A logo on a grey background

Description automatically generated

NextPointLogix

Нотатки до ходу виконання проекту

Backend

**Recommendations for Next Steps**

1. **Backend Development**:
   * **Data Models**: Ensure all necessary data models and relationships are defined based on the current technical specifications.
   * **API Design**: Continue with the REST API design to cover all necessary interactions between the frontend and backend. This includes authentication, data retrieval, updates, and other operations.
   * **Security**: Implement authentication and authorization mechanisms, such as JWT for secure API access.
   * **Testing**: Set up unit and integration tests to ensure the backend services function correctly and reliably.
2. **Additional Sections for the Technical Specification**:
   * **Deployment Strategy**: Define how the application will be deployed and maintained in the production environment. This includes details about the server setup, CI/CD pipelines, and monitoring tools.
   * **User Stories or Use Cases**: Add detailed user stories or use cases to provide a clearer understanding of user interactions with the system.
   * **Maintenance and Support Plan**: Outline how ongoing support and maintenance will be handled, including potential updates and bug fixes.
3. **Backend Implementation**:
   * With the setup of PyCharm and Django, you can start implementing the backend services as per the REST API design.
   * Utilize the database schema already created to ensure seamless data storage and retrieval.
4. **Data Models**

This section describes the essential data models and their relationships for the TransLogix project. These models define the structure of the database and how different entities interact with each other. The data models are designed to support all functional and non-functional requirements outlined in the technical specification.

**1. User Model**

**Attributes:**

* user\_id: Unique identifier for each user (Primary Key)
* username: Login name for authentication
* password: Encrypted password for security
* email: User's email address for communication
* role: Defines the role of the user (e.g., driver, operator, manager, administrator)

**2. Driver Model**

**Attributes:**

* driver\_id: Unique identifier for each driver (Primary Key)
* user\_id: Foreign key linking to the User model
* name: Full name of the driver
* license\_number: Driver's license number
* phone\_number: Contact number

**3. Vehicle Model**

**Attributes:**

* vehicle\_id: Unique identifier for each vehicle (Primary Key)
* license\_plate: Vehicle's license plate number
* model: Vehicle model name
* capacity: Number of passengers the vehicle can transport
* fuel\_type: Type of fuel used by the vehicle

**4. Route Model**

**Attributes:**

* route\_id: Unique identifier for each route (Primary Key)
* start\_location: Starting point of the route
* end\_location: Destination of the route
* distance: Distance of the route in kilometers
* estimated\_time: Estimated travel time for the route

**5. Trip Model**

**Attributes:**

* trip\_id: Unique identifier for each trip (Primary Key)
* route\_id: Foreign key linking to the Route model
* driver\_id: Foreign key linking to the Driver model
* vehicle\_id: Foreign key linking to the Vehicle model
* date: Date of the trip
* status: Current status of the trip (e.g., scheduled, ongoing, completed)

**6. Fuel Log Model**

**Attributes:**

* fuel\_log\_id: Unique identifier for each fuel entry (Primary Key)
* trip\_id: Foreign key linking to the Trip model
* fuel\_amount: Amount of fuel added
* price: Price per unit of fuel
* date: Date of fueling
* station\_name: Name of the fuel station

**7. Feedback Model**

**Attributes:**

* feedback\_id: Unique identifier for each feedback entry (Primary Key)
* user\_id: Foreign key linking to the User model
* trip\_id: Foreign key linking to the Trip model
* comments: User's feedback or comments
* rating: User's rating for the trip

**8. Booking Request Model**

**Attributes:**

* booking\_id: Unique identifier for each booking request (Primary Key)
* user\_id: Foreign key linking to the User model
* route\_id: Foreign key linking to the Route model
* date: Date of the requested trip
* status: Current status of the booking (e.g., pending, confirmed, canceled)

**Relationships**

* **User to Driver**: One-to-One
* **User to Feedback**: One-to-Many
* **Driver to Trip**: One-to-Many
* **Vehicle to Trip**: One-to-Many
* **Route to Trip**: One-to-Many
* **Trip to Fuel Log**: One-to-Many
* **User to Booking Request**: One-to-Many
* **Route to Booking Request**: One-to-Many

**Additional Considerations**

* Implement necessary indexing for attributes frequently queried to optimize performance.
* Ensure proper validation and constraints on the attributes, such as unique constraints for username, email, and license\_plate.

1. **REST API Design**

**Base URL**

The base URL for all API requests is:

arduino

Copy code

https://api.translogix.com/v1/

**Authentication**

All API requests require authentication using JSON Web Tokens (JWT). Users must include the token in the Authorization header in the format Bearer <token>.

**Endpoints**

**1. User Authentication**

* **Login**
  + **Endpoint:** POST /auth/login
  + **Description:** Authenticates a user and returns a JWT token.
  + **Request Body:**

json

Copy code

{

"username": "string",

"password": "string"

}

* + **Response:**

json

Copy code

{

"token": "string",

"expires\_in": "integer"

}

* **Register**
  + **Endpoint:** POST /auth/register
  + **Description:** Registers a new user.
  + **Request Body:**

json

Copy code

{

"username": "string",

"password": "string",

"email": "string",

"role": "string"

}

* + **Response:**

json

Copy code

{

"user\_id": "integer",

"username": "string",

"email": "string",

"role": "string"

}

* **Logout**
  + **Endpoint:** POST /auth/logout
  + **Description:** Logs out the user and invalidates the token.
  + **Headers:**
    - Authorization: Bearer <token>

**2. Driver Management**

* **Get All Drivers**
  + **Endpoint:** GET /drivers
  + **Description:** Retrieves a list of all drivers.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Response:**

json

Copy code

[

{

"driver\_id": "integer",

"name": "string",

"license\_number": "string",

"phone\_number": "string"

}

]

* **Get Driver by ID**
  + **Endpoint:** GET /drivers/{driver\_id}
  + **Description:** Retrieves details of a specific driver.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Response:**

json

Copy code

{

"driver\_id": "integer",

"name": "string",

"license\_number": "string",

"phone\_number": "string"

}

* **Create Driver**
  + **Endpoint:** POST /drivers
  + **Description:** Adds a new driver.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Request Body:**

json

Copy code

{

"name": "string",

"license\_number": "string",

"phone\_number": "string"

}

* + **Response:**

json

Copy code

{

"driver\_id": "integer",

"name": "string",

"license\_number": "string",

"phone\_number": "string"

}

* **Update Driver**
  + **Endpoint:** PUT /drivers/{driver\_id}
  + **Description:** Updates an existing driver's details.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Request Body:**

json

Copy code

{

"name": "string",

"license\_number": "string",

"phone\_number": "string"

}

* + **Response:**

json

Copy code

{

"driver\_id": "integer",

"name": "string",

"license\_number": "string",

"phone\_number": "string"

}

* **Delete Driver**
  + **Endpoint:** DELETE /drivers/{driver\_id}
  + **Description:** Deletes a specific driver.
  + **Headers:**
    - Authorization: Bearer <token>

**3. Vehicle Management**

* **Get All Vehicles**
  + **Endpoint:** GET /vehicles
  + **Description:** Retrieves a list of all vehicles.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Response:**

json

Copy code

[

{

"vehicle\_id": "integer",

"license\_plate": "string",

"model": "string",

"capacity": "integer",

"fuel\_type": "string"

}

]

* **Get Vehicle by ID**
  + **Endpoint:** GET /vehicles/{vehicle\_id}
  + **Description:** Retrieves details of a specific vehicle.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Response:**

json

Copy code

{

"vehicle\_id": "integer",

"license\_plate": "string",

"model": "string",

"capacity": "integer",

"fuel\_type": "string"

}

* **Create Vehicle**
  + **Endpoint:** POST /vehicles
  + **Description:** Adds a new vehicle.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Request Body:**

json

Copy code

{

"license\_plate": "string",

"model": "string",

"capacity": "integer",

"fuel\_type": "string"

}

* + **Response:**

json

Copy code

{

"vehicle\_id": "integer",

"license\_plate": "string",

"model": "string",

"capacity": "integer",

"fuel\_type": "string"

}

* **Update Vehicle**
  + **Endpoint:** PUT /vehicles/{vehicle\_id}
  + **Description:** Updates an existing vehicle's details.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Request Body:**

json

Copy code

{

"license\_plate": "string",

"model": "string",

"capacity": "integer",

"fuel\_type": "string"

}

* + **Response:**

json

Copy code

{

"vehicle\_id": "integer",

"license\_plate": "string",

"model": "string",

"capacity": "integer",

"fuel\_type": "string"

}

* **Delete Vehicle**
  + **Endpoint:** DELETE /vehicles/{vehicle\_id}
  + **Description:** Deletes a specific vehicle.
  + **Headers:**
    - Authorization: Bearer <token>

**4. Trip Management**

* **Get All Trips**
  + **Endpoint:** GET /trips
  + **Description:** Retrieves a list of all trips.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Response:**

json

Copy code

[

{

"trip\_id": "integer",

"route\_id": "integer",

"driver\_id": "integer",

"vehicle\_id": "integer",

"date": "string",

"status": "string"

}

]

* **Get Trip by ID**
  + **Endpoint:** GET /trips/{trip\_id}
  + **Description:** Retrieves details of a specific trip.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Response:**

json

Copy code

{

"trip\_id": "integer",

"route\_id": "integer",

"driver\_id": "integer",

"vehicle\_id": "integer",

"date": "string",

"status": "string"

}

* **Create Trip**
  + **Endpoint:** POST /trips
  + **Description:** Adds a new trip.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Request Body:**

json

Copy code

{

"route\_id": "integer",

"driver\_id": "integer",

"vehicle\_id": "integer",

"date": "string",

"status": "string"

}

* + **Response:**

json

Copy code

{

"trip\_id": "integer",

"route\_id": "integer",

"driver\_id": "integer",

"vehicle\_id": "integer",

"date": "string",

"status": "string"

}

* **Update Trip**
  + **Endpoint:** PUT /trips/{trip\_id}
  + **Description:** Updates an existing trip's details.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Request Body:**

json

Copy code

{

"route\_id": "integer",

"driver\_id": "integer",

"vehicle\_id": "integer",

"date": "string",

"status": "string"

}

* + **Response:**

json

Copy code

{

"trip\_id": "integer",

"route\_id": "integer",

"driver\_id": "integer",

"vehicle\_id": "integer",

"date": "string",

"status": "string"

}

* **Delete Trip**
  + **Endpoint:** DELETE /trips/{trip\_id}
  + **Description:** Deletes a specific trip.
  + **Headers:**
    - Authorization: Bearer <token>

**5. Route Management**

* **Get All Routes**
  + **Endpoint:** GET /routes
  + **Description:** Retrieves a list of all routes.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Response:**

json

Copy code

[

{

"route\_id": "integer",

"start\_location": "string",

"end\_location": "string",

"distance": "float",

"estimated\_time": "string"

}

]

* **Get Route by ID**
  + **Endpoint:** GET /routes/{route\_id}
  + **Description:** Retrieves details of a specific route.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Response:**

json

Copy code

{

"route\_id": "integer",

"start\_location": "string",

"end\_location": "string",

"distance": "float",

"estimated\_time": "string"

}

* **Create Route**
  + **Endpoint:** POST /routes
  + **Description:** Adds a new route.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Request Body:**

json

Copy code

{

"start\_location": "string",

"end\_location": "string",

"distance": "float",

"estimated\_time": "string"

}

* + **Response:**

json

Copy code

{

"route\_id": "integer",

"start\_location": "string",

"end\_location": "string",

"distance": "float",

"estimated\_time": "string"

}

* **Update Route**
  + **Endpoint:** PUT /routes/{route\_id}
  + **Description:** Updates an existing route's details.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Request Body:**

json

Copy code

{

"start\_location": "string",

"end\_location": "string",

"distance": "float",

"estimated\_time": "string"

}

* + **Response:**

json

Copy code

{

"route\_id": "integer",

"start\_location": "string",

"end\_location": "string",

"distance": "float",

"estimated\_time": "string"

}

* **Delete Route**
  + **Endpoint:** DELETE /routes/{route\_id}
  + **Description:** Deletes a specific route.
  + **Headers:**
    - Authorization: Bearer <token>

**6. Feedback Management**

* **Get All Feedback**
  + **Endpoint:** GET /feedback
  + **Description:** Retrieves all feedback entries.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Response:**

json

Copy code

[

{

"feedback\_id": "integer",

"user\_id": "integer",

"trip\_id": "integer",

"comments": "string",

"rating": "integer"

}

]

* **Create Feedback**
  + **Endpoint:** POST /feedback
  + **Description:** Submits new feedback.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Request Body:**

json

Copy code

{

"user\_id": "integer",

"trip\_id": "integer",

"comments": "string",

"rating": "integer"

}

* + **Response:**

json

Copy code

{

"feedback\_id": "integer",

"user\_id": "integer",

"trip\_id": "integer",

"comments": "string",

"rating": "integer"

}

**7. Booking Management**

* **Get All Bookings**
  + **Endpoint:** GET /bookings
  + **Description:** Retrieves all booking requests.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Response:**

json

Copy code

[

{

"booking\_id": "integer",

"user\_id": "integer",

"route\_id": "integer",

"date": "string",

"status": "string"

}

]

* **Create Booking**
  + **Endpoint:** POST /bookings
  + **Description:** Creates a new booking request.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Request Body:**

json

Copy code

{

"user\_id": "integer",

"route\_id": "integer",

"date": "string",

"status": "string"

}

* + **Response:**

json

Copy code

{

"booking\_id": "integer",

"user\_id": "integer",

"route\_id": "integer",

"date": "string",

"status": "string"

}

* **Update Booking**
  + **Endpoint:** PUT /bookings/{booking\_id}
  + **Description:** Updates the status of a booking request.
  + **Headers:**
    - Authorization: Bearer <token>
  + **Request Body:**

json

Copy code

{

"status": "string"

}

* + **Response:**

json

Copy code

{

"booking\_id": "integer",

"user\_id": "integer",

"route\_id": "integer",

"date": "string",

"status": "string"

}

* **Delete Booking**
  + **Endpoint:** DELETE /bookings/{booking\_id}
  + **Description:** Deletes a specific booking request.
  + **Headers:**
    - Authorization: Bearer <token>

**Additional Considerations**

* Ensure all endpoints are secure by enforcing proper authentication and authorization.
* Validate all incoming data to prevent SQL injection and other vulnerabilities.
* Consider implementing rate limiting to prevent abuse of the API.

Під час виконання міграції у Django виявилось неможливим виконати міграцію, оскільки код конфліктував з уже наявними таблицями створеними через код SQL.

**Зроблено наступне:**

1. **Видалення існуючої бази даних**:
   * Користувач видалив існуючу базу даних translogix\_main\_db, яка була створена раніше за допомогою MySQL Workbench.
2. **Створення нової бази даних через Django міграції**:
   * Після видалення попередньої бази даних була створена нова база з тією ж назвою translogix\_main\_db, але вже через механізм міграцій Django.
   * Міграції були успішно застосовані, що означає, що всі необхідні таблиці були створені згідно з моделями в проекті Django.
3. **Пароль для користувача root**:
   * Користувач згадав пароль для користувача root: **Kitmarsik#1**.

Налаштування міграції дивитись у файлі Settings.py

C:\Users\ypyuryk\OneDrive - SoftServe, Inc\TransLogix\Solution files\myproject\backend files\TransLogix\_djangoProject\TransLogix\_djangoProject

**Кроки для створення API:**

**1. Створення серіалізаторів**

Серіалізатори перетворюють дані ваших моделей у формат JSON, який можна передати через API, і навпаки.

* Створіть файл serializers.py у вашому додатку (якщо його ще немає):

bash

Copy code

touch TransLogix\_djangoProject/serializers.py

* Додайте серіалізатор для однієї з моделей, наприклад, User:

python

Copy code

# TransLogix\_djangoProject/serializers.py

from rest\_framework import serializers

from .models import User

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = '\_\_all\_\_' # або вкажіть конкретні поля, які необхідно серіалізувати

**2. Створення представлень (views)**

Представлення обробляють запити та повертають відповідь API.

* Створіть файл views.py у вашому додатку (якщо його ще немає):

bash

Copy code

touch TransLogix\_djangoProject/views.py

* Додайте представлення для CRUD-операцій з користувачами:

python

Copy code

# TransLogix\_djangoProject/views.py

from rest\_framework import viewsets

from .models import User

from .serializers import UserSerializer

from rest\_framework.permissions import IsAuthenticated

class UserViewSet(viewsets.ModelViewSet):

queryset = User.objects.all()

serializer\_class = UserSerializer

permission\_classes = [IsAuthenticated] # захистити доступ аутентифікацією

**3. Налаштування маршрутизації (URLs)**

Маршрутизація визначає, які URL відповідають за виклик певних представлень.

* Створіть файл urls.py у вашому додатку (якщо його ще немає):

bash

Copy code

touch TransLogix\_djangoProject/urls.py

* Додайте маршрутизацію для ваших API ендпоінтів:

python

Copy code

# TransLogix\_djangoProject/urls.py

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views import UserViewSet

router = DefaultRouter()

router.register(r'users', UserViewSet) # ендпоінт для роботи з користувачами

urlpatterns = [

path('', include(router.urls)), # включаємо всі маршрути, що є в router

]

**4. Додайте маршрути до основного файлу urls.py проекту**

Переконайтеся, що ваш основний urls.py проекту включає маршрути вашого додатку.

* Відредагуйте основний urls.py вашого проекту:

python

Copy code

# TransLogix\_djangoProject/urls.py (основний файл, не у додатку)

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('api/', include('TransLogix\_djangoProject.urls')), # включаємо маршрути з додатку

path('api/auth/', include('rest\_framework.urls')), # ендпоінти для аутентифікації

]

**5. Тестування API**

Запустіть сервер розробки та перевірте ваш API.

bash

Copy code

python manage.py runserver

* Використовуйте Postman або інший клієнт для тестування API. Наприклад, ви можете зробити GET-запит до http://localhost:8000/api/users/ для отримання списку користувачів.

**6. Налаштування JWT аутентифікації**

Переконайтеся, що ви налаштували JWT аутентифікацію в settings.py, як було описано раніше. Використовуйте ендпоінти для отримання токенів:

python

Copy code

# Додайте до основного urls.py проекту для роботи з токенами

from rest\_framework\_simplejwt.views import (

TokenObtainPairView,

TokenRefreshView,

)

urlpatterns = [

path('api/token/', TokenObtainPairView.as\_view(), name='token\_obtain\_pair'),

path('api/token/refresh/', TokenRefreshView.as\_view(), name='token\_refresh'),

# інші маршрути

]

Отримавши JWT токен, використовуйте його для аутентифікації запитів до захищених ендпоінтів.

**Далі:**

* Додайте серіалізатори, представлення і маршрути для інших моделей, як це зроблено для User.
* Напишіть тести для API.
* Продовжуйте налаштовувати і розширювати API відповідно до вимог вашого проекту.

Скачав і створив акаунт в Postman.

Реалізація взаємодії бекенд-фронтенд для вікна авторизації.

 TokenObtainPairView: Endpoint for obtaining a new JWT (login).

 TokenRefreshView: Endpoint for refreshing the token when it’s about to expire.

 **Сторінка для скидання паролю (ForgotPassword.jsx)**:

* Створено сторінку скидання паролю, оформлену в стилі AppSelectionPage.css.
* Додано поле для введення юзернейму або е-мейлу. Після натискання на кнопку "Submit" перевіряється, чи існує користувач із вказаними даними. Якщо так, на вказаний е-мейл відправляється новий пароль.
* Підтвердження успішного скидання пароля виводиться у вигляді спливаючого вікна з повідомленням "An email with a new password has been sent to your email. We recommend changing it after logging in."
* Додано кнопку "Back", яка повертає на попередню сторінку.

 **Логіка для авторизації з новим паролем**:

* Після отримання нового пароля через пошту, користувач може використати його для входу в систему через сторінку логіну.
* Важливе: токен авторизації використовувався в заголовках запитів на зміну пароля.

Ці зміни дозволяють користувачам легко відновлювати пароль у разі його втрати, з дотриманням сучасних стандартів безпеки.

Додано функціонал.

 **Перевірка блокування юзера при логуванні**:

* Додана перевірка на статус блокування користувача. Якщо користувач заблокований, йому заборонено авторизуватись, і система відправляє відповідне повідомлення.

 **Інтеграція бекенду для сторінки адміністратора**:

* Реалізовано інтеграцію з бекендом, що дозволяє адміністратору отримувати та змінювати дані користувачів через API. Це включає блокування/розблокування користувачів та редагування ролей.

 **Кастомізація моделі юзера**:

* Внесені зміни у модель користувача для додавання нових ролей: логістичний оператор, фінансовий менеджер, адміністратор, staff та суперюзер.
* Ці зміни забезпечують більш гнучке управління доступом користувачів до різних частин системи.

 **Нові енд-поінти**:

* Створені нові енд-поінти для управління користувачами, які дозволяють змінювати ролі користувачів, блокувати та розблоковувати їх, а також отримувати дані про користувачів для відображення на сторінці адміністратора.

Повний список ендпоінтів станом на зараз (8оновити при додаванні нових)

urlpatterns = [  
 path('admin/', admin.site.urls), # Вже існуючий маршрут  
 path('api/auth/', include('rest\_framework.urls')), # Маршрут для аутентифікації  
 path('api/token/', TokenObtainPairView.as\_view(), name='token\_obtain\_pair'), # JWT токен  
 path('api/token/refresh/', TokenRefreshView.as\_view(), name='token\_refresh'), # Оновлення JWT токену  
 path('login/', CustomLoginView.as\_view(), name='custom\_login'), # Маршрут для кастомного логіну  
  
 # Новий маршрут для створення користувача  
 path('api/users/', CreateUserView.as\_view(), name='create\_user'),  
 path('api/change-password/', ChangePasswordView.as\_view(), name='change\_password'),  
 path('api/me/', UserProfileView.as\_view(), name='user-profile'),  
 path('api/forgot-password/', ForgotPasswordView.as\_view(), name='forgot\_password'), # Маршрут для скидання пароля  
 path('send-test-email/', send\_test\_email, name='send\_test\_email'),  
  
 path('api/', include(router.urls)), # Додаємо маршрути API з router  
 path('api/user-list/', UserListView.as\_view(), name='user-list'), # Додаємо маршрут для отримання списку користувачів  
 path('api/user-update/', update\_users, name='user-update'),  
 path('api/custom-login/', CustomLoginView.as\_view(), name='custom\_login'),  
]