facility is already booked at that time.")

        super().save(*args, **kwargs)

    def __str__(self):
        return f"{self.title} - {self.facility}"

    class Meta:
        ordering = ['start_time']
```

Figure 10: Events model within the scheduler/models.py

```python
from rest_framework.routers import DefaultRouter
from .views import FacilityViewSet, EventViewSet

router = DefaultRouter()
router.register('facilities', FacilityViewSet)
router.register('events', EventViewSet)
```

- A custom management command `load_sample_data` that inserts realistic An Tóchar GAA events, including one deliberate overlap to demonstrate constraint enforcement.

```
backend > scheduler > management > commands > load_sample_data.py > Command > handle
```

```python
from django.core.management.base import BaseCommand
from scheduler.models import Facility, Event
from datetime import datetime
from django.utils import timezone


class Command(BaseCommand):
    help = "Load sample An Tóchar events (including one deliberate overlap to prove constraint)"

    def handle(self, *args, **options):
        # Get facilities
        main = Facility.objects.get(name="Main Pitch")
        training = Facility.objects.get(name="Training Pitch")
        hall = Facility.objects.get(name="Gym")

        events = [
            ("Senior Men vs Bray Emmets", "2025-04-05", "14:30", "16:30", main, True),
            ("U14 Boys Training", "2025-01-20", "18:30", "20:00", training, False),
            ("U12 Girls Training", "2025-01-20", "18:30", "19:30", training, False),  # mimics the overlap
            ("Senior Ladies Training", "2025-01-21", "19:00", "20:30", hall, False),
            ("U16 Boys vs Roundwood", "2025-03-15", "11:00", "12:30", main, True),
            ("U17 Strength & Conditioning", "2025-01-22", "19:30", "20:30", hall, False),
            ("U10 Football Skills", "2025-01-25", "10:00", "11:00", training, False),
            ("Senior Men Training", "2025-01-23", "19:00", "20:30", main, False),
            ("Committee Meeting", "2025-02-03", "20:00", "21:30", hall, True),
            ("U13 Football Blitz", "2025-05-10", "10:00", "14:00", main, True),
        ]

```

Figure 11: Management/commands folder loading data into events table

- Results shown below is the data loaded into the events table, but notice due to the

logic provided within the Events model in Figure 10.You can see that one event fails due to hard no-overlap constraint within events at database level entry, this is crucial step for our later implementation OR-tools Contrainst Programming solver

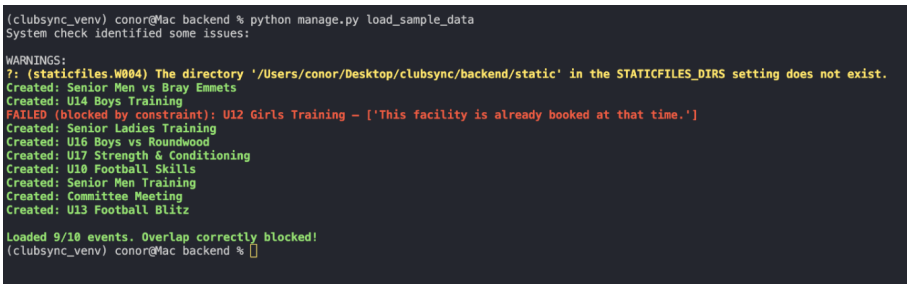

Figure 12: Successful entry into events table, constraint validated

- A `/api/schedule/solve/` POST endpoint that feeds all non-fixed events into OR-Tools CP-SAT and returns whether the schedule is feasible.

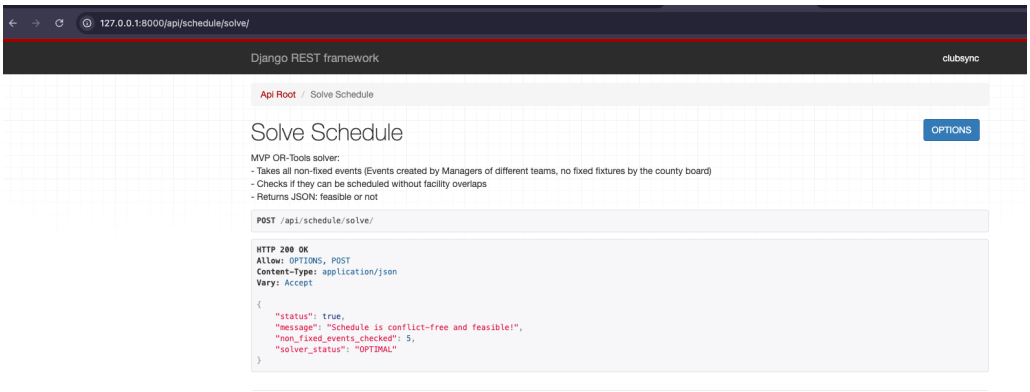

Figure 13: Shows results of endpoint validating the use of the scheduler taking in non fixed events, POST results status 200, JSON response returned

6.3 Evaluation

All functional requirements for the MVP were met:

- Real club data successfully imported and displayed.
- Hard constraint prevents overlapping bookings at database level.
- OR-Tools CP-SAT solver correctly validates schedule feasibility.

The prototype provides conclusive proof that constraint programming can eliminate double-bookings in a real GAA club environment.

6.4 Conclusions

7 Issues and Future Work

7.1 Introduction

This section summaries the main issues and risks identified to date and outlined the planned future work required to complete the ClubSync System. it builds directly on the Feature-Driven-Development approach adopted in Section 4, describing how remaining features will be delivered incrementally while managing technical and project risks.

7.2 Issues and Risks

There has been Several issues and risks have emerged during the interim phase:

- **Technical integration risk :** Integrating the Django Backend, Or-tools scheduling engine, and React front-end introduces complexity. Data Model changes or API mismatches could cause breakages across layers, especially as new features are added.
- **Scope and time risk :** The overall vision for ClubSync includes predictive analytics, drag-and-drop scheduling, role-based access control, and weather aware planning. These are beyond the MVP and create a risk of scope creep if not carefully phased.
- **Data quality and availability :** Real club data may be incomplete, inconsistent, or noisy. Poor-quality input data can reduce the usefulness of both constraint programming and any future predictive models. When dealing with Data provided its crucial to make sure this is cleaned and fit for purpose.
- **Security and Privacy :** Although full GDPR-compliant implementation is deferred to later phases, handling personal data (availability, attendance, contact details) will require secure authentication, data protection, and careful access control that are only partially addressed in the MVP.

7.3 Plans and Future Work

Future work will follow the FDD methodology by implementing small, well-defined features on top of the existing MVP foundation. Key planned areas include:

- **Enhanced Scheduling and Editing:** Extend the OR-Tools model to support additional soft constraints (coach preferences, age-appropriate times, changeover buffers beyond the MVP). Introduce basic schedule editing (e.g. moving or cancelling events) with re-validation, preparing the ground for a full drag-and-drop

interface.

- **Improved frontend UX:** Evolve the React schedule view into a more interactive interface, including filters by team/facility and clearer free/busy indicators. Align the implementation more closely with the Figma wireframes and gather feedback from real club stakeholders.
- **Role-Based access and basic security:** Add simple role-based access control (admin vs coach vs viewer) to the Django backend. Prepare for future authentication and authorisation features needed for GDPR compliance in the final system.
- **Data Ingestion and robustness:** Generalise the ICS import and sample management command into a more robust pipeline for loading and cleaning club calendars. Add validation and error reporting for malformed or overlapping input events to reduce data-quality risks.
- **Initial foundations for predictive analytics:** While full predictive analytics is beyond the interim MVP, the next phase will focus on collecting and structuring historical data so that basic prediction experiments (e.g. simple demand forecasts) can be run later in the project.
- **Process and evaluation work:** Continue applying FDD by designing and building each feature in small iterations, with regular checkpoints against the original aims and scope. Plan and execute all relevant tests mentioned in section 5 above and user acceptance testing with coaches and administrators to evaluate usability, solver correctness, and perceived time savings compared to existing processes.

7.4 Project Plan with GANTT Chart

Break down a realistic schedule. Include Gantt chart.

References

Bedo, Mohamed (2023). *Model-View-Template (MVT) Architecture Pattern*. Accessed: 2025-01-30. URL: https://medium.com/@m.bedo20012015/model-view-template-mvt-architecture-pattern-32786118ca1b.

Club & County (2025). *Club & County: GAA Club Management*. URL: https://clubandcounty.com/ (visited on 11/19/2025).

Clubforce (2025). *Clubforce: All-in-One Sports Club Management*. URL: https://clubforce.com/ (visited on 11/19/2025).

ClubZap (2025). *ClubZap: Club Management Software*. URL: https://clubzap.com/ (visited on 11/19/2025).

Foireann (2025). *Foireann – Official GAA Membership & Games Management*. URL: https://www.foireann.ie/ (visited on 11/17/2025).

Foundation, Django Software (2025). *The Django Web Framework*. URL: https://www.djangoproject.com (visited on 10/18/2025).

GAA (2025). *Managing Burn-Out and Stress as a GAA Volunteer*. URL: https://www.gaa.ie/article/managing-burn-out-and-stress-as-a-gaa-volunteer (visited on 11/18/2025).

Gaelic Athletic Association (2025). *GAA Annual Report 2024*. Accessed: 17 November 2025. URL: https://www.gaa.ie/article/gaa-report-healthy-financial-year-for-2024 (visited on 11/17/2025).

Google (2025). *Google OR-Tools – Constraint Optimization*. URL: https://developers.google.com/optimization (visited on 11/26/2025).

Group, PostgreSQL Global Development (2025). *PostgreSQL 17 Documentation*. URL: https://www.postgresql.org/docs/current/ (visited on 11/26/2025).

Meta (2025). *React – A JavaScript library for building user interfaces*. URL: https://react.dev (visited on 11/26/2025).

Moynihan, Michael (2020). *Warning of "silent burnout" among GAA volunteers and officers*. URL: https://www.irishexaminer.com/sport/gaa/arid-30817385.html (visited on 11/18/2025).

React (2025). *Full Calendar React component*. URL: https://fullcalendar.io/docs/react (visited on 11/26/2025).

Spanne, Anton (2024). *Constraint Programming for Sports Scheduling*. URL: https://www.cupmanager.net/wp-content/uploads/2024/07/scheduling_cm.pdf.

SportLoMo (2025). *SportLoMo: Sports Management Software*. URL: https://www.sportlomo.com/ (visited on 11/19/2025).

TeamSnap (2025). *TeamSnap: Youth Sports Management*. URL: https://www.teamsnap.com/ (visited on 11/19/2025).

Vite – Frontend Tooling (2025). URL: https://vitejs.dev (visited on 11/26/2025).

Zhang, Ling (20225). *Enhancing Sports Yeam Management Through Machine Learning*. URL: https://www.researchgate.net/publication/389959214_Enhancing_Sports_Team_Management_Through_Machine_Learning.

Appendix A: Survey Results Summary

78 percent of administrators reported frequent pitch double-bookings; 100 percent currently use WhatsApp for coordination.

Appendix B: Key Code Samples

- Hard no-overlap constraint in `Event.save()`
- OR-Tools CP-SAT solver call in `solve_schedule()`
- Sample data loading command output showing blocked overlap

Appendix C: Screenshot Evidence

1. Django admin with real An Tóchar events
2. Blocked overlapping booking error
3. `POST /api/schedule/solve/` → feasible response

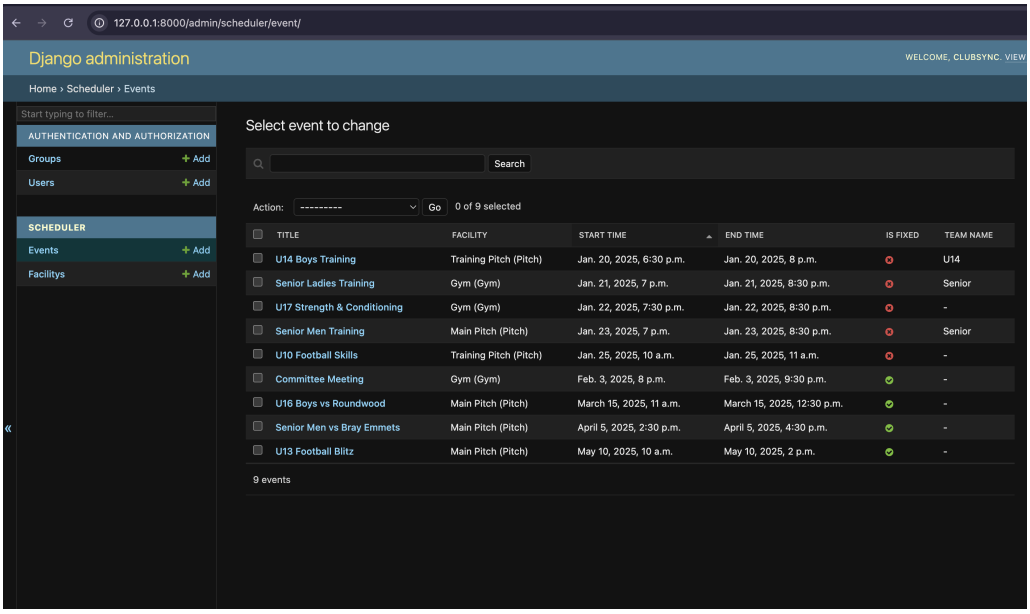

Figure 14: Django admin page on events table after data loaded

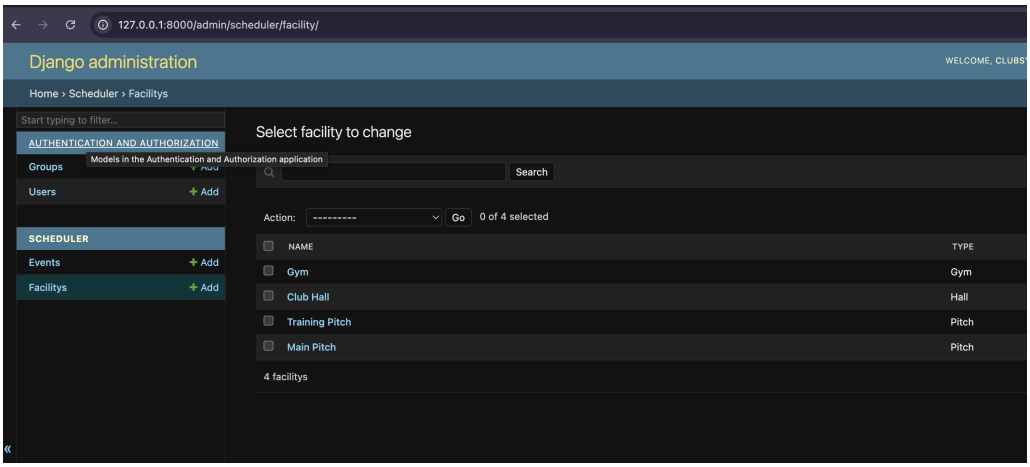

Figure 15: Django admin page on facilities table

Figure 16: POST to endpoint /api/schedule/solve/