COVERING NOTE

Project: Full Stack Code Development for "AllTheBeans" Business

Summary:

This project was implemented as a full-stack solution to manage and display available beans, support customer orders, and provide a modern, responsive user experience. It features backend-driven search functionality and business logic such as "Bean of the Day". The solution leverages Angular for the frontend, ASP.NET Core Web API for the backend, and SQL Server with Entity Framework Core for data persistence. API testing is supported through xUnit to ensure reliability and maintainability.

Angular was chosen because it provides built-in support for components, routing, forms, and services, making it suitable for enterprise-scale applications. Its component-based architecture enforces a clean separation of concerns, improves readability, and supports scalability through modular design, while TypeScript strengthens testability and maintainability.

ASP.NET Core provides a fast, lightweight, and secure foundation for building backend services. Its built-in support for dependency injection and asynchronous APIs helps keep controllers clean and focused, with business logic neatly organised in separate services, making the code easier to read, maintain, and test. Security is ensured by features like controller, model validation, and serialization, which help to prevent common vulnerabilities. Additionally, it works efficiently with Entity Framework Core, streamlining data access and persistence.

SQL Server provides a reliable and robust relational database with strong transaction guarantees, making it well-suited for handling structured data. Paired with Entity Framework Core, it offers developer-friendly, type-safe LINQ queries that translate into efficient SQL. This keeps things simple by mapping data models directly to database tables like Beans and Orders, while supporting scalability, filtering, and sorting to maintain consistent performance with large datasets.

xUnit enables thorough validation of business logic, controllers, and data access layers independently, promoting a clean separation of concerns and easier maintenance. The use of an in-memory database simplifies the testing setup, removing the need for external dependencies and making it faster and more practical to run tests frequently throughout the development cycle.

Across the stack, security is achieved through input validation, serialization, and server-side rules. Performance is maintained through efficient EF Core queries and API-level pagination, ensuring the system remains responsive under load. A clear separation of concerns between controllers, services, and models enhances readability and maintainability, while modular design and dependency injection support testability by allowing components to be verified independently. Scalability is built in, with RESTful API boundaries enabling the frontend and backend to evolve separately. Throughout, simplicity is preserved by using each framework purposefully and avoiding unnecessary complexity.

Together, these choices form a clean, secure, responsive, and testable architecture that reflects Tombola's values of readability, testability, scalability, security, and simplicity, ensuring the solution is both robust today and adaptable for future growth.

Business Logic Choice: "Bean of the Day"

A database-driven approach was chosen to ensure daily randomness while preventing consecutive repeats. This method guarantees consistency across servers, as all users see the same selection regardless of where they connect. It also provides persistence and auditability, allowing the system to retain historical data for future analysis. Unlike in-memory or cached solutions, this setup maintains reliability even after restarts. By combining controlled randomness with business rules, it strikes a balance between unpredictability and user engagement, avoiding the predictability of rotation-based methods.