# -Project Plan RePlan-

By Matei Copoeru

**V1.1**

## **1. Executive Summary**

**Project Title:** RePlan

**Context:** In the amateur eSports scene a lot of teams are extremely unorganised or have very inefficient methods of keeping up with schedules. Some games also don’t provide a unified Video on Demand Replay system, so players have to record their own POV and upload it onto YouTube so everyone can look at it and learn. There is a clear space here for a tool that could help with both team scheduling and reviewing gameplay.

**Objective:** Develop a web application that enables teams to annotate video content with timestamped notes, supports multi-user sessions, provides real-time updates, and integrates scheduling features with team management capabilities. The application aims to enhance collaboration—for example, in sports teams, eSports, or coaching groups—through efficient video review and scheduling tools.

**Technology Stack:**

* **Frontend:** React, Shadcn, Tailwind CSS, Axios
* **Backend:** Spring Boot (Java) following Clean Architecture principles with WebSocket for real-time updates and Microsoft’s AZURE SQL server for storage

**Key Deliverables:**

* Video embedding with timestamped note-taking
* Multi-user session support with persistent note storage
* Real-time note updates via WebSocket integration
* Team management module with role assignments
* Practice scheduling with availability tracking and conflict detection
* Optional notifications
* Polished UI, thorough testing, and production deployment

**Updated Timeline:** With the first sprint already completed, the remaining work is organized into **5 sprints** (four 3‑week sprints and one final 2‑week sprint), totaling **14 weeks** of work.

## **2. Project Scope**

### **In Scope**

* **Basic Setup & Video Notes:** *[Completed]*
  + Frontend project established using React with Shadcn and Tailwind CSS.
  + Backend initialized with Spring Boot and Clean Architecture principles.
  + YouTube embedding and basic timestamped note-taking implemented.
* **Multi-User Support:**
  + Integrate a database to persist notes.
  + Enable multi-user access with unique session IDs and provide API endpoints consumed via Axios.
* **Real-Time Notes:**
  + Implement real-time note updates using Spring Boot’s WebSocket (or STOMP) integration.
  + Synchronize notes across users and display authorship.
* **Team Management:**
  + Allow users to create/join teams and manage roles (e.g., Coach, Player, Analyst).
  + Include invitation and removal functionalities.
* **Practice Scheduling:**
  + Provide tools for users to mark availability and for admins to schedule sessions.
  + Develop a calendar UI with conflict detection.
* **Video Syncing (Optional):**
  + Enable synchronized video controls (play, pause, seek) across users with an opt in/out option.
* **UI & Final Testing:**
  + Ensure responsive design with dark mode, comprehensive error handling, and bug fixes.
* **Deployment:**
  + Deploy the application on a cloud platform (e.g., AWS, Azure) with final production testing.

### **Out of Scope**

* Integration with third-party analytics beyond basic notifications (e.g., Discord)
* Advanced video processing beyond YouTube embedding and simple synchronization
* Native mobile applications (responsive web design only)

## **3. Scrum Methodology & Sprint Breakdown**

Since the first sprint (Basic Setup & Video Notes) is completed, the remaining work is organized into 5 sprints as follows:

* **Sprint 1 (3 weeks): Multi-User Support & Real-Time Notes** *Focus:*
  + Integrate database persistence and develop API endpoints for multi-user sessions.
  + Implement WebSocket-based real-time note updates.
* **Sprint 2 (3 weeks): Team Management & Initial Scheduling Features** *Focus:*
  + Develop functionalities for team creation/joining and role management.
  + Begin implementation of the basic scheduling module.
* **Sprint 3 (3 weeks): Advanced Scheduling & Conflict Detection** *Focus:*
  + Enhance the scheduling system with a comprehensive calendar UI.
  + Integrate conflict detection algorithms and improve the admin view.
* **Sprint 4 (3 weeks): Video Syncing & UI Polish** *Focus:*
  + Implement optional video syncing features (synchronized controls).
  + Refine UI/UX, including dark mode, error handling, and responsiveness.
* **Sprint 5 (2 weeks): Final Integration, Comprehensive Testing & Deployment** *Focus:*
  + Integrate all modules and perform extensive testing (unit, integration, and end-to-end).
  + Finalize documentation and deploy the application to production.

*Total Remaining Duration: 18 weeks*

## **4. Roles & Responsibilities**

Since this is a solo project, a single developer will manage all roles:

* **Solo Developer:**
  + **Project Management:** Plan the timeline, monitor progress, manage risks, and maintain project documentation.
  + **Development:** Architect and implement both frontend and backend systems.
  + **Testing:** Create and execute unit, integration, system, and acceptance tests aiming for 80% code coverage.
  + **UI/UX:** Design and refine the application interface including dashboards, scheduling views, and admin panels.
  + **DevOps:** Set up and manage hosting, CI/CD pipelines, and configuration management.

## **5. Testing Strategy and Configuration Management**

**Testing Strategy:**

* **Unit Testing:**
  + **Backend:** Use JUnit and Mockito to test individual components.
  + **Frontend:** Use Jest (or similar frameworks) to test UI components and API interactions.
* **Integration Testing:**
  + Validate interactions between multi-user modules, WebSocket communications, and scheduling components.
* **System & End-to-End Testing:**
  + Ensure complete application flows function as intended with automated tests.
* **Acceptance Testing:**
  + Conduct final tests to ensure the product meets requirements.
* **Code Coverage:**
  + Target a minimum of 80% coverage across the codebase.

**Configuration Management:**

* **Version Control:**
  + Use Git with a feature-branching strategy, ensuring regular commits and code reviews (self-review).
* **Release Management:**
  + Automate deployments via CI/CD pipelines, with staging releases before production rollout.
* **Documentation:**
  + Maintain updated documentation, including API docs, system architecture diagrams, and developer guides.

## **6. Risk Management**

**Key Risks & Mitigation Strategies:**

* **Scope Creep:** *Risk:* Additional feature requests beyond the initial scope.  
   *Mitigation:* Maintain strict change management and use sprint reviews to manage scope.
* **Technical Integration Challenges:** *Risk:* Difficulties integrating WebSocket real-time updates or third-party APIs.  
   *Mitigation:* Implement early prototyping and continuous integration testing.
* **Resource Bottlenecks:** *Risk:* Delays due to working solo on multiple roles.  
   *Mitigation:* Use time management strategies, plan realistic goals, and adjust sprint scopes if necessary.
* **Deployment Issues:** *Risk:* Challenges with cloud deployment, security, or performance in production.  
   *Mitigation:* Engage in early DevOps setup, perform load testing, and maintain a rollback plan.

## **7. Communication & Reporting**

As a solo project, formal communication channels are simplified:

* **Daily Self-Check-ins:**
  + Brief reviews to track progress and adjust priorities.
* **Sprint Reviews:**
  + End-of-sprint self-assessments to document progress and lessons learned.
* **Project Dashboard:**
  + Utilize a personal project management tool (e.g., Trello or Jira) to keep tasks organized.
* **Documentation:**
  + Maintain comprehensive documentation in a shared repository (for future reference).

## **8. Final Deliverables & Handover**

* **Source Code Repository:**
  + Complete, well-documented code with version control history.
* **Deployment Package:**
  + Scripts and detailed documentation for cloud deployment.
* **User Documentation:**
  + Guides covering video annotation, team management, and scheduling features.
* **Technical Documentation:**
  + Detailed API documentation, system architecture diagrams, and developer guides for future maintenance.
* **Post-Launch Support Plan:**
  + A self-managed support plan to address any issues post-deployment.

## **9. Conclusion**

This updated project plan for RePlan now reflects a solo project structure with a revised Agile sprint breakdown. With the first sprint completed, the remaining work is organized into five sprints (four 4‑week sprints and a final 2‑week sprint), totaling 18 weeks of focused work. By consolidating roles and responsibilities and incorporating a detailed testing and configuration management strategy, this plan is designed to deliver a high-quality, feature-rich application with continuous self-assessment and iterative improvement throughout the development process.