ORP - Detailed Design

ORP (OffRoad Pathfinder)Architecture

The architecture best suited for our project is **Server-Client Architecture**.

Server-Client Architecture in Our Project:

Server:

- **Responsibilities:**
  - **Map Preprocessing and Graph Construction:**
    - Converts the user-selected points and map region (bounding box or points) into a graph using Digital Elevation Model (DEM) data.
    - Nodes represent traversable points, while edges represent valid paths calculated based on the slope (angle of attack) and terrain data.
    - Identifies obstacles and barriers (non-crossable areas) based on DEM data and excludes them from the graph.
  - **Pathfinding and Optimization:**
    - Implements the **RRT*** algorithm to calculate the optimal route between user-defined points (up to 5 points).
    - Handles multi-point route calculations and sequencing for efficient travel paths.
  - **Visualization Generation:**
    - Combines calculated routes with the satellite image of the selected region.
    - Overlays routes, user-defined points, and identified points of interest (POIs) onto the map.
    - Generates graphical outputs for display on the client.
  - **Trip Management:**
    - Groups routes into labeled "Trips," categorized by user-defined criteria (e.g., day, purpose, or custom tags).
    - Provides CRUD (Create, Read, Update, Delete) operations for routes and trips.

- **Data Storage:**
  - Stores processed routes, trip data, user preferences, and graph data for reuse in future requests.
  - Saves satellite images and DEM-based transformations for selected regions.

Client:

- **Responsibilities:**
  - **Interactive Map Interface:**
    - Displays a "live" map viewer using libraries like **Leaflet.js**, **Mapbox**, or **Google Maps API**.
    - Allows users to:
      - Select points of interest on the map.
      - Define up to 5 points for pathfinding.
      - Select a region by drawing a bounding box or clicking points.
  - **User Interaction and Input:**
    - Provides tools to adjust route preferences (e.g., avoid steep slopes or certain barriers).
    - Accepts input for grouping routes into trips and tagging them.
  - **Data Visualization:**
    - Displays generated routes overlaid on the selected satellite photo or map.
    - Highlights user-provided points and points of interest (POIs) identified by the server.
    - Allows for route exploration, zooming, and editing via the GUI.
  - **Communication with Server:**
    - Sends user-selected points, region, and trip details to the server.
    - Receives processed maps, routes, and trip data from the server for display.
  - **Local Cache and Performance:**
    - Temporarily caches data like user-selected points and map tiles for smoother interaction.
    - Provides immediate feedback for basic map operations (e.g., zooming, panning).

Data Storage

Server-Side:

- **Map Data:**
  - Stores DEM maps, satellite imagery, and preprocessed graph data for selected regions.
- **Route Data:**
  - Saves calculated routes with metadata (e.g., user-defined points, POIs, difficulty level, terrain type).
- **Trip Data:**
  - Organizes routes into trips, labeled by criteria such as day, region, or custom tags.
- **User Data:**
  - Tracks user preferences, trip histories, and saved regions/routes for personalization.

Client-Side:

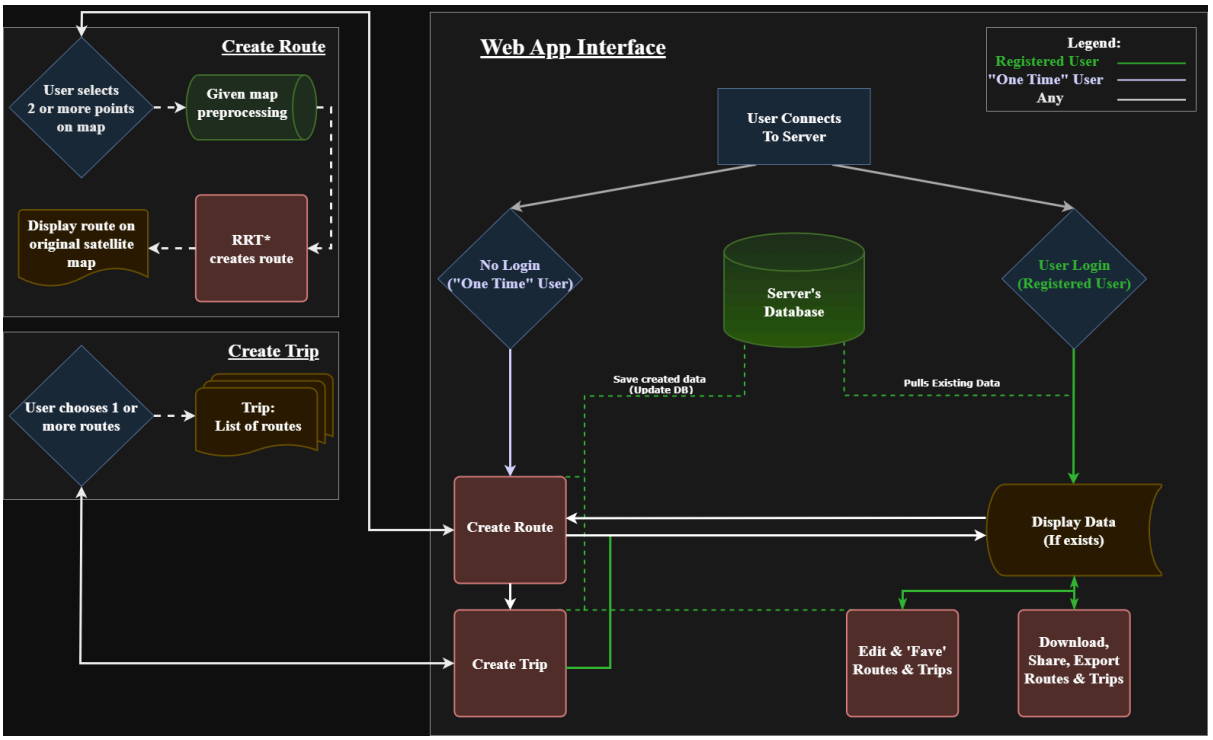
- **Cache:**
  - Temporarily caches user-selected points and basic map tiles for smoother interaction.

Interaction Flow

1. **Point and Region Selection:**
   - The client sends the user-selected points and the bounding region to the server.
   - User preferences (e.g., avoid steep slopes) are also sent.
2. **Map Preprocessing:**
   - The server processes the DEM data for the region, generating a graph with nodes and barriers.
3. **Route Calculation:**
   - The server runs the **RRT*** algorithm to compute the optimal path(s) between user-defined points.

4. **Route Visualization:**
    - The server overlays the calculated routes and POIs on the satellite map of the selected region and sends it to the client.
5. **Trip Management:**
    - The client organizes routes into trips and sends trip metadata to the server for storage.
6. **Display and Storage:**
    - The client displays the processed route and trip data to the user.
    - The server stores the finalized routes and trips for future access.

Flow Visualization



API Specification

User Interface:

- Upload DEM data or select regions on the map.
- Define points for pathfinding (up to 5 points).
- View calculated routes overlaid on satellite imagery.
- Group routes into trips and categorize them with custom tags.
- Retrieve previously saved trips or routes.

Endpoints:

- **POST /select-region:** Accepts bounding region data and user preferences.
- **POST /calculate-route:** Sends user-defined points and retrieves the calculated route.
- **POST /save-trip:** Saves a group of routes into a labeled trip.
- **GET /get-trip:** Retrieves previously stored trips or routes.

Programming Languages and Tools

Server:

- **Backend Framework:** Flask, FastAPI, or Django for API handling.
- **DEM Processing:** GDAL or Rasterio for map preprocessing.
- **Pathfinding Algorithms:** Custom RRT* implementation using Python.
- **Data Storage:** PostgreSQL/MongoDB, or a database with spatial extensions for geospatial data.
- **Visualization Tools:** PIL/Matplotlib for image processing or GeoJSON for map overlays.

Client:

- **Map Viewer:** Leaflet.js, Mapbox, or Google Maps API for interactive map functionality.
- **Frontend Framework:** React, Angular, or Vue.js for dynamic UI.
- **Communication:** RESTful APIs or WebSocket for data exchange.

Algorithm Description

Graph Construction:

- DEM maps are parsed to identify terrain features and calculate slopes between points.
- Obstacles or barriers are identified based on slope thresholds or predefined terrain types (e.g., water bodies, cliffs).
- The map is converted into a graph with:

    - **Nodes:** Traversable points.
    - **Edges:** Valid connections based on terrain analysis.

Pathfinding with RRT*:

- **Input:** Graph nodes, user-defined points, and terrain constraints.
- **Process:**
    - Randomly samples points within the region.
    - Builds a tree by connecting points based on traversal feasibility.
    - Optimizes the path using the RRT* algorithm to minimize cost.
- **Output:** Optimal route(s) between user-defined points.

Web App's Output

- **Routes:** Calculated and visualized on satellite maps.
- **Trips:** Organized and stored for later retrieval.
- **POIs:** Highlighted along the calculated routes.