Enhancement 3 Narrative: Database Integration

CS-499 Computer Science Capstone - Category 3: Databases

Ian Repsher
30 November 2025
iOS Backend Integration with PostgreSQL Database

Artifact Description

The artifact is the RPSMF (Rock Paper Scissors Middle Finger) iOS game application, originally created in CS-330 (Mobile Architecture and Programming) in November 2024. The original artifact was a standalone iOS game with local gameplay only, featuring a bot opponent and no persistent player data or online capabilities.

For Enhancement 3, I transformed this local application into a full-stack system by: 1. Designing and implementing a PostgreSQL database with a comprehensive relational schema 2. Building a Node.js/Express REST API backend with JWT authentication 3. Creating an iOS networking layer to integrate with the backend 4. Implementing user authentication, profile management, and leaderboard functionality

The enhancement spans approximately 3,500 lines of new code across backend (Node.js/PostgreSQL) and frontend (Swift) components.

Justification for Inclusion

I selected this artifact for the databases category because it provided an ideal foundation for demonstrating enterprise-level database design and integration skills. The original game had all the components needed for a competitive online system but lacked the critical database infrastructure to track players, matches, statistics, and rankings.

Skills Showcased

Database Design and Architecture: - Designed a normalized relational database schema with 8 interconnected tables - Implemented proper primary/foreign key relationships and constraints - Created database views for complex aggregations (player statistics, leaderboard rankings) - Applied database normalization principles to avoid data redundancy

Backend API Development: - Built RESTful API with Node.js and Express following industry best practices - Implemented secure JWT-based authentication with bcrypt password hashing - Created connection pooling for efficient database access - Developed comprehensive API endpoints for authentication, player data, and leaderboards

iOS Integration: - Designed a robust networking layer with async/await patterns - Implemented proper error handling and data model serialization - Created authentication flow with session persistence - Built dynamic UI components that display real-time database data

Security Implementation: - Secured all endpoints with JWT authentication middleware - Implemented password hashing with bcrypt (12 salt rounds) - Protected sensitive routes requiring user authentication - Validated all database inputs to prevent SQL injection

---

Improvements Made

The artifact was improved by adding: 1. **PostgreSQL Database**: Complete relational schema with users, matches, player_stats, achievements, and inventory tables 2. **Backend API**: 20+ REST endpoints for authentication, player management, statistics, and leaderboards 3. **iOS Networking Layer**: APIClient singleton with JWT authentication and session management 4. **User Interface**: Login/registration, profile view with stats and ELO rating, and global leaderboard 5. **ELO Rating System**: Competitive ranking system that updates based on match outcomes 6. **Data Persistence**: All user progress, statistics, and achievements stored in PostgreSQL

Course Outcomes Coverage

Original Plan (Module One)

I planned to meet the following course outcomes with this enhancement: - **Outcome 4**: Demonstrate ability to use well-founded techniques, skills, and tools in computing practices for implementing solutions that deliver value - **Outcome 5**: Develop a security mindset that anticipates adversarial exploits and ensures privacy and enhanced security

Achievement Analysis

Outcome 4 - Fully Met: I successfully demonstrated proficiency in database design and full-stack development by: - Implementing a production-ready PostgreSQL database with proper schema design - Building a secure REST API following Node.js best practices - Creating efficient database queries with connection pooling - Integrating the iOS frontend seamlessly with the backend - Using industry-standard tools (PostgreSQL, Express, JWT, bcrypt)

Outcome 5 - Fully Met: I applied security principles throughout the implementation: - JWT authentication with token-based authorization - Bcrypt password hashing with appropriate salt rounds - SQL injection prevention through parameterized queries - Authentication middleware protecting sensitive endpoints - Secure token storage and session management in iOS - Input validation on both frontend and backend

Updates to Coverage Plans

No updates needed. The enhancement met and exceeded the planned outcomes. Additionally, this work demonstrates progress toward **Outcome 3** (algorithmic principles) through the ELO rating calculation algorithm that adjusts player rankings based on match outcomes.

Reflection on Enhancement Process

What I Learned

Database Schema Design: The most valuable learning came from designing a relational schema that balances normalization with query performance. I learned to think about data relationships holistically - for example, creating the `player_stats` view to aggregate match data rather than storing redundant statistics in the users table.

Backend API Architecture: Building the Node.js backend taught me the importance of separation of concerns. I structured the code with distinct layers: routes for endpoints, middleware for authentication, and database pool management for connections. This architecture makes the code maintainable and testable.

iOS Networking Patterns: Implementing the iOS networking layer deepened my understanding of async/await patterns in Swift and the importance of proper error handling in network requests. I learned to handle various edge cases like token expiration, network failures, and data serialization mismatches.

Security Best Practices: Working with authentication and authorization reinforced the importance of defense-in-depth. I learned that security isn't just about one technique (like password hashing) but about layering multiple protections: JWT tokens, middleware authentication, input validation, and parameterized queries.

Full-Stack Integration Challenges: Connecting the iOS app to the backend revealed the complexities of full-stack development. I encountered issues with: - **Case sensitivity**: The API returned camelCase JSON but iOS expected snake_case, requiring careful CodingKeys configuration - **Type mismatches**: Database rank fields returned as strings instead of integers, requiring custom decoders - **Session management**: The app needed to restore user sessions on launch by fetching current user data when a saved token existed

---

Challenges Faced

Challenge 1: Database View Performance
Initially, I created the leaderboard by querying multiple tables and calculating ranks in the application code. This was inefficient. I learned to leverage PostgreSQL's window functions (ROW_NUMBER()) to create a database view that pre-calculates rankings, significantly improving query performance.

Challenge 2: iOS Type Decoding
The most frustrating challenge was debugging why the Profile and Leaderboard tabs showed "data couldn't be read" errors. Through systematic debugging, I discovered the API returned camelCase field names, but I had configured the iOS decoder to expect snake_case. This taught me the importance of validating data contracts between frontend and backend early in development.

Challenge 3: Session Restoration
When the app restarted with a saved authentication token, the profile screen showed "Profile Not Found" because `currentUser` was nil even though `isAuthenticated` was true. I solved this by modifying the APIClient initialization to automatically fetch current user data when restoring a session, ensuring consistent state.
Challenge 4: Shared State Management
The iOS views initially used `@StateObject` which created separate APIClient instances, preventing them from seeing the authenticated user. I learned about SwiftUI's environment object pattern and refactored to use `@EnvironmentObject`, ensuring all views share the same APIClient instance with proper state synchronization.

Technical Growth

This enhancement significantly expanded my technical capabilities: - **Database expertise**: From basic SQL to advanced schema design, views, and query optimization - **Backend development**: Building production-ready APIs with proper authentication and error handling - **Mobile integration**: Creating robust networking layers that handle real-world complexities - **Security implementation**: Applying multiple security layers to protect user data - **Problem-solving**: Systematically debugging full-stack issues by isolating problems at each layer

The most important lesson was learning to think in terms of complete systems rather than individual components. Every decision - from database column types to iOS model structures - affects the entire stack. This holistic perspective is essential for building reliable, scalable applications.