**Chapter Three**

**System Design**

**3.1 Introduction**

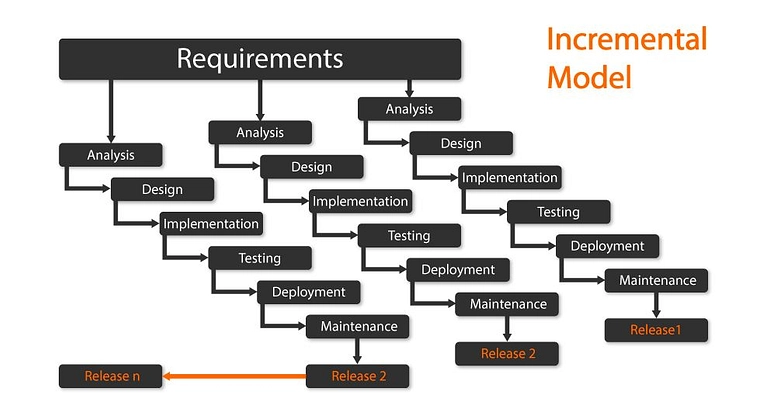
This chapter contains information about the system analysis design segment and design methods. This chapter describes the system to be built and the process model used in the design and implementation of the web-based liver disease detection system. In addition, the various design and development tools that will be used and the requirements for both hardware and software are outlined. Flowcharts, use case diagrams and sequence diagrams are used to show the processes that should take place.

**3.2 Research Design**

The software development life cycle (SDLC) is a conceptual framework that outlines the stages involved in the development of software or systems. SDLC models serve as the backbone for managing large-scale software and IT projects by incorporating software engineering techniques and practices into the system development process. These models adapt to generic framework activities, establishing a workflow that uniquely sequences each activity.

In the analysis and design process, a model is constructed to gain a deeper understanding of the system's requirements and to determine the most effective approach to fulfill them. For this research, the Incremental Development model is employed because it allows for partial implementation of the system requirements in increments, accommodating changes and refinements as the system is developed. The incremental model is characterized by its flexibility and responsiveness to evolving requirements, which is particularly beneficial when the full scope of the system cannot be completely defined from the outset.

Each increment in the incremental model is a slice of the complete system functionality. The development process is divided into smaller, more manageable segments, with each one expanding upon the previous increment. This approach allows for portions of the system to be delivered incrementally, ensuring that each release builds upon the last to evolve the full system. A review is conducted at the end of each increment to ensure that it meets its specified goals and is aligned with the evolving system requirements. The incremental model supports iterative development and frequent reassessment, which can be particularly advantageous when dealing with complex systems where requirements may change or be refined over time. The processes involved in incremental development are depicted in the subsequent sections.

**Figure 3.1 Incremental model (Amninder S., 2018)**

1. **Requirement Analysis and Specification:** The purpose requirements analysis and specification phase is to completely comprehend and describe the customer's exact requirements for the application. This phase is divided into two parts: requirements gathering and analysis, and requirements specification, as shown below.
   1. **Requirement Gathering and Analysis:** The requirement gathering activity is undertaken to obtain all necessary information about the product that is intended to be developed from the customer in order the ensure that the product satisfies the customer’s needs. The analytical activity begins after the requirements have been gathered. The analysis is carried out to identify requirement gaps and inconsistencies in the application
   2. **Requirement Specification:** A software requirement specification (SRS) document is created from the customer requirements established during the requirements gathering and analysis phase. Functional requirements, non-functional requirements, and implementation goals are all contained in this document.
2. **Design:** In the design phase we achieved our goal by turning the requirements specified in the Software Requirement Specification (SRS) document into a structure that was later implemented to develop the system. In technical words, the software architecture was developed from the SRS document during the design process.
3. **Coding:** The coding phase of software development was responsible for translating the software design into source code. The coding phase is also known as the implementation phase since it is during this step that the design was turned into a usable solution. Each design element was implemented as a separate program module. After the code was finished, each module was unit-tested to ensure that all of the separate modules were working properly.
4. **Testing:** During this phase, the various modules were integrated. The plan outlined the sequence in which modules were assembled to complete the system. The integration of several components was done in stages over a period of time. The partially integrated system was tested after each integration phase. Finally, a system test was conducted to ensure the system was operational with all modules successfully merged and tested and to confirm that the developed system met the SRS document's requirements.
5. **Deployment**: The deployment phase is the culmination of the software development process where the software product is transferred from development to production, making it available to users.
6. **Maintenance:** Monitoring and optimizing system performance and enhancing system services are all the steps that were taken in the maintenance phase.

**3.2.1 System Requirements**

For the Web-Based Liver Cirrhosis Detection System, system requirements define the functionalities and capabilities that the system must offer when fully deployed. These requirements are derived from the user needs and expectations of the system:

**3.2.2 User Requirements**

User requirements describe the functionalities and activities that end-users of the web-based liver disease detection system must be able to perform:

i. Registration/Login: Users must be able to register for an account and log in securely.

ii. Upload Medical Imaging Data: Users must be able to upload cell imaging data

iii. Download Prediction Results: Users shall be able to download the results generated by the machine learning model

iv. Progression of disease: Users should be able to track the progression of the liver diseases

v. Logout: Users should have the ability to log out of the system securely.

These requirements are essential in guiding the development of the system, ensuring that it meets the needs of its users and facilitates efficient disease classification.

**3.3 Functional and Non-Functional Requirements**

In the development of the web-based liver disease classification system, a clear distinction between functional and non-functional requirements is crucial. Functional requirements outline the specific behaviors and functions the system must exhibit, such as user management among other things. These are the core actions and processes that define the system's capabilities and how it interacts with its users.

On the other hand, non-functional requirements focus on the system's overall qualities and constraints, such as scalability, security, performance, reliability, and usability. These requirements are not about what the system does, but how it does it, ensuring the system is efficient, secure, and user-friendly. Together, these functional and non-functional requirements form the backbone of the system's design and development, ensuring it meets the needs of its users and operates effectively in its intended environment.

**3.3.1 Functional Requirements**

Functional requirements specify the operations and activities that the web-based liver disease detection system must be able to perform:

The system should allow users to upload medical data, including lab results and patient information.

The system should process the uploaded data using a machine learning model to detect the presence of liver disease.

The system should display the detection results to the user, indicating the likelihood of liver disease and any relevant recommendations.

The system should allow users to download the detection results for further analysis or consultation with healthcare professionals.

**3.3.2 Non-Functional Requirements**

Non-functional requirements define the quality attributes, performance measures, and standards the system must meet. These include:

1. Security: With the handling of sensitive personal and medical data, the system must incorporate advanced security protocols, including data encryption and secure authentication processes.

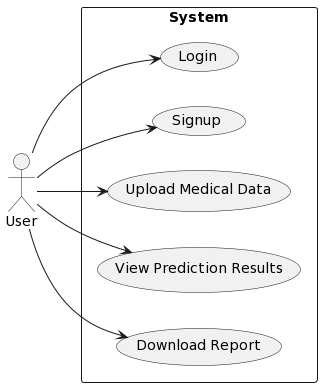
2. Scalability: The system should be designed to accommodate an increasing number of users and data volume, ensuring consistent performance and responsiveness.

3. Reliability: It is crucial that the system operates reliably, maintaining high accuracy and availability, especially in emergency scenarios where timely access to blood resources is critical.

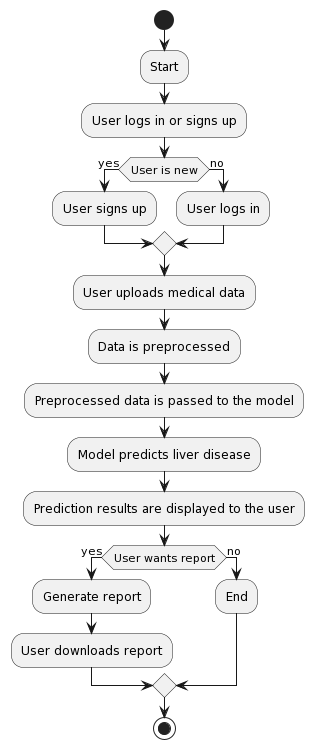
4. User-Friendly Interface: The design of the mobile application should prioritize user-friendliness, with an intuitive interface that caters to users of varying technical backgrounds.

5. Performance: Optimal performance is essential, particularly in terms of fast response times for search queries and real-time data updates.

6. Compatibility: The system must be compatible across a wide range of mobile devices and operating systems to ensure broad accessibility.

****

**Figure 3.2 Use Case Diagram**

****

**Figure 3.3 Flowchart Of the Proposed System**

**3.4 Development Tools**

The development of the web-based liver disease detection system utilized a range of specialized tools, each contributing uniquely to the project's success:

Visual Studio Code: Visual Studio Code (VS Code) is a popular source-code editor developed by Microsoft for Windows, Linux, and macOS. It supports debugging, syntax highlighting, code completion, refactoring, and Git integration. It includes built-in source control features and allows users to customize settings for each document.

Python: Python is a high-level, interpreted programming language known for its simplicity and readability. Python's syntax allows programmers to express concepts in fewer lines of code compared to languages like C++ or Java. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python has a large standard library that provides support for tasks such as file I/O, networking, and web development. It is widely used in various fields, including web development, data science, artificial intelligence, and automation. Python's community is known for its inclusivity and supportiveness, with a wealth of documentation and resources available online. The language is dynamically typed and has automatic memory management, making it beginner-friendly and easy to learn.

PyTorch: PyTorch is a popular machine learning library developed by Meta AI, now part of the Linux Foundation. It is used for tasks like computer vision and natural language processing and is known for its ease of use and flexibility. PyTorch is open-source and released under the BSD License. It provides tensor computing similar to NumPy, with GPU acceleration, and supports deep neural networks with automatic differentiation.

FastAPI: FastAPI is a modern Python web framework designed for building fast and efficient RESTful APIs. It was first released in 2018 and has quickly become popular among developers for its speed, ease of use, and robustness. FastAPI is built on Pydantic and uses Python's standard type hints for data validation, serialization, and deserialization. One of its key features is automatic generation of OpenAPI documentation for APIs created with it.

FastAPI fully supports asynchronous programming, making it suitable for high-performance applications. It can be run on servers like Gunicorn and ASGI servers such as Uvicorn and Hypercorn, making it well-suited for production environments. The framework also focuses on developer-friendliness, with strong editor support and a clear, concise syntax.

React: React.js is a popular JavaScript library for building user interfaces, particularly for single-page applications. It was developed by Facebook and released in 2013. React uses a component-based architecture, allowing developers to create reusable UI components. It uses a virtual DOM to improve performance by updating only the necessary parts of the actual DOM. React supports both client-side and server-side rendering. It uses JSX, a syntax extension that allows embedding HTML within JavaScript code. React's one-way data flow and unidirectional data binding make it easier to manage and update state. React's ecosystem includes tools like React Router for routing and Redux for state management. React Native allows for mobile app development using React principles. The React community is active, with a large number of open-source libraries and components available. It is maintained by Facebook and a community of developers.

Node: Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It allows developers to run JavaScript code outside the browser, making it ideal for server-side development. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, suitable for building fast and scalable network applications. It has a large ecosystem of packages available through npm, the Node.js package manager. Node.js is cross-platform, supporting Windows, macOS, and Linux. It is commonly used for building web servers, API servers, and real-time applications. Node.js applications are written in JavaScript and can be run on the command line using the Node.js interpreter. Node.js has a built-in module system that allows for easy integration of third-party modules. It supports modern JavaScript features and provides a rich set of APIs for network communication, file system operations, and more. Node.js has a vibrant community and is actively maintained by the Node.js Foundation.