WEB APPLICATION TECHNICAL DOCUMENT: HARVEST 4 FREE

TABLE OF CONTENTS

1. INTRODUCTION

The **Harvest 4 Free** project aims to educate users about growing vegetables—especially garlic—and promotes healthy living through gardening tips, recipes, and pest control methods. This document is designed to guide developers, product managers, and stakeholders through the technical specifics of the project. The purpose is to outline the structure of the web application, key technical components, and interaction mechanisms between various parts of the system.

The audience for this document includes both technical and non-technical stakeholders. Developers and designers will find details on the technical architecture, database design, and development workflow. Product managers and stakeholders will gain insight into how the system supports user interaction, its scalability, and its long-term viability.

2. ARCHITECTURE

2.1 SYSTEM OVERVIEW

The application employs a **client-server architecture** where the frontend (client-side) interfaces directly with users, and the backend (server-side) processes data and user requests. The separation of concerns between frontend and backend allows the system to handle multiple users concurrently without performance bottlenecks. The client side uses **HTML, CSS, and JavaScript**, which provides users with an interactive, easy-to-navigate platform, while the server-side **JSP** handles form submissions and database interactions.

2.2 TECHNOLOGICAL STACK

- **Frontend**: The structure is built with **HTML5** for content, **CSS** for layout and design, and **JavaScript** to manage dynamic behaviours like the slideshow and form validation.
- **Backend**: **Java Server Pages (JSP)** is the backbone of the server-side logic, handling requests and generating dynamic responses based on user input. JSP's ability to integrate seamlessly with Java makes it a reliable choice for creating dynamic web applications.
- **Database**: **PostgreSQL** is chosen for its advanced capabilities such as complex queries, indexing, and reliability. It is an object-relational database system that supports ACID properties, ensuring transactional integrity.

2.3 EXTERNAL APIS

Although no external APIs are listed in the current scope, this section provides a placeholder for potential future integrations with APIs, such as weather data APIs for gardening recommendations or recipe databases.

3. FUNCTIONAL REQUIREMENTS

3.1 USER STORIES

The application's core functionality is modelled around user stories that reflect real-world usage:

- **User Story 1**: As a user, I can view information about growing garlic and other vegetables.
- **User Story 2**: I can browse a collection of recipes using vegetables promoted on the site.
- **User Story 3**: I can submit my contact details via a form for further interaction.

3.2 FEATURES

- **Home Page**: The homepage displays a **slideshow** that cycles through different vegetable images. It also provides navigation links to subsections such as **Recipes**, **Health Benefits**, and **Pest Control**.
- **Contact Form**: A user-friendly form where visitors can submit their **name**, **email**, **phone number**, and **address**. This data is securely stored in the PostgreSQL database.
- **Clients Section**: The clients' section highlights collaborators by displaying their logos and linking to their websites.
- **Tutorial Links**: This feature provides links to YouTube videos, offering **step-by-step tutorials** for users to learn effective gardening techniques.

4. NON-FUNCTIONAL REQUIREMENTS

4.1 PERFORMANCE

Performance is a key requirement. The website must load within **3 seconds** for users on standard broadband connections to ensure a smooth user experience. Any delays may discourage users from interacting with the site.

4.2 SECURITY

Security measures, including **SSL encryption**, are employed to secure sensitive user information (such as contact details) submitted through the forms. In the future, **OAuth** or another authentication mechanism could be added to enhance user privacy and account management.

4.3 SCALABILITY AND AVAILABILITY

The system should be capable of scaling horizontally by adding additional servers as needed. It must be designed to handle spikes in traffic without performance degradation, ensuring a **99.9% uptime** for high availability.

5. DATABASE DESIGN

5.1 TABLES

The database is structured with the following key tables:

- **Clients Table**: This stores information about collaborating clients, including their **client_id**, **client_name**, **logo_url**, and a **timestamp** for when the client record was created.
- **Contact Submissions Table**: This holds data from user form submissions, including **submission_id**, **full_name**, **email**, **phone_number**, and **submitted_at** timestamp.

5.2 DATA FLOW

The flow of data begins when a user submits their contact details through the form. This data is then stored in the **PostgreSQL** database, ensuring it is securely handled for future administrative review.

5.3 POSTGRESQL SETUP

The PostgreSQL database requires setup on the server, utilizing **connection pools** to manage multiple database requests. **Indexes** are employed to optimize queries, particularly for frequently accessed columns.

6. API DOCUMENTATION

Though the current version of the application does not include APIs, this section remains open for future expansion. Potential API functionalities could include interacting with external gardening data or fetching up-to-date recipes from third-party services.

---

7. FRONTEND DESIGN

7.1 HTML/CSS COMPONENTS

The frontend design utilizes **semantic HTML5** for the structural content, while **CSS** is used for layout and responsiveness. **Media queries** are included to ensure the application is responsive on different screen sizes.

7.2 KEY COMPONENTS

- **Slideshow**: The homepage features a dynamic slideshow of vegetable images, using **JavaScript** to handle transitions.
- **Planting Instructions**: A detailed guide on garlic cultivation, presented in a step-by-step format for users to follow.

7.3 RESPONSIVENESS

The design ensures compatibility across devices (desktop, tablet, mobile) using **CSS media queries** to adjust layouts, making it accessible to a broader audience.

---

8. DEVELOPMENT WORKFLOW

8.1 VERSION CONTROL

The project is managed using **Git**. The **main** branch contains the production-ready code, while the **dev** branch is for ongoing development. This branching strategy ensures stability while enabling agile development.

8.2 ENVIRONMENT SETUP

For development, a local environment includes **Apache Netbeans** for running JSP pages and **PostgreSQL** for database operations. Developers must configure the environment with appropriate connection strings and credentials.

9. TESTING STRATEGY

9.1 UNIT TESTS

Unit tests are conducted on individual components, such as validating user input in forms and ensuring the database connection functions properly.

9.2 INTEGRATION TESTS

Integration testing ensures that the flow of data from the form to the database is seamless, and administrators can retrieve and review submissions as intended.

9.3 END-TO-END TESTING

End-to-end testing simulates the full user journey, from accessing the homepage to successfully submitting a form. This tests the system's ability to process and handle user data efficiently.

10. DEPLOYMENT & MAINTENANCE

10.1 ENVIRONMENT SETUP

The production environment involves deploying the application on **AWS** with **PostgreSQL** hosted on **AWS RDS** for scalable and secure data management.

10.2 DEPLOYMENT PIPELINE

A **CI/CD pipeline** is implemented to automate testing and deployment processes. Each code push to the repository triggers a build and test cycle, ensuring only well-tested code makes it to production.

10.3 MONITORING & LOGGING

The system will employ logging mechanisms for database operations, including successful form submissions and errors, using monitoring tools such as **Prometheus** or **AWS CloudWatch**.

11. SECURITY CONSIDERATIONS

11.1 AUTHENTICATION & AUTHORIZATION

Admin access will be restricted through secure login mechanisms, such as **OAuth**, to ensure only authorized personnel can access sensitive data.

11.2 DATA PROTECTION

Sensitive user data is encrypted using **AES encryption** before being stored in the database. This protects users' personal information from unauthorized access.

11.3 DATABASE SECURITY

PostgreSQL is configured with **secure access controls**, and all database connections are encrypted to ensure secure data transmission.

---

12. APPENDICES

12.1 GLOSSARY

- **JSP**: Java Server Pages, used for creating dynamic web content with Java.
- **PostgreSQL**: An open-source relational database management system.

12.2 REFERENCES

- **PostgreSQL Documentation**: https://www.postgresql.org/docs/
- **HTML Documentation**: https://developer.mozilla.org/en-US/docs/Web/HTML
- **Java JSP Documentation**: https://docs.oracle.com/javase/tutorial/jsp/