**Software design documentation**

**1 Introduction**

The Software Design Document describes the architecture and system design for Road Trip Adviser, a road trip planning website. Road Trip Adviser is designed to help travelers plan and oversee their trip. This document is intended for Project Managers, Software Engineers, and anyone else who will be involved in the implementation of the system.

**2. scope**

The scope of this Software Design Document (SDD) is to outline the architecture and design principles for the Road Trip Advisor (RTA) web application, aiming to simplify road trip planning with optimized routes, customization, and real-time interaction. Intended for project managers and software engineers, the document focuses on a user-friendly interface, efficient algorithms, and scalability. It excludes low-level implementation details and specific UI designs, serving as a crucial reference for the development, implementation, and maintenance of RTA.

**1.3 overview**

Software Design Document (SDD) of Road Trip Advsor(RTA) provides necessary definitions to conceptualize and further formalize design of the software, whose requirements and functionalities were summarized in Software Requirements Specifications (SRS) Report. Aim is to provide guidance to a design which could be easily implemented by any programmer reading this report. The document complies with the IEEE standards (IEEE Std 1016 – 2009).

**1.4 REFERENCE MATERIAL**

**1.5 Definitions and acronyms**

|  |  |
| --- | --- |
| **acronym** | **meaning** |
| RTA | ROAD TRIP ADVISOR |
| SDD | SOFTWARE DESIGN DOCUMENT |
| OS | OPERATING SYSTEM |
| API | APPLICATION PROGRAMMING INTERFACE |
|  |  |

2. System overview

The Road Trip Advisor Web Application is a robust platform designed to simplify and enhance the process of planning road trips. It caters to individuals seeking an efficient and personalized way to organize their travel experiences.

Functionality:

* Trip Planning: Users can input their starting and destination locations along with preferences, receiving optimized route suggestions tailored to their choices.
* Database Management: Storing trip details and user preferences enables the system to offer personalized trip suggestions and recall similar routes efficiently.
* User Authentication: Managing user accounts and sessions, ensuring personalized trip planning experiences.
* Map Integration: Utilizing map APIs to visually represent routes, user locations, and points of interest along the journey.
* Preference Customization: Allowing users to specify meal preferences, price ranges, and distances to tailor their trip experience.
* Real-time Interaction: Enabling users to make real-time updates to preferences, view saved trips, and modify planned trips dynamically.

**Context:**

The project operates within the domain of travel and trip planning, aiming to simplify the complexities users face when organizing road trips. It addresses uncertainties and decision-making challenges users encounter, providing a user-friendly platform to optimize their travel experiences. It's designed to consider and accommodate user preferences during the trip planning process.

**Design:**

**Architecture:** Utilizes a Client-Server model, leveraging NodeJS as the backend framework for efficient communication between client-side and server-side components.

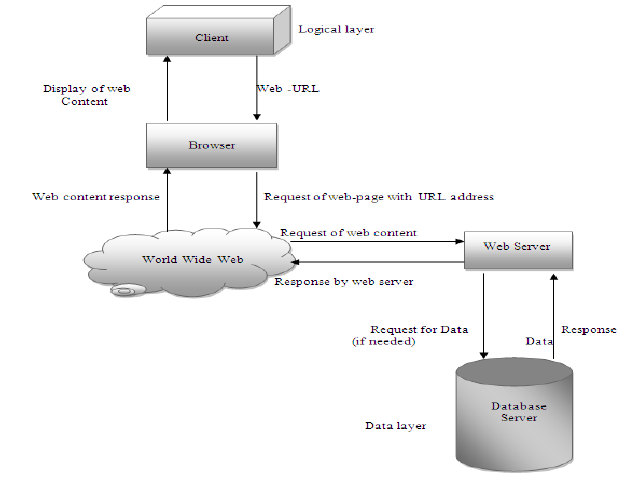


Figure 1.1 client server architecture of road trip advisor

**Module Structure**: Organizes functionalities into Trip Planning, Database, Optimization, Map, User, and Authentication modules, handling specific aspects of trip planning and management.

**Component Interaction**: Illustrates how various components like Trip Planner, Session Manager, Optimized Path Finder, User Profile, and UI Controller interact to facilitate the trip planning process.

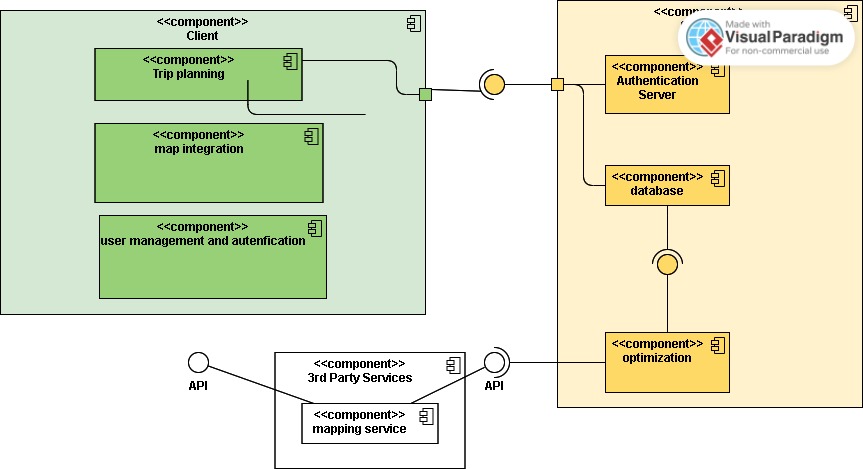
Interfaces: Includes intuitive user interfaces (GUI) for both new and returning users, offering landing pages, trip planning screens, and preference selection screens for a seamless user experience.

**Background:**

Developed by a team from Bellevue College supervised by Dr. Fatma Serce, the initial release (Version 1.0) of the application occurred on November 7th, 2018. Subsequent revisions were made to refine and enhance functionalities, aiming to provide a comprehensive and personalized solution for efficient trip planning and management.3.

3. **system architecture**

**3.1 architectural design**

****

Components:

**1. Client-Side (Frontend):**

**Components:**

**User Interface (UI):** The graphical interface that users interact with.

**UI Controller:** Manages user input and communicates with the backend.

**Preferences Interface**: Allows users to customize trip preferences.

**Map Interface:** Displays visual representations of routes and points of interest.

**Connectors:**

**HTTP/HTTPS Requests:** The UI Controller sends HTTP/HTTPS requests to the server to retrieve data or trigger actions.

**Communication:**

The UI interacts with the user, collects input, and sends requests to the server.

**2. Server-Side (Backend):**

**Components:**

**NodeJS Server:** The backend server implemented using NodeJS.

**Trip Planning Module:** Manages trip planning functionalities.

**Database Module:** Handles data storage and retrieval.

**Optimization Algorithm**: Performs route optimization.

**Authentication Module:** Manages user authentication and sessions.

Connectors:

**HTTP/HTTPS Responses**: The server responds to client requests with data or actions.

**Database Connection**: Connects to the database to store and retrieve trip-related data.

**Communication:**

The backend processes client requests, interacts with modules, and communicates with external services (e.g., mapping APIs).

**Communication Flow:**

User Input:

The user interacts with the UI, providing input for trip planning preferences.

UI Controller:

The UI Controller processes user input and forms HTTP/HTTPS requests.

HTTP/HTTPS Requests:

Requests are sent from the UI Controller to specific endpoints on the NodeJS server.

NodeJS Server:

The server processes incoming requests, invoking the relevant modules (Trip Planning, Database, Optimization, Authentication).

Module Interaction:

Modules interact with each other as needed. For example, the Trip Planning Module may communicate with the Optimization Algorithm and the Database Module.

Database Interaction:

The Database Module connects to the database to store and retrieve user profiles, trip details, and other relevant data.

HTTP/HTTPS Responses:

The server sends back responses to the client, containing requested data or confirmation of actions.

UI Update:

The UI updates based on the server responses, providing real-time feedback to the user.

3.2 Design Decisions:

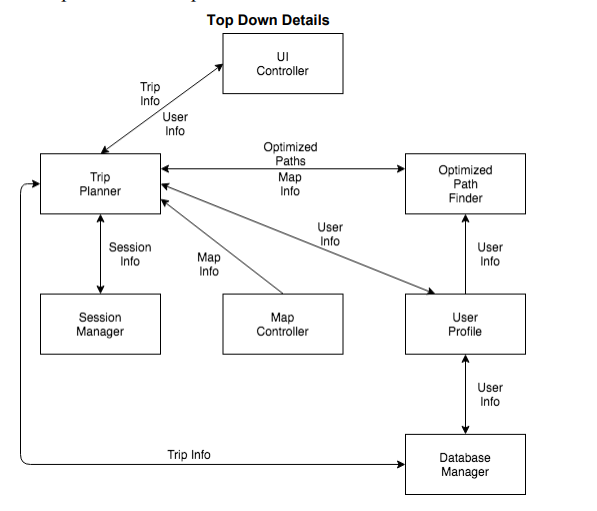
NodeJS as Backend: NodeJS is selected for the backend due to its event-driven model, aligning with real-time interactions, scalability, and development efficiency.

Client-Server Architecture: The decision is made based on the need for real-time interactions and scalability, with considerations for alternatives like monolithic or microservices architectures.

**Design entity and design entity attributess**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity name** | **UserID** | **MAP** | **Database** | **autentification** | **Trip** | **Optimization**  **Algorithm** |
| Identification | USER Id | Mapid | Database Id | SessionID | Trip ID | Algorithm ID |
| Type | USER | Map | storage | security | TRIP | Algorithm |
| PURPOSE | Store user information | Store map related data | Store application data | Manage user sessions | Manage trip information | Optimize trip planning |
| Subordinate | Manage user detials | None | user | User, database | Map, user | Trip, map |
| Function | Manage user details | Display maps, traffic | Data storage | Authenticate users | Plan and schedule trips | Optimize routes |
| Dependencies | None | None | User, authenfication | User, database | Map,user | Trip, map |
| Interface | User interface | UI interface | None | UI interface | UI | None |
| Resources | User profiles | Map data, traffic information | Storage capacity, backup | Session management data | Trip planning data | Algorithm parametres |
| Data | Updated when users update their profiles or trip history | Updated for real time data or map display changes | Updated when new data is added or during back up operations | Updated during login/ logout or session creation/deletion | Updated on users plan and modify trips | Updateduring route optimization |

**3.2 decomposition description**

****

**Figure 2 top down description of the application**

Figure 2 above shows a top down description of how the web application is expected to work and how components will interact with one another. The trip planner is the main component. The trip planner can save and recall trips from the database manager. The session manager is for logging in, logging out, and authenticating users. The trip planner can get and update user profiles that are saved with the database manager. The trip planner will use the optimized path finder to create routes. The optimized pathfinder can create routes usings the user info provided by the user profile. Finally, the UI controller, shows how the user interacts with the road trip adviser application.

**3.3 design rationalie**

The selection of a Client-Server architecture with NodeJS as the backend for the Road Trip Advisor (RTA) system stemmed from priorities on real-time interactions, scalability, and development efficiency. NodeJS's event-driven model aligned with the need for responsive trip planning, while its JavaScript-based full-stack development facilitated code sharing and eased development. Considered alternatives such as monolithic or microservices architectures, as well as traditional backend technologies like Java or .NET, posed trade-offs in terms of scalability, performance, or development complexity that didn't suit RTA's requirements for real-time, scalable, and efficient trip planning functionalities.

The key considerations to choose Client server architecture.

**Real-Time Interaction:**

Event-Driven Model: NodeJS, as the backend framework, employs an event-driven model. This ensures real-time communication between the client and server, allowing users to receive instant updates and make dynamic changes to their trip plans.

**Scalability:**

Efficient Communication: The Client-Server model facilitates efficient communication between the client-side and server-side components. This is essential for handling a potentially large user base and providing a responsive experience even as the application scales.

**Development Efficiency:**

JavaScript Full-Stack: Leveraging NodeJS for both frontend and backend development streamlines the development process. A JavaScript full-stack approach promotes code sharing, easing collaboration among developers and accelerating the implementation of new features.

**Responsive Trip Planning:**

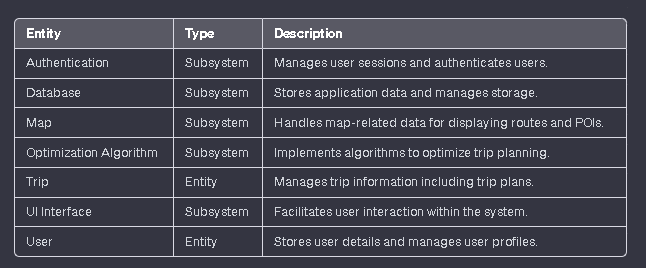
Prioritizing Real-Time Interactions: Other considered architectures, such as monolithic or microservices, posed trade-offs that didn't align with RTA's emphasis on real-time, scalable, and efficient trip planning functionalities.

Conclusion:

The Client-Server architecture, particularly with NodeJS as the backend framework, aligns with RTA's requirements for real-time interaction, scalability, and development efficiency. This design choice forms the foundation for a responsive and user-centric road trip planning experience.

4. Data Design

3.4 data dictionary



5. component Design

Component: Trip Planning pseudocode

Algorithm planTrip(startLocation, destination, preferences):

// Plans a trip based on user inputs and preferences

route = OptimizationAlgorithm.calculateOptimizedRoute(startLocation, destination, preferences)

return route

Component: Databasepseudocode

Function storeData(data):

// Stores given data in the database

Database.save(data)

Function retrieveData(query):

// Retrieves data based on the provided query

return Database.query(query)

Component: Map Integration pseudocode

Function displayMap(route, pointsOfInterest):

// Displays a map with the given route and points of interest

Map.showRoute(route)

Map.displayPOIs(pointsOfInterest)

Component: Authentication

Function authenticateUser(username, password):

// Authenticates user based on provided credentials

if User.validateCredentials(username, password):

Session.createSession(username)

return true

else:

return false

Component: Optimization Algorithm

Algorithm calculateOptimizedRoute(startLocation, destination, preferences):

// Calculates the optimized route based on user inputs and preferences

// Uses complex algorithms for optimization

optimizedRoute = Algorithm.calculateRoute(startLocation, destination, preferences)

return optimizedRoute

Component: UI Interface

pseudocode

Function displayTripPlan(route):

// Displays the planned trip on the user interface

UI.showTrip(route)

Function updateUserPreferences(user, newPreferences):

// Updates user preferences in the UI

UI.updatePreferences(user, newPreferences)

**6. human interface design**

6.1 overview of user interface

User Functionality:

**Trip Planning Interface:** Users access the system through a web application or mobile app. They input their starting point, destination, and trip preferences (such as preferred stops, meal choices, budget, etc.).

**Preferences Customization:** The system allows users to customize trip preferences, including meal types, price ranges, and distance limits for stops or attractions.

Real-Time Interaction: Users can make real-time adjustments to their trip plans, updating preferences or modifying trip details even during the planning phase.

**Route Optimization:** The system employs algorithms to optimize routes based on user inputs, providing the most efficient or preferred itinerary.

**Map Integration:** The planned route and points of interest (POIs) are displayed visually on an interactive map, offering users a clear visualization of their trip.

User Authentication: Users log in to manage their profiles, view saved trips, or access personalized suggestions based on their preferences and past trips.

**User Feedback and Information Display:**

**Trip Suggestions:** Upon inputting trip details, the system provides suggestions or multiple route options based on preferences, displaying estimated travel times, distances, and suggested stops.

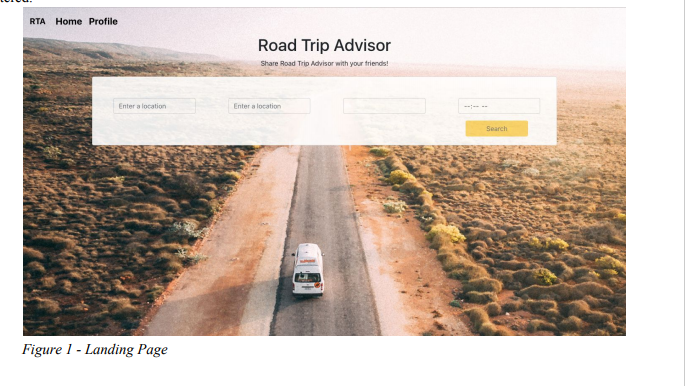
**Interactive Map:** The map interface offers a visual representation of the planned route, highlighting selected stops or attractions, traffic information, and nearby POIs based on user preferences.

Real-Time Updates: Users receive real-time updates on route changes, traffic conditions, or suggested attractions/restaurants that fit their preferences.

**Profile Management:** Feedback on saved trips, preferred destinations, or frequently visited locations can be displayed within the user profile, aiding future trip planning.

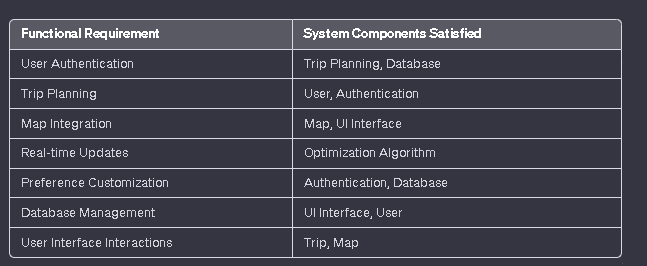
**Customization Confirmation:** Confirmation messages or summaries are displayed upon preferences customization or trip modifications to ensure user selections are accurately updated.

**6.2 screen images**

****

**6.3 screen objects and actions**

**7. Requirment matrix**

****

**8. Appendices**