The methodology employed in this project is structured to systematically address the design, development, and implementation of a comprehensive hostel navigation system for the Federal University of Technology Akure (FUTA). The approach integrates geospatial data handling, web-based technologies, real-time user interaction, and system validation to ensure the development of a reliable and user-friendly application. The following sections provide an extensive overview of the methodology.

1. Data Collection and Preparation

1.2. Data Cleaning and Processing

The raw data collected from GPS required cleaning and processing to ensure its suitability for database integration and map visualization. This involved the following processes:

1. Preparation of the GPS data and names iin an excel sheet. The table was structured and the names were cross checked. Duplicates were also removed.
2. The excel sheet was then saved as a csv file and imported to ArcGIS desktop for further processing.
3. The csv file was used to create a layer by importing the csv file to ArcGIS as xy data.
4. The created layer was then exported as a feature class
5. The shapefile was reprojected to the EPSG: 4326 (WGS 1984) coordinate system as the web mapping package to be used does not accept none other than the WGS 198 or web Mercator. The result was exported as a shapefile.

A PostgreSQL database named ‘FutaHostels’ was created using the PG Admin app on a windws PC

A postGIS extension was also added so as to facilitate the management of spatial data

The postGIS bundle app for PostgreSQL was launched and connected to the created database

The shapefile created from aArcMap was then imported to the database using the bundle app.

The database was assessed on PG Admin to check if it’s in a good condition.

The data was read to the web map application through a JQuery.ajax function on the javascript. This is used to execute a python file in the local web server (wampserver)

This python file connects to the database through the python psycopg package. This facilitates the connection to the database and executes queries on the database.

The Python code then process the result from the query and converts it to a geojson.

For example; during the retrieval of all hostels during the initialization of the map, the ajax function is used to execute a python function that connects to and query the database to return all features (hostels) as a geojson. The geojson data is then used by leaflet to ceate a visualization

1.3. Conversion to GeoJSON Format

To facilitate the integration of spatial data with web technologies, the cleaned data was converted into GeoJSON format. GeoJSON was chosen because it is lightweight and natively supported by most web mapping libraries, including Leaflet.js. The conversion process involved using Python's `geojson` library, ensuring that each hostel's data, including its name, coordinates, and additional attributes, was accurately represented in the GeoJSON files.

2. Database Design and Management

2.1. Database Schema Design

A robust database schema was designed to handle both spatial and non-spatial data efficiently. The database schema included tables for storing hostel information, such as hostel names, types, capacity, and coordinates. Relationships between tables were carefully designed to maintain data integrity and optimize query performance. The schema was implemented in PostgreSQL, chosen for its robust support for spatial data through the PostGIS extension.

2.2. Spatial Database Management with PostGIS

PostGIS was integrated into the PostgreSQL database to handle spatial data types and queries. The spatial indexing capabilities of PostGIS were leveraged to optimize the retrieval of location data, enabling quick and efficient queries that return hostel locations and spatial relationships (e.g., proximity to other campus landmarks). The use of PostGIS also facilitated advanced spatial analyses, such as calculating the distance between hostels and generating spatial buffers.

2.3. Data Insertion and Query Optimization

The cleaned and processed GeoJSON data was imported into the PostgreSQL database using Python scripts with the `psycopg2` library. Special attention was given to optimizing the database queries, particularly those involving spatial data, to ensure that the web application could handle multiple simultaneous requests without significant delays. Indexing strategies and query optimization techniques were applied to improve performance.

3. Web Development and Integration

3.1. Front-end Design with HTML, CSS, and JavaScript

The user interface of the hostel navigation system was developed using HTML, CSS, and JavaScript, ensuring a responsive and user-friendly design. The interface includes a map display, search bar, route planner, and additional information panels. CSS was used to style the elements, providing a visually appealing and consistent design across different devices and screen sizes.

3.2. Map Rendering with Leaflet.js

Leaflet.js, a popular open-source JavaScript library for interactive maps, was chosen to render the campus map and display hostel locations. Leaflet.js's lightweight nature and extensive plugin ecosystem made it ideal for this project. The map was initialized with multiple base layers (e.g., Google Streets, Google Hybrid, and Google Satellite), allowing users to switch between different views depending on their preference.

3.3. Integration of GeoJSON Data

The GeoJSON data containing hostel locations was integrated into the map using Leaflet.js's native support for GeoJSON layers. Each hostel was represented by a marker, with custom icons used to differentiate between different types of hostels (e.g., male, female, postgraduate). Popups were added to the markers to display detailed information about each hostel when clicked.

3.4. User Interaction Features

Interactive features were implemented to enhance the user experience. These included clickable hostel markers, draggable routes, and a search functionality that allows users to locate specific hostels by name. JavaScript functions were written to handle these interactions, ensuring smooth and intuitive user experience.

4. Implementing Search and Suggestion Features

4.1. Search Functionality with Python CGI Scripts

The search functionality was a key component of the hostel navigation system, allowing users to find specific hostels by name. This was implemented using Python Common Gateway Interface (CGI) scripts, which interacted with the PostgreSQL database to retrieve the relevant hostel data. The `search.py` script was designed to handle both exact and partial matches, ensuring flexibility in the search process.

4.2. Real-time Search Suggestions with AJAX and Fuzzy Matching

To enhance the search experience, real-time search suggestions were implemented using AJAX for asynchronous data retrieval and the `fuzzywuzzy` Python library for fuzzy string matching. As users typed in the search bar, AJAX requests were sent to the server to retrieve potential matches, which were then displayed as suggestions in a dropdown list. This approach significantly improved the usability of the search feature, particularly for users unsure of exact hostel names.

4.3. Displaying Search Results on the Map

Once a user selected a search result, the corresponding hostel's location was highlighted on the map, and the map view was automatically adjusted to center on the hostel. This was achieved by passing the selected hostel's coordinates from the CGI script to the Leaflet.js map via AJAX, ensuring a seamless and interactive experience.

5. Route Planning with OpenRouteService (ORS)

5.1. Initializing the ORS Client

Route planning was an essential feature of the hostel navigation system, enabling users to find the shortest path between two hostels. The OpenRouteService (ORS) API was used for this purpose, as it provides robust routing capabilities. The ORS client was initialized in the `route.py` Python script, with the project's API key securely stored and managed.

5.2. Implementing the Route Planning Algorithm

The route planning algorithm was implemented in Python, utilizing the ORS API to calculate the shortest driving route between two selected hostels. Users could input the names of the start and end hostels, which were then geocoded using the PostgreSQL database. The script sent the geocoded coordinates to the ORS API, which returned the route data, including distance, estimated time, and step-by-step directions.

5.3. Displaying Routes on the Map

The route data returned by the ORS API was displayed on the map using Leaflet.js. A polyline was drawn on the map to represent the route, with markers indicating the start and end points. Additional information, such as the total distance and estimated travel time, was displayed in an information panel beside the map. The route planning feature was designed to be dynamic, allowing users to drag the start or end markers to adjust the route in real-time.

6. Real-time Location Tracking

6.1. Geolocation API Integration

Real-time location tracking was implemented to allow users to determine their current position on the map. This feature utilized the browser's Geolocation API, which retrieves the user's current coordinates. The API was integrated into the web application using JavaScript, with fallback mechanisms in place for browsers that do not support geolocation.

6.2. Displaying User Location on the Map

Once the user's location was retrieved, it was displayed on the map with a distinct marker, different from those used for hostel locations. A radius circle was drawn around the marker to indicate the accuracy of the location data. The map was automatically centered on the user's location, providing a clear view of their position relative to nearby hostels.

6.3. Enhancing User Experience with Location Updates

To enhance the user experience, the system was designed to update the user's location in real-time, particularly for users navigating the campus. This was achieved by continuously polling the Geolocation API at regular intervals and updating the map marker accordingly. The feature was optimized to balance the frequency of location updates with the need to conserve device battery life.

7. System Testing and Validation

7.1. Unit Testing of Individual Components

Before integrating the various components of the hostel navigation system, each was subjected to rigorous unit testing to ensure they functioned correctly in isolation. This included testing the database queries, GeoJSON integration, map rendering, search functionality, route planning, and geolocation features. Python's `unittest` framework was used to automate and manage the testing process.

7.2. Integration Testing

Following successful unit testing, the components were integrated into a cohesive system and subjected to integration testing. This phase involved testing the interactions between different components, such as the communication between the front-end and back-end, the accuracy of spatial queries, and the performance of the map rendering with real-time data. Test cases were designed to cover various scenarios, including edge cases, to ensure the system's robustness.

7.3. User Testing and Feedback

To validate the system's usability and effectiveness, a series of user testing sessions were conducted. Selected users, including students and staff of FUTA, were invited to use the system and provide feedback on its functionality, ease of use, and performance. This feedback was analyzed to identify any issues or areas for improvement, which were then addressed in subsequent iterations of the system.

7.4. Performance Testing and Optimization

Performance testing was conducted to evaluate