Final Report

Slot machine game built using MEAN stack: **MongoDB + Express + Angular + Node.js**

Project Title:

Spin & Win – Interactive Slot Machine Web Application

1. Introduction

This project is developed as a complete full-stack web application that enables users to register an account, manage their balance, play a slot-machine–style mini-game, and compete on a leaderboard. The project aims to demonstrate full CRUD operations, secure backend communication, authentication, and a clean separation between the frontend and backend using MEAN stack.

Our project consists of the following:

- A responsive **Angular frontend** that manages routing, authentication state, forms, and gameplay UI.
- A **Node.js + Express backend** that exposes RESTful API endpoints for user registration, login, spin computation, transaction handling, and leaderboard access.
- A **MongoDB database** that stores persistent data such as users and transaction history, which includes the win/lose status, bet amount, and timestamps.

Users can register, log in, play the game, view their transaction history, and see how they rank compared to others.

2. Architecture and Tech Stack

System Architecture Overview

The application follows a clear three-layer structure:

Frontend (Angular)

- Presents UI components for login, registration, home page, slot game, profile (that includes recharge system and transaction history), and leaderboard.
- Uses Angular services to call backend APIs.

- Implements route protection with Auth Guards.
- Handles dynamic updates to user balance and leaderboard display.

Backend (Node.js + Express)

- Implements all RESTful endpoints.
- Uses controllers for user, spin, and auth logics.
- Includes middleware for JWT verification and error handling.
- Provides secure and consistent responses to the frontend.

Database (MongoDB with Mongoose)

- Stores users and transactions collections.
- Enforces schema validation.
- Provides auto-updated fields like timestamps.

Data Flow Summary

1. The frontend sends authenticated requests (email and password) to the backend.
2. The backend validates the password, generates JWT, and interacts with MongoDB.
3. MongoDB stores and retrieves data via Mongoose models.
4. Responses are sent back to Angular, where the UI updates accordingly.

Tools and Libraries Used

- **Angular**: Component-based structure, routing, HttpClient
- **Bootstrap/CSS**: UI styling
- **Node.js & Express**: Backend structure and routing
- **Mongoose**: MongoDB object modeling
- **JSON Web Tokens (JWT)**: Secure authentication
- **MongoDB Atlas**: Cloud database hosting

These technologies were chosen because they align with modern industry practices and course requirements.

3. Data Model

User Collection

Key Fields:

- Username
- Email (unique)
- Password (hashed)
- Balance
- CreatedAt / UpdatedAt

Validation Rules:

- Email format validation
- Required username
- Password must exist but is always stored hashed

Transactions Collection

Key Fields:

- User ID
- Type (“spin”, “recharge”)
- Amount (+ or -)
- Result (“jackpot”, “big win”, etc.)
- Timestamp

Validation Rules:

- Required user ID
- Amount must be a number
- Type must be one of the predefined options

Relationships

- One user can have many transactions.
- Each transaction has only one user ID.
- For fair play, Leaderboard is determined based on “spin” type transactions, not balance nor recharged amount.

4. Implementation Details

Core Features Implemented

- **User Authentication**:
  Register, login, protected backend routes, JWT storage on frontend, route guarding.

- **Slot Machine Gameplay**:
  Users spend balance to spin, random symbols generate a win or loss, and balance updates live.
- **Transaction History**:
  Every balance change is automatically recorded as a transaction.
  The UI shows a list of these records with amounts and timestamps.
- **Dynamic Leaderboard**:
  Ranks users based on balance.
  Updates whenever a user spins and their balance change.
- **Protected Pages**:
  Using Angular's AuthGuard, pages like play slot, profile, and leaderboard cannot be accessed without logging in.
  Manual URL typing redirects users back to login.
- **Responsive UI**:
  Bootstrap-styled layout with compact cards and a clean visual hierarchy.

Major Design Decisions

- **JWT stored in localStorage**:
  Simple for project-level authentication and supported by Angular interceptors.
- **Leaderboard recalculated via service whenever needed**:
  Instead of relying on page reloads, the leaderboard refreshes on spin completion.
- **Separation of Routes/Controllers/Models**:
  Improves code organization and makes the project easier to understand for instructors reviewing it.
- **Transaction-based balance updates**:
  Rather than updating balance directly, every change flows through a transaction entry.
  This ensures full accuracy and traceability.

5. Challenges and Solutions

Challenge 1: Transactions not updating immediately after navigating tabs

Cause:
Angular reused the component, and the transaction list did not refresh because the state persisted.

Solution:
We restructured the transaction component to fetch fresh data on each load and reset temporary messages. This ensured transaction feedback cleared properly when navigating.

Challenge 2: Leaderboard not dynamically updating after each spin

Cause:
The leaderboard was originally only loaded on component initialization.
When the user spun the slot machine, the updated balance was saved but the leaderboard did not refresh.

Solution:
We updated the Spin service to trigger a leaderboard refresh event and subscribed to it in the leaderboard component. Now the leaderboard updates instantly after a spin.

Challenge 3: AuthGuard not blocking manually typed URLs

Cause:
Routes were protected visually (buttons hidden), but users could still type URLs directly.

Solution:
We implemented a proper Angular AuthGuard checking for stored JWT tokens before allowing route access. Unauthorized attempts redirect to the login page. The backend also validates JWT for double protection.

6. Future Improvements

I omitted some logics that might have helped the application run more smoothly and applicable to bulk users. These improvements could be achieved if I had more time.

1. **Add real-time leaderboard updates using WebSockets**
   Currently, updates occur when the user triggers actions. WebSockets would allow live updating across all users.
2. **Add pagination or filters to transaction history**
   Transaction lists could grow large over time. Filtering by date or type would improve usability.
3. **Add animation and sound effects to the slot machine**
   Enhancing the gameplay experience would make the app more engaging.
4. **Add dark mode toggle**
   Angular's theming system would allow quick switching between themes for better accessibility.

Conclusion

The Spin & Earn application successfully integrates Angular, Node.js/Express, and MongoDB into one coherent, secure, and interactive full-stack system. The project demonstrates authentication, CRUD operations, UI design principles, route protection, transactional accuracy, and dynamic updates.

It took a lot of time and energy, but the outcome is satisfactory, resulting in a well-structured academic project that showcases practical full-stack development skills and clean architectural understanding.