# **Software Requirements Document (SRD) for Code Me**

## **Table of Contents**

1. **Introduction**
   * 1.1 Purpose
   * 1.2 Scope
   * 1.3 Definitions, Acronyms, and Abbreviations
2. **Overall Description**
   * 2.1 Product Perspective
   * 2.2 Product Features
3. **System Features**
   * 3.1 User Interface (UI)
   * 3.2 Backend Features
   * 3.3 Technical Specifications
4. **External Interface Requirements**
   * 4.1 User Interfaces
   * 4.2 Hardware Interfaces
   * 4.3 Software Interfaces
5. **Non-Functional Requirements**
   * 5.1 Performance Requirements
   * 5.2 Security Requirements
   * 5.3 Usability Requirements
   * 5.4 Availability Requirements
6. **System Architecture and Design**
   * 6.1 System Architecture
7. **Glossary**
8. **Appendix**
   * 8.1 Technologies Used
9. **Revision History**

## **1. Introduction**

### **1.1 Purpose**

The purpose of the **Code Me** project is to create an interactive website that teaches coding, starting with Python. The website will guide users through progressively harder lessons, from beginner to advanced topics, while providing them with hands-on exercises and challenges.

### **1.2 Scope**

The **Code Me** platform will include:

* Interactive coding lessons starting with Python.
* A simple user interface (UI) for navigation through lessons.
* Progressive difficulty levels in lessons.
* Code submission and feedback features.
* A backend for managing user progress and storing lessons.

The system will be built using Python (Flask) for the backend, with HTML, CSS, and JavaScript for the frontend.

### **1.3 Definitions, Acronyms, and Abbreviations**

* **SRD**: Software Requirements Document
* **Flask**: A micro web framework written in Python.
* **HTML/CSS**: Markup and styling languages for creating web pages.
* **WSL**: Windows Subsystem for Linux, used to run Linux distributions on Windows.
* **GitHub**: A platform for version control using Git.

## **2. Overall Description**

### **2.1 Product Perspective**

The **Code Me** project is an educational web application aimed at beginners and intermediate learners of Python. The system will run in a web browser and be responsive to various devices (desktop, tablet, mobile).

### **2.2 Product Features**

* **Lesson Navigation**: Users can progress through lessons, starting with basic Python syntax and advancing to more complex topics like functions, loops, data structures, and algorithms.
* **Interactive Exercises**: Each lesson will include coding challenges that users can complete directly in their browser.
* **User Accounts (Optional)**: Users can create an account to track their progress and receive personalized lesson recommendations.
* **Progress Tracking**: A backend system will store user progress, so they can resume lessons later.

## **3. System Features**

### **3.1 User Interface (UI)**

* **Homepage**: A clean, easy-to-navigate landing page displaying available lessons and an option to sign up for an account.
* **Lesson Pages**: Each page will include a code editor and an explanation of the lesson content. A button will allow users to submit their solutions for feedback.
* **Progress Dashboard**: Users can view their lesson progress and milestones.

### **3.2 Backend Features**

* **Lesson Management**: The system will store the content of lessons in a database (SQLite or PostgreSQL).
* **User Progress**: The backend will track user progress across lessons (using Flask and a database).
* **Code Submission and Feedback**: Users can submit their code solutions, and the system will provide basic feedback (e.g., error handling, completion status).

### **3.3 Technical Specifications**

* **Frontend**: HTML, CSS, JavaScript for basic UI and interaction.
* **Backend**: Python (Flask), with a simple REST API for handling requests.
* **Database**: SQLite (for simple setup) or PostgreSQL (if scalability is required).
* **Version Control**: Git for version control, with the repository hosted on GitHub.
* **Hosting**: Initially hosted locally for development, later on a cloud provider like Heroku or AWS for production.

## **4. External Interface Requirements**

### **4.1 User Interfaces**

The system will have a web-based interface with the following elements:

* **Main Page**: Displays links to the available coding lessons.
* **Lesson Pages**: Each page will contain explanations, code exercises, and a submission button.
* **Login/Signup** (optional): For user accounts to track progress.

### **4.2 Hardware Interfaces**

The system is accessible through any modern web browser. There are no specific hardware requirements beyond basic devices (PC, laptop, tablet, mobile).

### **4.3 Software Interfaces**

* **Web Browser**: Chrome, Firefox, Edge, Safari.
* **Web Framework**: Flask for backend server.
* **Version Control**: Git and GitHub for code management.

## **5. Non-Functional Requirements**

### **5.1 Performance Requirements**

* The system must load the homepage and lesson pages within **3 seconds**.
* The code editor must support real-time input with minimal lag.
* The backend should be able to handle up to **100 concurrent users** during peak load.

### **5.2 Security Requirements**

* User passwords (if implemented) will be hashed using a strong algorithm like bcrypt.
* HTTPS should be used to secure communication between users and the server.

### **5.3 Usability Requirements**

* The UI should be intuitive and user-friendly, ensuring that even beginners can easily navigate the platform.
* The website should be responsive and functional on both desktop and mobile devices.

### **5.4 Availability Requirements**

* The website should be available 99.9% of the time, excluding planned maintenance periods.

## **6. System Architecture and Design**

### **6.1 System Architecture**

The system will be built on a **Model-View-Controller (MVC)** architecture:

* **Model**: The database will store user information, lesson data, and progress.
* **View**: The HTML/CSS frontend will display the lesson content and the code editor.
* **Controller**: Flask will handle requests, serve content, and manage user interactions.

## **7. Glossary**

* **MVC**: A design pattern used for structuring software into three main components: Model, View, and Controller.
* **Flask**: A web framework for Python.
* **HTML**: HyperText Markup Language, used to structure content on the web.
* **CSS**: Cascading Style Sheets, used to style HTML content.

## **8. Appendix**

### **8.1 Technologies Used**

* **Frontend**: HTML5, CSS3, JavaScript.
* **Backend**: Python (Flask), SQLite or PostgreSQL for the database.
* **Version Control**: Git, GitHub for code hosting and collaboration.

## **9. Revision History**

| **Version** | **Date** | **Author** | **Changes** |
| --- | --- | --- | --- |
| 1.0 | 2024-12-27 | Rodtavius  Smith | Initial version of the Software Requirements Document (SRD) created. |
|  |  |  |  |
|  |  |  |  |