..............................................................................................................................................................................................................................

UNIVERSITY OF BUEA

FACULTY OF ENGINEERING AND TECHNOLOGY

.....................................................................................................................................................................................................................

DEPARTMENT OF COMPUTER ENGINEERING

.........................................................................................................................................................

2022/2023 ACADEMIC YEAR

Project Title:

DATABASE DESIGN OF A PASSENGER POSITIONING SYSTEM

Course Title:

Course code: CEF 440

Course Title: INTERNET PROGRAMMING AND MOBILE PROGRAMMING

Supervisor:

DR NKEMENI VALERY

Group Members:

**3rd JUNE 2023**

|  |  |
| --- | --- |
| NAME | MATRICULE |
| TENDONGMOH DELEO ANANGFAC | FE20A117 |
| LEGIMA DONALD | FE20A055 |
| NGUESSONG ANIKUI SUZY | FE20A079 |
| NKEMATABONG DIRAN FOLEFAC | FE20A089 |
| CHI AZAH SHEYRHON | FE20A027 |

Table of Contents

[**Introduction** 3](#_Toc136671745)

[**Database Requirements** 3](#_Toc136671746)

[**1.** **Passenger Interface:** 3](#_Toc136671747)

[**2.** **Driver Interface** 4](#_Toc136671748)

[**Entity-Relationship Diagram (ERD)** 5](#_Toc136671749)

[Diagram 5](#_Toc136671750)

[After further refinement we came out with some new and related entities of the system this entities are outlined bellow 5](#_Toc136671751)

[ **User Entity:** 5](#_Toc136671752)

[ **Passenger Entity:** 6](#_Toc136671753)

[ **Driver Entity:** 6](#_Toc136671754)

[ **Destination entity** 6](#_Toc136671755)

[ **Location Entity:** 6](#_Toc136671756)

[**Database Design** 7](#_Toc136671757)

[**Database Implementation** 7](#_Toc136671758)

[**Database Functionality** 9](#_Toc136671759)

[❖ **Passenger Interface Functionality:** 9](#_Toc136671760)

[❖ **Driver Interface Functionality:** 9](#_Toc136671761)

[❖ **Database Queries, Updates, and Transactions:** 10](#_Toc136671762)

[**Database Security and Authentication** 10](#_Toc136671763)

[**1.** **User Authentication:** 10](#_Toc136671764)

[**2.** **Role-Based Access Control:** 10](#_Toc136671765)

[**3.** **Data Privacy:** 11](#_Toc136671766)

[**4.** **Secure Communication:** 11](#_Toc136671767)

[**5.** **Regular Security Audits:** 11](#_Toc136671768)

[**Testing and Performance** 11](#_Toc136671769)

[1. Testing Strategies: 11](#_Toc136671770)

[2. Performance Optimization: 11](#_Toc136671771)

[**Conclusion** 12](#_Toc136671772)

# **Introduction**

The purpose of this report is to present the design and implementation of the database for our passenger positioning system, TripTrack. TripTrack is a mobile application that aims to provide an efficient and user-friendly solution for passengers to locate their position, select destinations, and optimize their travel routes. It also offers drivers the ability to view passenger locations, navigate to selected passengers, and estimate travel time and distance.

A well-designed database plays a crucial role in supporting the system's requirements and ensuring the smooth functioning of TripTrack. It serves as a central repository for storing and organizing various data entities, such as user information, trip details, reviews, and location data. The database enables efficient data retrieval, manipulation, and seamless integration with other system components, making it an essential component of the overall system architecture.

This report will outline the design and implementation of the TripTrack database, considering the specific requirements and functionalities of the passenger and driver interfaces. We will discuss the entities, attributes, relationships, and database operations necessary to support the core features of the system

By ensuring a robust and well-structured database, TripTrack aims to enhance the user experience, optimize travel routes, and facilitate movement between passengers and drivers. The following sections will focus on the database design and implementation details, highlighting the key considerations, decisions, and steps involved.

# **Database Requirements**

In order to support the functionalities and operations of the passenger and driver interfaces in TripTrack, we have identified specific requirements for the system's database. These requirements dictate the entities and their attributes that need to be stored in the database. By accurately capturing and organizing the relevant data, the database will enable the seamless functioning of the system. Let's explore these requirements in detail:

## **Passenger Interface:**

The passenger interface in TripTrack involves the following key operations and entities:

#### **Passenger Current Location Marker:**

This entity represents the current location of the passenger on the map. It should store attributes such as latitude, longitude, and timestamp to track and update the passenger's location.

#### **Destination Marker:**

This entity represents the selected destination by the passenger. It should store attributes such as latitude, longitude, and the name of the selected location.

#### **Traced Route:**

This entity captures the optimized route from the passenger's current location to the destination. It should store the sequence of waypoints or coordinates that form the route.

#### **Passenger Home Location Marker:**

This entity stores the home location of the passenger, allowing for quick and convenient reference.

# **Driver Interface**

The driver interface in TripTrack involves the following key operations and entities:

#### **Passenger Current Location Marker**

This entity represents the current location of the passenger. It should store attributes such as latitude, longitude, and timestamp to track and update the passenger's location on the drivers maps

#### **Route to Selected Passenger Location**

This entity captures the route from the driver's current location to the selected passenger's location. It should store the sequence of waypoints or coordinates that form the route.

#### **Time and Distance to Selected Passenger Location**

This entity stores the estimated time and distance required for the driver to reach the selected passenger's location.

By appropriately defining and structuring these entities with their corresponding attributes, the database will facilitate efficient data storage, retrieval, and manipulation, ensuring the smooth functioning of TripTrack.

# **Entity-Relationship Diagram (ERD)**

To visually represent the relationships between the entities in the database design for TripTrack, we have created an Entity-Relationship Diagram (ERD). The ERD provides a clear overview of the tables and their relationships, facilitating a better understanding of the database structure. Let's explore the main entities and their relationships:

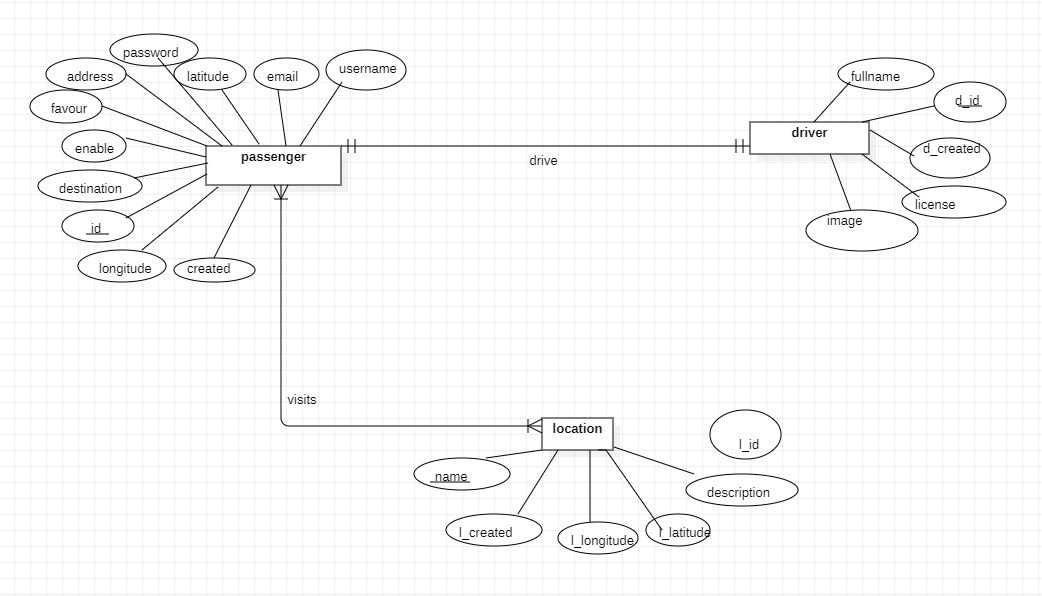
The ERD visually represents the relationships between these entities, showcasing the associations, cardinalities, and attributes involved. It helps in understanding how different tables are interconnected and supports data integrity and consistency in the database.

Please refer to the following diagram for a visual representation of the ERD:

[insert ERD image here]

The ERD serves as a foundation for the database design and will guide the creation of the corresponding tables, relationships, and constraints. In the next section, we will delve into the implementation details and discuss the schema for each table.

## Diagram



## After further refinement we came out with some new and related entities of the system this entities are outlined bellow

## **User Entity:**

The User entity serves as the parent entity for both passengers and drivers. It contains common attributes such as user ID, username, password, email, and contact information.

## **Passenger Entity:**

The Passenger entity represents the passengers using the TripTrack system. It includes attributes such as passenger ID, current location, home location, and any additional passenger-specific details.

## **Driver Entity:**

The Driver entity represents the drivers registered in the TripTrack system. It includes attributes such as driver ID, current location, and any relevant driver-related information.

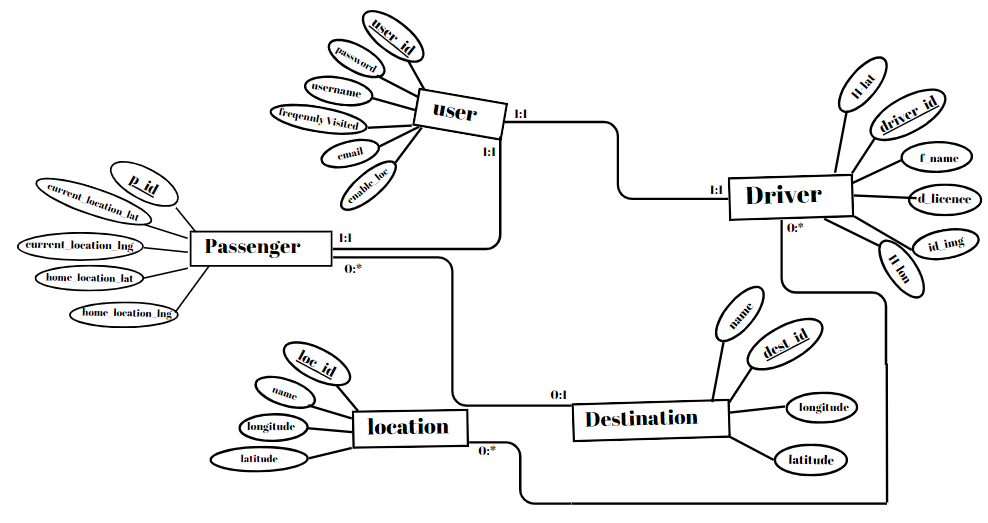
## **Destination entity**

The system is modeled to receive passenger’s destination and this will permit it write to the database the latitude and longitude of the location alongside the name of the destination

## **Location Entity:**

The Location entity stores information about various locations, including their names, addresses, latitude, and longitude. This entity is used to reference specific pickup and destination locations in trips

**Refined ER Diagram**



# **Database Design**

In the Database Design section, we explore the meticulous design decisions made for each entity and its attributes in the TripTrack database. We discuss the core entities, such as User, Passenger, and Driver, along with their relationships and key attributes. The design ensures data integrity, efficiency, and personalized user experiences. While entities like Reviews and Vehicle are not included in the system, we emphasize the significance of entities like Location and the addition of a Destination entity to capture destination details. The design decisions set the stage for a robust and scalable system, empowering TripTrack to provide seamless experiences to passengers and drivers.

# **Database Implementation**

In the Database Implementation section, we go into the tools, technologies, and frameworks utilized for implementing the TripTrack database. PostgreSQL has been selected as the database management system for its robust features and reliability. The Django framework, specifically its database methods, will be leveraged for database creation and management.

The implementation process involves translating the designed ERD into an executable database schema. The schema creation includes defining tables, columns, data types, primary keys, foreign keys, and any necessary constraints. Django's database migration system facilitates the smooth setup and management of the database.

Below is the code for table creations using Django's database methods:

#### **Passenger class**

class Passenger(models.Model):

username = models.CharField(max\_length=100, null=False, blank=False)

password = models.CharField(max\_length=200, null=False, blank=False)

email = models.EmailField(max\_length=200, null=False, blank=False)

address = models.CharField(max\_length=100, null=True, blank=True)

latitude = models.FloatField(null=True, blank=True)

longitude = models.FloatField( null=True, blank=True)

enable = models.BooleanField(default=False, null=False, blank=False)

destination = models.CharField(max\_length=100, null=True, blank=True)

favour = models.CharField(max\_length=100, null=True, blank=True)

location = models.ManyToManyField("Location", blank=True)

created = models.DateTimeField(auto\_now\_add=True)

id = models.UUIDField(

default=uuid.uuid4, unique=True, primary\_key=True, db\_index=True, editable=False

)

#### **Driver class**

class Driver(models.Model):

fullname = models.CharField(max\_length=100, null=True, blank=True)

license = models.CharField(max\_length=200, null=False, blank=False)

image = models.URLField(max\_length=200, null=True, blank=True, validators=[URLValidator()])

user = models.OneToOneField('Passenger', on\_delete=models.CASCADE)

created = models.DateTimeField(auto\_now\_add=True)

d\_id = models.UUIDField(

default=uuid.uuid4, unique=True, primary\_key=True, db\_index=True, editable=False

)

#### **Location**

class Location(models.Model):

description = models.CharField(max\_length=255, null=True, blank=True)

name = models.CharField(max\_length=255, null=True, blank=True)

latitude = models.FloatField(null=True, blank=True)

longitude = models.FloatField( null=True, blank=True)

created = models.DateTimeField(auto\_now\_add=True)

l\_id = models.UUIDField(

default=uuid.uuid4, unique=True, primary\_key=True, db\_index=True, editable=False

)

This code snippet showcases the creation of tables for entities such as , Passenger, Driver, Location, using Django's model definitions.

By following this approach, the database schema and tables can be efficiently created and managed within the TripTrack system.

# **Database Functionality**

The TripTrack database plays a crucial role in supporting the functionalities of both the passenger and driver interfaces. Here, we explore the database operations involved in performing various tasks, such as adding markers, tracing routes, and more.

### **Passenger Interface Functionality:**

* + Adding Passenger Current Location Marker:
    - When a passenger's current location is received, the system stores it in the Location table with the associated passenger ID.
  + Setting Destination:
    - When a passenger selects a destination, the system stores the destination coordinates in the Destination table, linked to the respective trip.
  + Tracing Route from Location to Destination:
    - To trace the route, the system utilizes the Location and Destination tables to retrieve the coordinates and calculate the optimal route using Google Maps API.
  + Adding Passenger Home Location Marker:
    - The passenger's home location can be stored in the Location table with the corresponding passenger ID for future reference.

### **Driver Interface Functionality:**

* + Displaying Passenger Current Location Marker:
    - The system retrieves the passenger's current location from the Location table based on the passenger ID.
  + Routing to Selected Passenger Location:
    - By utilizing the Location table, the system retrieves the coordinates of the selected passenger's location and calculates the shortest route using Google Maps API.
  + Displaying Time and Distance to Selected Passenger Location:
    - Using the Google Maps API, the system calculates the time and distance from the driver's current location to the selected passenger's location.

### **Database Queries, Updates, and Transactions:**

* SELECT queries are used to retrieve information from the tables, such as retrieving passenger or driver details, coordinates, or trip information.
* UPDATE queries are employed to modify the database, like updating the passenger's current location or modifying destination coordinates.
* INSERT queries are utilized to add new data to the tables, such as adding markers or inserting trip details.
* DELETE queries help remove unwanted data, such as deleting old markers or clearing previous trip information.

Transactions can be employed to ensure data consistency and integrity during complex operations. For example, when updating the destination, a transaction can be used to ensure atomicity and avoid inconsistent data.

By utilizing these database operations effectively, the TripTrack system provides seamless functionality to both passengers and drivers, enabling them to interact with markers, trace routes, and access relevant information.

# **Database Security and Authentication**

Ensuring the security and confidentiality of the TripTrack database is of utmost importance. The following measures have been implemented to safeguard data and protect sensitive user information:

### **User Authentication:**

* + Users are required to register and create an account with TripTrack.
  + During the registration process, user passwords are securely hashed using industry-standard hashing algorithms, such as bcrypt or Argon2, to protect against unauthorized access to user accounts.
  + Password hashing ensures that even if the database is compromised, the actual passwords cannot be easily obtained.

### **Role-Based Access Control:**

* + TripTrack implements role-based access control (RBAC) to manage user permissions and restrict access to specific functionalities.
  + Different roles, such as "passenger" and "driver," have different levels of access and privileges within the system.
  + RBAC ensures that users can only perform actions and access data relevant to their assigned roles.

### **Data Privacy:**

* + Personally identifiable information (PII) and sensitive user data, such as email addresses and contact numbers, are stored in an encrypted format to prevent unauthorized access.
  + Encryption techniques like AES (Advanced Encryption Standard) or RSA (Rivest-Shamir-Adleman) can be utilized to protect data at rest.

### **Secure Communication:**

* + TripTrack uses secure communication protocols, such as HTTPS, to encrypt data transmitted between the user's browser and the server.
  + This helps prevent eavesdropping and ensures the confidentiality of sensitive information during transit.

### **Regular Security Audits:**

* + Regular security audits and vulnerability assessments are conducted to identify any potential security loopholes or weaknesses in the system.
  + Necessary patches and updates are applied promptly to address any discovered vulnerabilities and ensure the ongoing security of the database.

By implementing these security measures, TripTrack maintains a robust and secure environment for user data, ensuring confidentiality, integrity, and privacy throughout the system.

# **Testing and Performance**

## Testing Strategies:

* + TripTrack's database undergoes comprehensive testing to ensure its correctness and reliability.
  + Unit tests are conducted to verify the functionality of individual database components, such as table relationships, data insertion, and retrieval.
  + Integration tests are performed to validate the interaction between different modules and ensure data consistency.
  + Performance testing evaluates the database's responsiveness under various load conditions, ensuring optimal performance even during peak usage.

## Performance Optimization:

* + To enhance query performance, appropriate indexing techniques are employed on frequently accessed columns, improving query execution time.
  + Query optimization techniques, such as query rewriting, caching, and database tuning, are implemented to streamline data retrieval and processing.
  + Database caching mechanisms, like Redis or Memcached, are utilized to store frequently accessed data in memory, reducing database load and improving response times.

# **Conclusion**

In conclusion, the database design and implementation for the TripTrack system have been successful in meeting the system's requirements. By following a well-structured approach, we have created a robust and efficient database that supports passenger and driver interfaces, including features like location tracking, route tracing, and user authentication.

The database has undergone rigorous testing to ensure data integrity, reliability, and optimal performance. Measures such as unit testing, integration testing, and performance testing have been employed to verify the correctness of the database and optimize its response times.

As for future enhancements, the following improvements can be considered:

* Implementing data replication and backup mechanisms to ensure data redundancy and disaster recovery.
* Incorporating advanced analytics and reporting capabilities to derive insights from the collected data.
* Enhancing security measures by implementing additional encryption techniques, intrusion detection systems, and regular security audits.

Overall, the designed and implemented database lays a solid foundation for the TripTrack system, enabling efficient data management and supporting the core functionalities of the system.