Low-Level Design (LLD) for Restaurant Rating Prediction System

By Ravi Kumar M

Table of Contents

Introduction

In the context of the Restaurant Rating Prediction System, the LLD outlines the various modules such as the login module, prediction module, data processing module, and user interface module. Each of these modules has specific responsibilities and functions that are essential for the application's overall functionality. For instance, the login module must handle user authentication securely, while the prediction module is responsible for processing input data and providing accurate restaurant ratings based on the trained machine learning model.

The LLD also emphasizes the flow of data between different components, ensuring that developers have a clear understanding of how information will be passed and manipulated throughout the system. This is critical for debugging and maintaining the application, as developers can easily identify potential points of failure or bottlenecks in the data flow.

Additionally, the LLD includes error handling strategies, security considerations, and performance optimization techniques, providing a holistic approach to software development. By anticipating potential issues and planning for them in advance, the LLD helps to create a robust and resilient application.

Introduction

Furthermore, the document acts as a reference for future development and maintenance. As the project evolves, the LLD can be updated to reflect changes in requirements or architecture, ensuring that all team members are aligned and informed about the current state of the project. This is particularly important in collaborative environments where multiple developers are working on different parts of the application.

In summary, the LLD document is an essential resource that complements the HLD by providing detailed implementation specifics necessary for building a successful software project. It empowers developers with clear instructions and guidelines, promotes effective collaboration, and serves as a foundational tool for maintaining the system throughout its lifecycle. By meticulously detailing each module's functions, data interactions, and error handling, the LLD ultimately contributes to the overall quality and reliability of the final product.

Abstraction

Abstraction in Low-Level Design (LLD) is a foundational principle that focuses on simplifying the complexities of a software system by breaking it down into manageable and coherent modules. This design approach allows developers to concentrate on the essential aspects of each module, emphasizing what each component does rather than how it performs its tasks. By encapsulating the intricate details of functionality within defined boundaries, abstraction helps to create a clear and organized framework for software development.

In an LLD document, abstraction is achieved by delineating the responsibilities of various components and specifying their interactions without delving into the specific implementation details. Each module is described in terms of its inputs, outputs, and functionalities, enabling developers to understand its purpose within the larger system architecture. For example, a module responsible for user authentication would outline its role in verifying credentials and managing sessions but would not necessarily detail the underlying algorithms or database queries involved in the process.

Abstraction

Purpose:

The primary purpose of abstraction in LLD is to facilitate a more straightforward and efficient development process. By allowing developers to focus on specific components independently, abstraction minimizes the cognitive load associated with understanding the entire system at once. This enables team members to work concurrently on different modules, thereby enhancing productivity and promoting collaboration. Each developer can take ownership of a module, ensuring that their work aligns with the overall system goals while maintaining the flexibility to make design decisions pertinent to their component.

Furthermore, abstraction aids in debugging and troubleshooting by localizing issues within individual modules. When a bug is identified, developers can concentrate on the specific module responsible for the failure without needing to analyze the entire system. This modular approach not only streamlines the debugging process but also leads to cleaner and more maintainable code, as developers can implement fixes or enhancements without fear of inadvertently impacting unrelated parts of the application.

Abstraction

Additionally, abstraction promotes code reusability. Well-defined modules can often be reused across different projects or within different parts of the same application, saving development time and effort. For instance, a module that handles API requests for fetching restaurant data can be reused in various applications requiring similar functionalities, reducing redundancy in code development.

Moreover, abstraction enhances the scalability of the system. As the application grows and evolves, new features or modules can be added with minimal disruption to existing components. Developers can introduce new functionalities by creating additional modules that interact with existing ones through well-defined interfaces. This scalability is essential in a fast-paced development environment, where requirements may change frequently, and new features need to be integrated smoothly.

Abstraction also plays a significant role in documentation. By providing clear descriptions of each module functionality and purpose, the LLD serves as a valuable reference for current and future developers. New team members can quickly familiarize themselves with the system by understanding the responsibilities and interactions of different modules without being overwhelmed by implementation specifics.

Abstraction

Lastly, abstraction in LLD fosters a better understanding of system architecture. By visualizing the system as a collection of well-defined modules, developers can appreciate the overall design and how different components contribute to achieving the project's objectives. This holistic understanding encourages best practices in design and implementation, leading to a more robust and efficient software solution.

In summary, abstraction is a vital concept in Low-Level Design that simplifies complex systems into manageable modules, emphasizing their functionalities over implementation details. This approach enhances collaboration, facilitates debugging, promotes reusability, supports scalability, improves documentation, and fosters a deeper understanding of system architecture. By incorporating abstraction into LLD, development teams can work more efficiently and effectively, ultimately contributing to the success of the software project.

Module Design

Login Module

- **Description: Manages user authentication and session handling.**
- **Key Functions:**
  - **login(): Validates user credentials.**
  - **logout(): Ends user session.**
  - **Data Flow: User inputs credentials, which are validated and processed to allow access to the system.**

Model Design

Prediction Module

- **Description: Handles user input and makes predictions using the trained model.**
- **Key Functions:**
  - **predict_rating(): Processes input features, encodes categorical variables, and returns the predicted rating.**
  - **Data Flow: User inputs data through the form, which is then processed and sent to the ML model for prediction.**

Model Design

Data Processing Module

- **Description: Prepares data for model training and prediction.**
- **Key Functions:**
  - **preprocess_data(): Cleans and encodes data before feeding it into the model.**
  - **load_model(): Loads the trained machine learning model and label encoders.**
  - **Data Flow: Raw data is cleaned, transformed, and used in predictions.**

Model Design

User Interface Module

- **Description: Manages the frontend components of the application.**
- **Key Functions:**
  - **render_login_page(): Displays the login form.**
  - **render_prediction_page(): Shows the prediction form and results.**
  - **Data Flow: User interacts with the UI, which communicates with the backend to display results and receive inputs.**

Data Flow Diagrams

User Authentication Flow

Components:

1. **User: Initiates login with credentials.**
2. **Login Page: Captures user input (username and password).**
3. **Backend Authentication Service:**
    - **Validates credentials.**
    - **Checks against stored hashed passwords.**
4. **Response:**
    - **Success: Redirects to the home page.**
    - **Failure: Displays an error message.**

Flow:

- **Step 1: User enters credentials on the login page.**
- **Step 2: Credentials are sent to the backend authentication service.**
- **Step 3: The service checks the validity of the credentials.**
- **Step 4: Based on the authentication result, the user is either granted access or presented with an error.**

Data Flow Diagrams

Prediction Request Flow

Components:

1. **User: Inputs features for prediction.**
2. **Prediction Form: Captures inputs (e.g., location, restaurant type).**
3. **Prediction Module:**
    - **Processes inputs.**
    - **Calls the machine learning model for prediction.**
4. **Model Output: Returns the predicted rating to the user.**

Flow:

- **Step 1: User fills out the prediction form with relevant features.**
- **Step 2: Inputs are sent to the prediction module.**
- **Step 3: The module processes the input and invokes the machine learning model.**
- **Step 4: The model returns the predicted rating.**
- **Step 5: The prediction result is displayed to the user.**

Data Flow Diagrams

Data Processing Flow

Components:

1. **Raw Data Source: Initial dataset (e.g., CSV file).**
2. **Data Preprocessing Module:**
    - **Cleans data (removes duplicates, handles missing values).**
    - **Applies encoding (label encoding for categorical features).**
3. **Processed Data Output:**
    - **Cleaned and formatted data ready for model training and predictions.**

Flow:

- **Step 1: Raw data is loaded from the source.**
- **Step 2: The data preprocessing module cleans and processes the data.**
- **Step 3: The module applies transformations (e.g., encoding).**
- **Step 4: Processed data is outputted for use in model training or prediction.**

Data Flow Diagrams

Prediction Request Flow

Components:

1. **User: Inputs features for prediction.**
2. **Prediction Form: Captures inputs (e.g., location, restaurant type).**
3. **Prediction Module:**
    - **Processes inputs.**
    - **Calls the machine learning model for prediction.**
4. **Model Output: Returns the predicted rating to the user.**

Flow:

- **Step 1: User fills out the prediction form with relevant features.**
- **Step 2: Inputs are sent to the prediction module.**
- **Step 3: The module processes the input and invokes the machine learning model.**
- **Step 4: The model returns the predicted rating.**
- **Step 5: The prediction result is displayed to the user.**

Thank You