Software Design Specifications

for

Sustainable Gardening Companion

Prepared by:

Team Agri Guide

Document Information

Title:			
Project Manager:		Document Version No:	
		Document Version Date:	
Prepared By:		Preparation Date:	

Version History

Ver. No.	Ver. Date	Revised By	Description	Filename

Table of Contents

1 Introduction

The Software Design Specifications (SDS) for the *Sustainable Gardening Companion* provide a comprehensive technical blueprint for developing a web-based application that encourages and facilitates eco-friendly gardening practices. This document outlines the architectural and detailed design of the system, defining how the software components will be structured and interact with each other to meet the specified requirements.
This section includes the purpose of the SDS, the scope of the design, definitions and acronyms relevant to the project, a list of references used during the design process, and an overview of the document's structure and use.

1.1 Purpose

The purpose of this Software Design Specification is to present a detailed technical description of the design and structure of the Sustainable Gardening Companion web application. This document serves as a guide for developers, testers, UI/UX designers, and stakeholders to understand the internal architecture, components, and data flow that support the system's functionality.
The primary audiences for this document include:

- Development Team – to implement the system as designed.
- Testing Team – to verify that the implementation aligns with the design.
- UI/UX Designers – to understand user interaction points and interface flow.
- Faculty and Reviewers – to evaluate the completeness and feasibility of the design.

The document is structured to provide both high-level architectural overviews and detailed component-level specifications for all major modules of the system.

1.2 Scope

This Software Design Specification applies to the Sustainable Gardening Companion project – a web-based platform designed to assist individuals in practicing environmentally sustainable gardening. The design affects all modules of the application including the AI-powered plant health checker, personalized dashboard, weather-based watering suggestions, educational content module, and community discussion forum.
It provides detailed design considerations for the following:

- *Web application architecture (frontend and backend)*
- *Integration with third-party weather and plant health APIs*
- *User interface design elements and user interaction flow*
- *Data handling, validation, and storage mechanisms*
- *Component modularity and scalability*

1.3 Definitions, Acronyms, and Abbreviations

Term	**Definition**
AI	Artificial Intelligence
API	Application Programming Interface
UI	User Interface
UX	User Experience
SDS	Software Design Specification
SRS	Software Requirements Specification
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
CRUD	Create, Read, Update, Delete
REST	Representational State Transfer

1.4 References

The following documents and sources have been used as references in the creation of this Software Design Specification:

- **Statement of Work (SOW)** – Mahindra University, dated February 7, 2025.
- **Software Requirements Specification (SRS)** – Submitted March 10, 2025.
- **IEEE Std 1016-2009** – IEEE Standard for Information Technology—Systems Design—Software Design Descriptions.
- **Weather and Plant Health APIs** – Referenced from OpenWeatherMap and Plant.id.
- **Web Development Documentation** – MDN Web Docs for HTML, CSS, and JavaScript: https://developer.mozilla.org

2 Use Case View

This section presents the central use cases derived from the functional requirements of the *Sustainable Gardening Companion*. These use cases represent essential and high-impact functionalities, such as user interaction, AI-based plant health detection, weather-based watering recommendations, and community engagement. Each use case involves multiple system components and helps illustrate the core design structure.
The design covers user flows from authentication to interaction with AI tools, viewing dynamic content, and participating in a collaborative gardening community

2.1 Use Case

Use Case 1: User Registration and Login

- **Brief Description**: Enables new users to create an account and existing users to securely log into the system.
- **Usage Steps**:
    1. User accesses the landing page.
    2. User selects either "Register" or "Login".
    3. Enters credentials (username, password) and optional profile data (location, gardening preferences).
    4. System validates credentials and redirects to personalized dashboard.

Use Case 2: AI-Based Plant Health Detection

- **Brief Description**: Allows users to upload plant images and receive AI-driven diagnostics regarding plant health.
- **Usage Steps**:
    1. User navigates to the "Plant Health Checker" section.
    2. Uploads an image of their plant.
    3. System processes the image using an integrated AI model.
    4. Displays results including diagnosis, suggested actions, and potential causes.

Use Case 3: View Personalized Dashboard

- **Brief Description**: Displays user-specific gardening tips, weather updates, and progress tracking.
- **Usage Steps**:
    1. Upon login, the user is directed to the dashboard.
    2. Dashboard fetches data from user history, API results, and stored preferences.
    3. Displays widgets like current weather, watering alerts, recent uploads, and recommended guides.

Use Case 4: Manage Watering Schedule

- **Brief Description**: Uses real-time weather data to provide optimal watering recommendations for the user's garden.

- **Usage Steps**:
  1. User accesses the watering schedule feature.
  2. System fetches local weather data using third-party API.
  3. Based on plant types and forecast, system generates a watering schedule.
  4. User receives notifications or views the schedule on the dashboard.

Use Case 5: Participate in Community Forum

- **Brief Description**: Users can discuss topics, ask questions, and share gardening experiences with the community.
- **Usage Steps**:
  1. User clicks on the "Community" tab.
  2. Browses categories or searches for a topic.
  3. Creates a new post or replies to existing ones.
  4. Can upvote or bookmark useful content.

Use Case 6: Access Educational Resources

- **Brief Description**: Provides access to curated gardening guides, tutorials, and sustainability resources.
- **Usage Steps**:
  1. User navigates to the "Learn" section.
  2. Filters resources based on topic, skill level, or plant type.
  3. Views video or article content.
  4. Optionally downloads materials or saves favorites.

Use Case 7: Admin Content Management

- **Brief Description**: Admin users manage the platform’s content including learning materials and forum moderation.
- **Usage Steps**:
  1. Admin logs in through a secure backend panel.
  2. Views content library or user activity.
  3. Adds, edits, or removes educational posts.
  4. Flags or deletes inappropriate forum content.

3 Design Overview

The *Sustainable Gardening Companion* software design is structured using a modular and layered architecture. The system is divided into functional packages (modules) that operate independently but communicate through clearly defined interfaces. The design prioritizes scalability, usability, and integration with external services, especially for AI diagnostics and weather-based insights.

3.1 Design Goals and Constraints

Design Goals	Description
Modularity	Ensure individual components (AI, UI, forum, weather scheduling) are independently manageable and scalable.
Responsiveness	Support access across devices (desktop, tablet, mobile).
Real-time Data Integration	Seamless access to weather and AI services.
User Engagement	Personalized dashboards and community interaction.

Design Goals	Description
Security	Secure login, session management, and role-based access control.

Constraints	Description
Academic Schedule	Project deliverables must align with semester milestones.
Third-Party Dependence	Reliant on plant health datasets and weather APIs.
Limited Infrastructure	Backend hosted on free-tier cloud services.
Team Composition	Divided roles among UI/UX, frontend, backend, and testing teams.
Tools & Stack	React.js (frontend), Node.js/Express (backend), MongoDB, OpenWeatherMap API, AI model (Plant.id or custom).

3.2 Design Assumptions

- Users have basic gardening knowledge and access to a device with internet connectivity.
- AI image classification APIs or models are pre-trained and ready to be integrated.
- Weather API services are accessible and provide accurate, location-based forecasts.
- The number of concurrent users will be moderate (as expected for a university demo).
- Educational content is primarily static and updated periodically by admins.
- Community moderation will be semi-automated or managed by Admin users.

3.3 Significant Design Packages

The application is organized into layered and hierarchical modules as follows:

Package	Sub-packages / Modules	Responsibility
ui	dashboard, forum, resources, auth	Handles user interaction and layout logic
services	authService, aiService, weatherService, userService	Business logic and data processing
controllers	userController, plantController, forumController	Orchestrates requests and response logic
models	User, Post, Plant, Resource	Database schema definitions
utils	validator, logger, scheduler	Utility and helper functions
external	weatherAPI, plantHealthAPI	Third-party API interaction layer

These packages ensure clear separation of concerns and maintainability.

3.4 Dependent External Interfaces

The application relies on several external interfaces for delivering core functionality.

External Application and Interface Name	**Module Using the Interface**	**Functionality / Description**
OpenWeatherMap API	weatherService	Retrieves real-time weather forecasts used to calculate optimal watering schedules based on location.
Plant.id API (or Custom AI)	aiService	Processes uploaded plant images and returns disease diagnosis or health analysis.
Firebase Auth / JWT	authService	Manages secure user authentication and session control.
MongoDB Atlas	userService, forumController	Stores user data, forum posts, resource metadata, and system logs.

3.5 Implemented Application External Interfaces (and SOA web services)

The application also exposes several public interfaces to support internal and future integrations.

Interface Name	**Module Implementing the Interface**	**Functionality / Description**
/api/auth/login, /register	authController	Handles user registration and login via POST methods, validates credentials, and returns tokens.
/api/plant/diagnose	plantController	Accepts plant images, sends to AI service, returns

Interface Name	Module Implementing the Interface	Functionality / Description
		diagnostic report.
/api/weather/schedule	weatherController	Fetches weather data and returns a JSON object with a personalized watering schedule.
/api/forum/posts	forumController	Manages CRUD operations for community posts, comments, and user interactions.
/api/resources	resourceController	Provides structured educational material to frontend modules.

4 Logical View

This section provides a layered and modular view of the internal design of the *Sustainable Gardening Companion* system. It breaks down the application into modules and then into collaborating classes that realize specific functionalities outlined in Section 2 (Use Case View). The Logical View also depicts the interactions among classes and modules using diagrams and descriptions.
The design follows the Model-View-Controller (MVC) pattern, with clear separation between data, logic, and user interface. Each layer progressively handles system responsibilities, allowing scalability, maintainability, and enhanced readability.

4.1 Design Model
The software is decomposed into the following major modules, each composed of significant classes:
Module: Auth Module
- **Class:** AuthController
  - Responsibilities: Handle registration, login, token issuance
  - Methods: registerUser(), loginUser(), validateToken()
- **Class:** AuthService
  - Responsibilities: Password hashing, JWT creation
  - Methods: hashPassword(), generateToken(), verifyToken()

Module: Plant Health Diagnosis
- **Class:** PlantController
  - Responsibilities: Receives plant image, sends to AI service
  - Methods: uploadPlantImage(), getDiagnosis()
- **Class:** AIService
  - Responsibilities: Handles API interaction with plant ID service
  - Methods: sendToAIModel(), parseResult()

Module: Weather Schedule
- **Class:** WeatherController
  - Responsibilities: Fetch local weather and generate schedule
  - Methods: fetchWeatherData(), generateSchedule()
- **Class:** WeatherService
  - Responsibilities: External API communication
  - Methods: callWeatherAPI(), calculateOptimalWateringTimes()

Module: Forum Module
- **Class:** ForumController
  - Responsibilities: Manage posts, comments, votes
  - Methods: createPost(), addComment(), upvotePost()
- **Class:** ForumService
  - Responsibilities: Business logic and data processing
  - Methods: filterPosts(), validatePostContent()

Module: Dashboard & User Profile
- **Class:** DashboardController
  - Responsibilities: Load personalized widgets
  - Methods: loadUserData(), fetchWidgets()
- **Class: UserModel**

    - Attributes: name, email, plantHistory, location
    - Responsibilities: Stores all user-related data

I can generate a UML Class Diagram image for these classes on request.

4.2 Use Case Realization
Let’s describe how the core use cases from Section 2 are realized, both at the module level and the class level.

Use Case: AI-Based Plant Health Detection
High-Level Interaction (Module-Level):
plaintext
CopyEdit
User → UI → PlantController → AIService → Plant.id API → AIService → PlantController → UI
Sequence Diagram (Textual Description):

1. User uploads plant image via UI.
2. PlantController receives image and forwards it to AIService.
3. AIService sends request to external Plant.id API.
4. Response is parsed and diagnosis is returned.
5. UI displays results to the user.

Internal Class Collaboration:

- PlantController.uploadPlantImage() validates input and calls AIService.sendToAIModel().
- AIService.sendToAIModel() sends a POST request with the image.
- AIService.parseResult() converts API response to a user-readable format.

Use Case: View Personalized Dashboard
High-Level Interaction (Module-Level):
plaintext
CopyEdit
User → Dashboard UI → DashboardController → UserModel + WeatherService + AIService
Activity Flow:

1. User logs in and is redirected to dashboard.
2. DashboardController collects:
    - Recent AI scans (from AIService)
    - Watering schedules (from WeatherService)
    - User data (from UserModel)
3. Dashboard renders the widgets dynamically.

Use Case: Participate in Community Forum
High-Level Interaction:
plaintext
CopyEdit
User → Forum UI → ForumController → ForumService → Database
Sequence:

1. User creates a new post.
2. ForumController.createPost() calls ForumService.validatePostContent().
3. Validated content is saved to the database.
4. UI updates forum feed.

Use Case: Admin Content Management
High-Level Interaction:
plaintext
CopyEdit
Admin → Admin UI → ResourceController → ResourceService → Database
Operations:

- Add/edit/delete resources and forum posts
- Manage user-reported content
- Validate inputs and return status to UI

---

5 Data View

This section describes the persistent data storage perspective of the *Sustainable Gardening Companion* system. The system relies on a document-based database (e.g., MongoDB) to store and retrieve structured and semi-structured data. Persistent storage is required for user accounts, AI diagnosis history, community posts, learning resources, and system logs.

5.1 Domain Model

The domain model represents the key entities that reflect real-world concepts and how they are related. These domain objects are mapped to the system's internal data models and persisted in the database.

Main Entities:

- User
- PlantDiagnosis
- WateringSchedule
- ForumPost
- Comment
- Resource
- Admin

Relationships:

- A User can have many PlantDiagnosis entries.
- A User can have one WateringSchedule.
- A User can create many ForumPosts and Comments.
- An Admin manages Resources and moderates ForumPosts.

I can provide an ER diagram image on request to illustrate this visually.

5.2 Data Model (Persistent Data View)

The data model is implemented using collections and documents in a NoSQL format (assuming MongoDB), allowing flexibility in storing semi-structured data while maintaining relationships through referencing.

Primary Collections:

- users
- diagnoses
- schedules
- forum_posts
- comments
- resources

5.2.1 Data Dictionary

Below is the data dictionary that defines each key entity's attributes, data types, and descriptions.

Table: User

Attribute	**Data Type**	**Description**
userId	ObjectId	Unique identifier
name	String	Full name of the user
email	String	User's email address
passwordHash	String	Encrypted password
location	String	User's region or city
registrationDate	Date	Date of account creation

Attribute	**Data Type**	**Description**
role	String	'user' or 'admin'

Table: PlantDiagnosis

Attribute	Data Type	Description
diagnosisId	ObjectId	Unique ID for diagnosis
userId	ObjectId	Reference to user
imageURL	String	Path or link to uploaded plant image
diagnosisResult	String	Result returned from AI model
timestamp	Date	Date and time of diagnosis

Table: WateringSchedule

Attribute	Data Type	Description
scheduleId	ObjectId	Unique identifier
userId	ObjectId	Reference to user
location	String	Location used for weather data
upcomingSchedule	Array of Objects	Watering time, date, and suggestion
lastUpdated	Date	Last time the schedule was refreshed

Table: ForumPost

Attribute	Data Type	Description
postId	ObjectId	Unique ID for the post
userId	ObjectId	Author of the post
title	String	Title of the discussion
body	String	Content of the post
tags	Array	List of categories or hashtags
createdAt	Date	Post creation timestamp
upvotes	Number	Number of upvotes

Table: Comment

Attribute	Data Type	Description
commentId	ObjectId	Unique comment ID
postId	ObjectId	Reference to ForumPost
userId	ObjectId	Author of the comment
text	String	Comment content
createdAt	Date	Date of comment

Table: Resource

Attribute	Data Type	Description
resourceId	ObjectId	Unique ID for the resource
title	String	Title of the article or guide
description	String	Short summary
type	String	Video, PDF, article, etc.
url	String	Link to the resource

Attribute	Data Type	Description
uploadedBy	ObjectId	Admin ID
uploadDate	Date	Date it was added

6 Exception Handling

This section defines how the *Sustainable Gardening Companion* application manages unexpected behaviors and runtime failures. Robust exception handling is vital for ensuring a secure, reliable, and user-friendly system. This includes both client-side and server-side exception management, with appropriate logging and recovery procedures.
The application implements structured exception handling using try-catch blocks, middleware (in case of Node.js/Express backend), and user-friendly error messages in the UI. All critical exceptions are logged and, where needed, escalated to developers or administrators.

6.1 Exception Categories

Exception Type	Module/Context	Cause	Handling Strategy	User Feedback
AuthenticationError	AuthService, AuthController	Invalid login, expired or missing token	Returns HTTP 401, logs attempt	"Invalid credentials" or "Session expired"
ValidationError	ForumController, PlantController, User Input	Missing or malformed input data	Prevent API call, notify user	"Please fill all required fields"
APIConnectionError	WeatherService, AIService	Third-party API is unreachable	Retry logic or fallback, logs error	"Service currently unavailable, try again later"
FileUploadError	PlantController	Incorrect image format, upload failure	Reject and log error, notify user	"Upload failed. Please try a different image"
DatabaseError	UserService, ForumService	Failed CRUD operations	Catch and rollback changes, logs error	"Unexpected error occurred. Please try again"
AuthorizationError	AdminController, Restricted routes	Unauthorized access to admin routes	Return HTTP 403, log user ID	"Access denied. Admin privileges required"
NotFoundError	Any controller	Resource doesn't exist (post, schedule, etc.)	Return HTTP 404 with context	"Requested item not found"
InternalServerError	Global fallback	Uncaught exception or logic failure	Catch in global handler, log full trace	"An error occurred. We're working on it."

6.2 Logging Mechanism

All exceptions are logged using a centralized logging utility (logger.js). This module captures:

- Error type and message
- Stack trace (for debugging)
- Timestamp
- User ID (if available)
- Affected module or API route

Log levels are defined as:

- INFO: General activity (e.g., user logins)
- WARN: Recoverable exceptions or invalid inputs

- ERROR: API failures, database issues
- FATAL: Unrecoverable system-level exceptions

Logs are written to:

- **Console** (for development)
- **Log files** or **cloud logging service** (in production)

6.3 Follow-up Actions

Severity	**System Response**	**Developer/Admin Action**
Low (e.g., input errors)	Message shown to user	None
Medium (e.g., failed API call)	Retry or graceful fallback	Monitor logs
High (e.g., database or auth failure)	Log and alert admin	Investigate root cause
Critical (e.g., crash or security breach)	Log, halt affected module, send email alert	Immediate investigation and hotfix

7 Configurable Parameters

The following table lists the primary configuration parameters used in the *Sustainable Gardening Companion* application. These parameters influence core functionality such as API integration, environment settings, user behavior customization, and feature toggling. Parameters marked as dynamic can be adjusted without requiring a system restart or redeployment.

Configuration Parameter Name	**Definition and Usage**	**Dynamic?**
PORT	Defines the port number on which the backend server runs.	No
MONGO_DB_URI	MongoDB connection string used to connect to the database.	No
JWT_SECRET_KEY	Secret key used to generate and validate authentication tokens.	No
PLANT_AI_API_KEY	Key for accessing the external plant health AI service.	No
WEATHER_API_KEY	API key used to fetch weather data from the OpenWeatherMap API.	No
MAX_UPLOAD_SIZE	Maximum file size (in MB) for uploading plant images.	Yes
SCHEDULE_REFRESH_INTERVAL	Time interval (in hours) for refreshing watering schedules based on updated weather data.	Yes
ENABLE_COMMUNITY_FORUM	Toggle to enable/disable access to the forum section.	Yes
DEFAULT_LOCATION	Default fallback location used when user's location is unavailable.	Yes
ALLOWED_IMAGE_FORMATS	Allowed file types for image uploads (e.g., JPG, PNG).	Yes
LOG_LEVEL	Defines the minimum severity level to log (INFO, WARN, ERROR).	Yes
ADMIN_EMAILS	List of email IDs that are allowed admin access.	Yes

8 Quality of Service

*This section outlines the non-functional design aspects of the Sustainable Gardening Companion system that support operational quality, including **availability**, **security**, **performance**, and **system monitoring and control**. These elements ensure that the system is dependable, safe, responsive, and maintainable in a production-like environment.*

8.1 Application Availability

*The application is designed for **high availability**, with minimal downtime to ensure consistent user access. The design supports availability in the following ways:*

- ***Stateless Backend Services**: Enables easy horizontal scaling and container-based deployment for fault tolerance.*

- ***Auto-Restart via Process Manager**: Tools like PM2 or Docker restart failed services automatically.*
- ***Periodic Maintenance**:*
    - *Non-critical maintenance (e.g., log cleanup, cache purging) scheduled during off-peak hours.*
    - *Feature updates are deployed using blue-green or rolling deployment strategies to avoid downtime.*

Design Elements Supporting Availability

- ***Cloud-based Hosting (e.g., Render, Vercel)**: Ensures automatic failover.*
- ***Rate Limiting**: Prevents abuse and keeps the service stable under unexpected traffic surges.*
- ***Retry Mechanism**: Built into API services to handle temporary network or service disruptions.*

Potential Availability Impacts

- ***Mass Data Operations**: Bulk user imports or image scans may cause temporary slowdowns.*
- ***Third-Party API Downtime**: Temporary unavailability of AI or weather APIs may impact features like plant health checks or watering schedules.*

8.2 Security and Authorization

*The system enforces strict **authentication and authorization** controls aligned with business needs to protect sensitive user and community data.*

Security Mechanisms Implemented

- ***JWT-based Authentication**: All users must log in to access personalized features; tokens are securely signed and verified.*
- ***Role-Based Access Control (RBAC)**:*
    - *Regular users: limited to dashboard, forum, and diagnosis features.*
    - *Admin users: can add/edit resources and moderate forum content.*
- ***Password Security**: All passwords are hashed using bcrypt before storage.*
- ***HTTPS Only**: Communication is encrypted to prevent interception or data leaks.*
- ***Input Validation & Sanitization**: Prevents injection attacks or malformed data from affecting server logic.*

Authorization Management

- *Admins are designated manually through configuration or an admin dashboard interface.*
- *All routes requiring elevated privileges are protected using middleware to verify roles.*

8.3 Load and Performance Implications

The application is designed to meet performance expectations under moderate load, as anticipated for academic and demo use cases, with room for scaling if adoption increases.

Load Projections

Component	***Expected Load***
Plant Health Scans	*~20–50 image uploads/day*
Weather Schedule Queries	*~100 queries/day*
Forum Activity	*~30–50 posts/comments/day*
User Accounts	*~200–300 users*

Performance Considerations

- ***Database Indexing**: Commonly queried fields (user ID, timestamps) are indexed.*
- ***Caching**: Weather results are cached temporarily to reduce API calls.*
- ***Asynchronous Processing**: Image upload and diagnosis are processed asynchronously to maintain UI responsiveness.*
- ***Pagination**: Forum results and resources are paginated to reduce client load and server response time.*

8.4 Monitoring and Control

*The application includes key **monitoring hooks** and **control processes** to ensure real-time visibility and maintain system health.*

Controllable Processes

- ***Scheduled Tasks (CRON jobs)**:*
    - *Refresh watering schedules every 12 hours.*

    - *Clean up expired sessions daily.*
- ***Service Watchers**: PM2 or Docker health checks ensure services restart if they crash.*

Monitored Metrics

Metric	***Purpose***
API response times	*Detect performance bottlenecks*
Login and error rates	*Detect brute-force or auth issues*
Uptime percentage	*Track overall availability*
Resource consumption (RAM/CPU)	*Prevent overload and scaling decisions*
Image upload size and frequency	*Monitor usage of AI diagnosis service*

*Monitoring data can be logged or integrated with platforms like **LogRocket**, **New Relic**, or **Grafana** if scaled to production.*
