Software Design and Architecture Laboratory Exercises- Homework 2

Architectural design

Conceptual Architecture for Winery Web Application:

Components:

User Interface (UI):

- *Description:* Represents the web interface where users input their location and initiate the search.
- *Special Features:* Includes components for displaying the list of wineries.

Client-Side Logic:

- *Description:* Manages user input, sends requests to the server, and updates the UI.
- *Special Features:* Utilizes a JavaScript framework (React, Angular, or Vue) for dynamic client-side rendering.

Express.js Server:

- *Description:* Backend server handling HTTP requests and responses.
- *Special Features:* Implements API endpoints for communication with the client.

Application Logic:

- *Description:* Contains the core business logic of the application.
- *Special Features:* Processes user requests and manages data flow between components.

Data Gathering:

- *Description:* Component responsible for gathering data from external sources.
- *Special Features:* Utilizes the OpenStreetMap API to fetch the locations of wineries in Macedonia. May include a CSV/Excel data sheet of winery information.

Filter 1 (Open Wineries):

- *Description:* Takes the user's current location and time into account.
- *Special Features:* Filters the list of wineries to return only those that are currently open.

Filter 2 (Proximity):

- *Description:* Takes the output from Filter 1 and the user's location.
- *Special Features:* Arranges the list of wineries based on proximity to the user.

Main Database (Wineries Data):

- *Description:* Centralized storage for winery data.
- *Special Features:* Includes information such as winery name, location, opening hours, etc.

External Services:

- OpenStreetMap API:
  - *Description:* Used by the Data Gathering component to fetch winery locations.

Interactions:

User Interaction:

- *What happens:* Users interact with the UI to input their location and initiate the search.

Client-Server Interaction:

- *What happens:* The client communicates with the server through well-defined API endpoints. RESTful principles or GraphQL can be used to structure API communication.

Data Flow:

- *What happens:* Data flows from the UI to the Express.js Server, which forwards requests to the Application Logic. The Data Gathering component fetches winery locations from OpenStreetMap. The Application Logic processes the data and passes it through Filter 1 and Filter 2. The final list is sent back to the client for display.

Architectural Design:

Pipe-and-Filter Architecture:

- *How it works:* Use Filter 1 and Filter 2 as filters in a pipe-and-filter architecture. Data flows through these filters for processing and transformation.

Architectural Styles:

Layered Architecture:

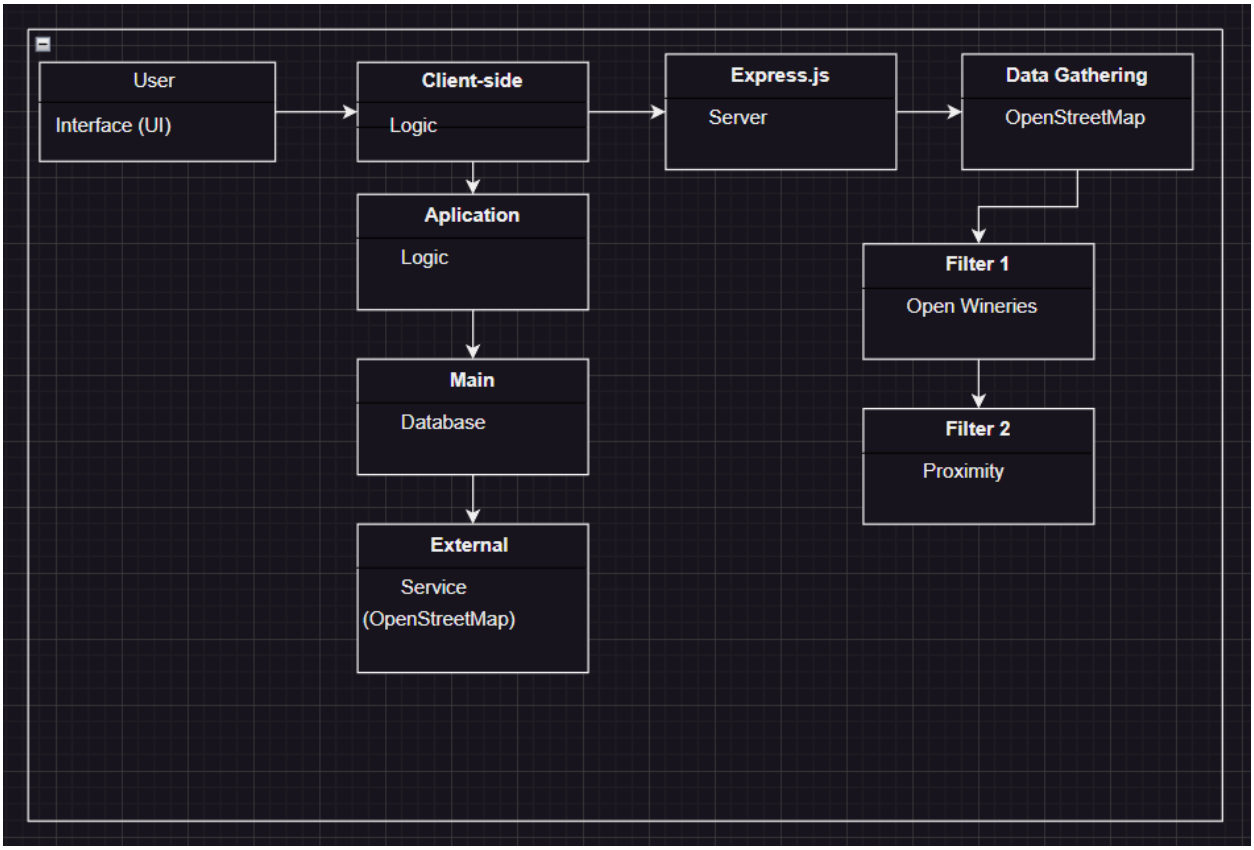
- *How it's utilized:* Utilize a layered architecture with clear separation of concerns. Layers may include Presentation (UI), Application Logic, and Data Storage.

Microservices:

- *Consideration:* Consider adopting a microservices architecture if scalability and modularity are critical. Microservices could handle specific aspects like user management, winery data retrieval, and filtering.

Containerization:

- *Implementation:* Implement containerization using Docker for packaging and deploying application components. Containers ensure consistency across development, testing, and production environments.



Execution Architecture for Winery Web Application:

Components:

User Interface (UI):

- *Description:* The Winery Explorer UI is the portal where users input their current location and initiate a search for open wineries. It provides a user-friendly interface, potentially including a map for visualizing winery locations.

Client-Side Logic:

- *Description:* This component manages user interactions, sending requests to the server, and dynamically updating the UI. Using a JavaScript framework (React, Angular, or Vue), it ensures a responsive and seamless user experience.

Express.js Server:

- *Description:* As the backend powerhouse, the Express.js Server handles HTTP requests and responses. It implements API endpoints for seamless communication with the client, orchestrating the flow of data between components.

Application Logic:

- *Description:* This vital component contains the core business logic of the Winery Explorer. It processes user requests, managing the data flow between components. It's responsible for determining open wineries and organizing them based on proximity to the user.

Data Gathering:

- *Description:* The Data Gathering component is tasked with fetching winery data. Utilizing OpenStreetMap, it retrieves locations of wineries in Macedonia. Additionally, it may integrate a CSV/Excel data sheet for comprehensive winery information.

Filter 1 (Open Wineries):

- *Description:* This filter, implemented in a programming language such as Java, filters wineries based on the user's current location and time. It selectively returns only those wineries that are currently open for visitors.

Filter 2 (Proximity):

- *Description:* Building on the output from Filter 1, Filter 2 arranges the list of open wineries based on their proximity to the user. Employing geospatial algorithms, it ensures the displayed list is ordered by closeness.

Main Database (Wineries Data):

- *Description:* Serving as the centralized storage, the Main Database contains essential winery data like names, locations, and opening hours. It may be populated from the CSV/Excel data sheet, ensuring a comprehensive and up-to-date repository.

External Services:

OpenStreetMap API:

- *Description:* Utilized by the Data Gathering component, the OpenStreetMap API provides geolocation data for wineries in Macedonia.

Interactions:

User Interaction:

- *What happens:* Users interact with the UI, providing their current location to initiate a search for open wineries.

Client-Server Interaction:

- *What happens:* The client communicates with the Express.js Server through well-defined API endpoints, following RESTful principles or GraphQL for efficient data exchange.

Data Flow:

- *What happens:* Data flows seamlessly from the UI to the Express.js Server, initiating a cascade of actions. The Data Gathering component fetches winery locations, and the Application Logic processes the data through Filter 1 and Filter 2. The final, refined list is then sent back to the client for display.

Execution Architecture:

Pipe-and-Filter Architecture:

- *How it works:* Data undergoes transformation through Filter 1 and Filter 2 in a pipe-and-filter architecture, ensuring efficient processing.

Architectural Styles:

Layered Architecture:

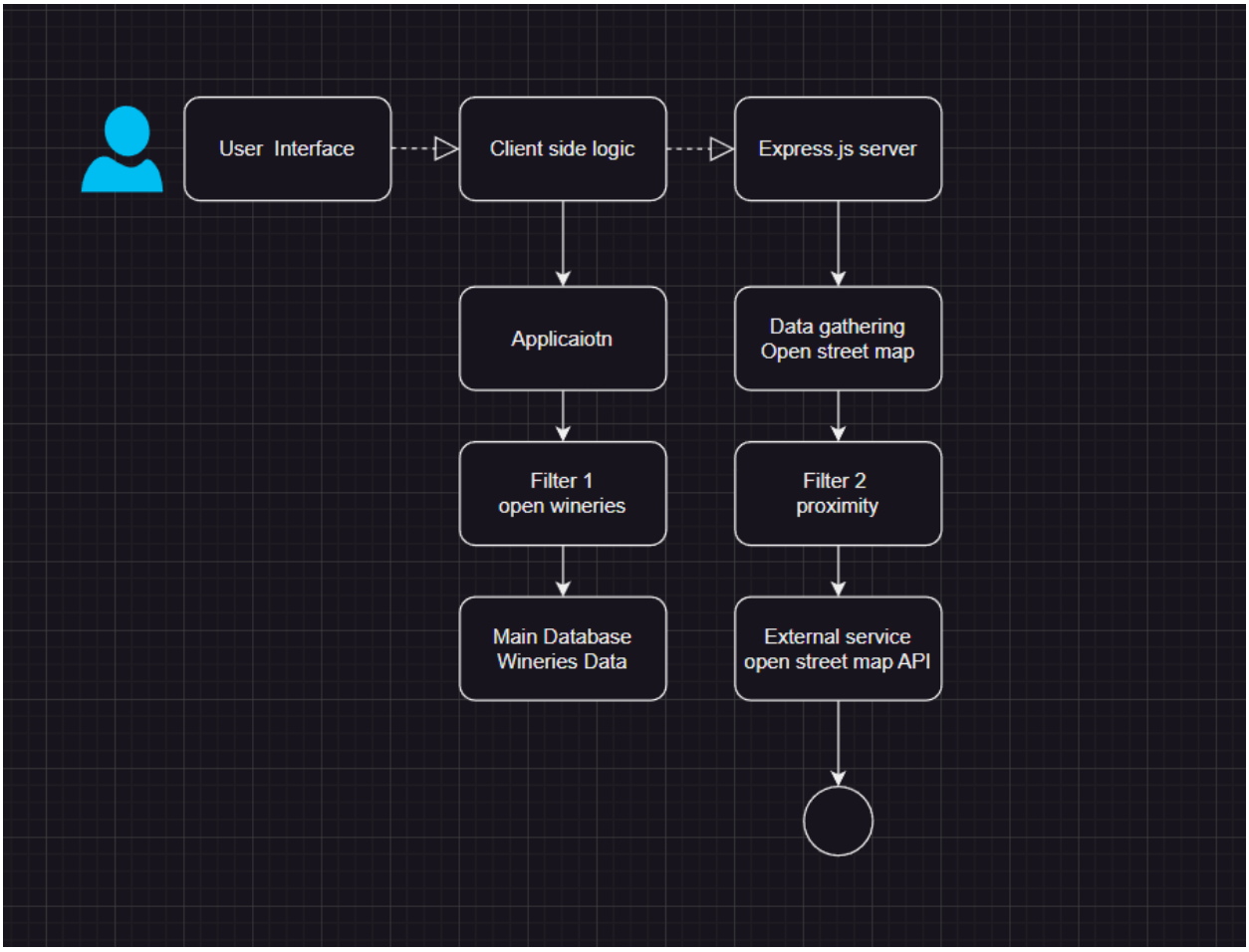
- *How it's utilized:* The Winery Explorer adopts a layered architecture with distinct layers for Presentation (UI), Application Logic, and Data Storage. This ensures a clear separation of concerns.

Microservices:

- *Consideration:* Microservices architecture is considered for scalability and modularity. Specific microservices could handle user management, winery data retrieval, and filtering, enhancing the application's flexibility.

Containerization:

- *Implementation:* Docker is employed for containerization, packaging, and deploying application components. This ensures consistency across various environments, from development to testing and production.



Implementation Architecture for Winery Web Application:

Components:

Frontend Application:

- *Description:* Represents the user interface and client-side logic of the web application.
- *Special Features:* Utilizes a JavaScript framework (e.g., React, Angular, Vue) for dynamic rendering.

Backend Application:

- *Description:* Houses the server-side logic, application, and data processing.
- *Special Features:* Implemented using Node.js with Express.js for handling HTTP requests and responses.

Database Management System (DBMS):

- *Description:* Manages the storage and retrieval of winery data.
- *Special Features:* Can be a relational database (e.g., PostgreSQL) or a NoSQL database (e.g., MongoDB).

OpenStreetMap API:

- *Description:* External service used for fetching geolocation data of wineries.
- *Special Features:* Provides accurate and up-to-date information about winery locations.

Security Module:

- *Description:* Ensures the security of user data and prevents unauthorized access.
- *Special Features:* Manages user authentication, authorization, and data encryption.

Logging and Monitoring Module:

- *Description:* Records important events and monitors application performance.
- *Special Features:* Generates logs, alerts, and reports for system health analysis.

Asynchronous Processing Module:

- *Description:* Handles background tasks asynchronously for improved responsiveness.
- *Special Features:* Utilizes message queues or event-driven architecture for task execution.

Containerization and Orchestration Module:

- *Description:* Manages application deployment and scaling using containerization.
- *Special Features:* Utilizes Docker for containerization and Kubernetes for orchestration.

Load Balancer:

- *Description:* Distributes incoming requests across multiple server instances for load balancing.
- *Special Features:* Enhances application scalability and ensures high availability.

Error Handling and Reporting Module:

- *Description:* Identifies, logs, and reports errors for efficient issue resolution.
- *Special Features:* Implements mechanisms for error detection, logging, and user-friendly error messages.

How it Works:

User Interaction:

- *What happens:* The Frontend Application allows users to input their location and initiates the search for wineries.

Client-Server Interaction:

- *What happens:* The Frontend communicates with the Backend Application, triggering actions and requesting data.

Data Flow:

- *What happens:* The Backend Application orchestrates the flow of data, interacting with the Database Management System to retrieve winery information. It also communicates with external services like the OpenStreetMap API.

Additional Modules:

- *What happens:* Security, Logging and Monitoring, Asynchronous Processing, Containerization and Orchestration, Load Balancer, and Error Handling and Reporting modules ensure a secure, robust, and scalable implementation.