Tablebookingapi

Project development plan

Tallinna Tööstushariduskeskus

Martin Rosman

Jegor Skljar

**TableBookingAPI Project Development (10 Steps)**

**Process Stages:**

**1. Gathering Project Requirements:**  
Discuss with the client and align on the project's goals and functionalities.

Develop APIs for managing clients, reservations, and tables.

Provide CRUD operations (Create, Read, Update, Delete) for all entities.

Enable interaction between reservations and related clients/tables.

**2. Selecting a Technical Solution**

Backend Framework: Use Express.js for creating a RESTful API. It's lightweight, flexible, and widely supported.

Programming Language **Node.js with JavaScript:** This ensures asynchronous processing, making it suitable for handling multiple API requests efficiently.

Use **Swagger** for maintaining a comprehensive and accessible API reference. This ensures clear communication of endpoint functionalities, parameters, and response formats.

3. **Project Planning:**

Features and functionality of the system:

* Client Managemenent

Add, Update and delete client information.

* Reservation Management

Manage reservations, including date number of adults and children, assigned tables.

* Table Managemement

Track table availability.

Assigned Tasks:

* Backend Developing

Impliment API Logic and routes.

* API testing

Test all endpoints using Swagger.

* Techical writing

Write documentation.

Milestones and deadlines:

* Initial API setup and server running
* Completed modules for clients, reservations, and tables.
* Integrated Swagger documentation.
* Fully tested and deployed API.

**4. Creating the Initial Codebase**

**Set Up the Development Environment**

1. Install the required tools and dependencies:

* Node.js for running the backend server.
* Express.js as the web framework.
* Swagger for API documentation.
* CORS Middleware for handling cross-origin requests.
* Initialize the project with npm init and install necessary packages:

npm install express swagger-ui-express cors

1. Create the Basic Project Structure

* index.js: Main entry point of the application.
* routes/: Folder for all API routes.
* docs/: Swagger documentation (e.g., swagger.json).

**5. Code Testing**

This stage is critical to ensure that the **reservation management system** performs reliably, meets the requirements, and is ready for deployment or further development.

Test the APIs manually using tools like **Swagger UI**:

* Test all endpoints (GET, POST, PUT, DELETE).
* Check edge cases (e.g., adding duplicate clients, invalid table IDs).
* Validate response times and behavior.

Ensure all endpoints meet the functional requirements of the system:

* Can clients be retrieved, added, updated, and deleted?
* Can reservations be managed (created, updated, or removed)?
* Are table data operations working correctly?

**6. Bug Fixing**

This stage ensures the reservation management system runs smoothly, efficiently, and without errors.

1. Bug Identification

* User Feedback: Collect feedback from testers, developers, or users to identify issues.
* Manual Testing: Perform thorough manual testing of all system features:
* Test API endpoints for incorrect responses or failures.
* Validate edge cases like empty inputs, invalid data, or extreme conditions.
* Automated Tests: Review results from automated unit tests, integration tests, and end-to-end tests.
* Error Logs: Analyze application logs to locate bugs.

2. Prioritizing Bugs

* Categorize bugs based on severity and priority:
* Critical Bugs: Break the core functionality (e.g., reservation creation fails).
* Major Bugs: Affect important functionality but don't crash the system.
* Minor Bugs: Small issues like typos, slow responses, or formatting problems.

**7. Creating Documentation**

Creating comprehensive documentation for the project to ensure clear understanding, proper usage, and easier maintenance for developers, testers, and end-users. Good documentation serves as a roadmap for anyone interacting with the project in the future.

1. API Documentation

Create clear, detailed API documentation that explains how each endpoint works, what parameters it accepts, and what responses it returns.

Use tools like Swagger to generate and display API documentation.

2. Codebase Documentation

Provide comments and explanations directly within the codebase to make it easier to understand. Follow these practices:

Function-Level Comments: Explain the purpose of each function, its parameters, and return values.

3. Setup and Installation Guide

Document how to set up and run the project. Include the following details:

* Prerequisites: List the tools and software required. Example:
* Node.js v14 or higher
* npm
* Installation Steps: Provide step-by-step commands to install and run the project.

**8. Finalizing the version**

This stage ensures that the application is stable, optimized, and production-ready. All components of the project undergo a final review, testing, and validation before being released to users or deployed to production.

1. Code Freeze

* Implement a code freeze, which means no new features or major changes are allowed at this stage.
* Only critical bug fixes and minor optimizations are permitted.
* This ensures stability and avoids introducing last-minute errors.

2. Final Testing

* Conduct a final round of comprehensive testing to ensure the application meets all requirements:
* Unit Testing: Validate that individual functions and components work correctly.
* Integration Testing: Verify that different parts of the system interact seamlessly.
* End-to-End Testing: Test the system as a whole to ensure workflows function as expected.
* Performance Testing: Measure response times, load handling, and system efficiency.
* Security Testing: Ensure the system is secure and free from vulnerabilities.

Example:

Check endpoints for correct functionality (e.g., creating and deleting clients).

Test edge cases like invalid input, missing fields, or unexpected user behavior.

3. Review and Approval

* Conduct a code review to verify that all code adheres to established standards.
* Check for proper code structure, clean syntax, and meaningful comments.
* Ensure all linting rules and formatting guidelines are followed.
* Obtain approval from developers or clients.

Checklist for Approval:

* All features are implemented and tested.
* No critical bugs remain unresolved.
* Documentation is complete and accurate.

Share the finalized version with all end-users, or the development team:

Publish updates via emails, project management tools, or communication platforms

Announce the version publicly if applicable (e.g., on GitHub).

**9. Client review and feedback**

This stage ensures that the delivered product aligns with client expectations, meets their needs, and functions as intended in real-world scenarios. Any necessary refinements or adjustments can be identified and prioritized for future updates or patches.

1. Client Demonstration

* Conduct a formal presentation or demonstration of the final product.
* Walk the client through the key features, functionality, and overall workflows.
* Highlight how the product meets their requirements and solves their pain points.

Example:

Walk the client through user registration, client management, and reservation workflows.

Showcase the system's performance, ease of use, and reliability.

2. Feedback Collection

Gather client feedback systematically using:

* Feedback Forms: Structured questions on usability, performance, and satisfaction.
* Interviews: Direct conversations to understand the client’s experience.
* Surveys: Use tools like Google Forms or Typeform for quick and organized feedback.
* Bug Reporting Tools: Platforms like Trello, Jira, or GitHub Issues can help track reported issues.

Example Questions for Feedback:

* How satisfied are you with the product?
* Are there any specific areas where improvement is needed?
* Did you encounter any functionality or performance issues?

**10. Maintenance and enhancement**

1. Monitoring and Bug Fixing

* System Monitoring: Continuously monitor the application's performance, uptime, and overall health using tools like Prometheus, New Relic, or Sentry.
* Bug Resolution: Fix any issues or bugs that were missed during testing or arise after real-world usage.
* Log reported issues in tools like Jira or GitHub Issues.
* Prioritize fixes based on urgency and impact.

Examples:

Address unexpected crashes or API errors.

Fix data inconsistencies (e.g., incorrect client reservations).

2. Performance Optimization

* Optimization: Analyze and improve application performance to ensure scalability and responsiveness.
* Optimize database queries.
* Reduce API response times.
* Improve resource utilization (e.g., memory, CPU).
* Monitoring Tools: Use performance tools like Postman for API performance or Lighthouse for front-end analysis.

Example Tasks:

Improve the reservation system’s response time when fetching large data sets.

Optimize database queries to avoid delays during client retrieval.

3. Security Updates and Patches

Stay proactive in addressing potential security vulnerabilities by:

* Updating dependencies and libraries to their latest, stable versions.
* Applying security patches to prevent vulnerabilities like SQL Injection or Cross-Site Scripting (XSS).
* Ensuring secure authentication mechanisms and data encryption protocols.

Tools: Use vulnerability scanners like OWASP ZAP or NPM Audit for Node.js projects.

Example:

Update express or swagger-ui-express dependencies if security vulnerabilities are reported.

4. Client Support and Training

Offer ongoing support to the client or users:

* Troubleshoot issues encountered during usage.
* Provide user guides, FAQs, or training sessions to help clients use the system effectively.
* Establish clear communication channels for support requests.

Use tools like email, Slack, or ticketing systems for tracking client issues.

Example:

Support the client when they need assistance creating new reservations or adding tables.

5. System Enhancements

* Continuous Improvement: Based on feedback, introduce new features or enhancements to improve functionality.
* These updates may align with changing business requirements or emerging technology trends.
* Plan incremental releases or updates (e.g., v1.1, v1.2).

Examples of enhancements include:

Adding search or filter functionalities for tables or reservations.

Improving the user interface for better usability.

Expanding features such as email notifications for reservation confirmations.

Tools: Use version control tools like Git for managing enhancements.