API Documentation

- Anne Ha
- Bence Danko
- Cyril Goud Bhooma Goud
- David Thach
- Kushal Atulbhai Adhyaru
- Nairui Liu

Driver Service API Documentation

Version: 1.0 **Base URL:** `/api/v1` **Authentication:** Bearer Token (Specifics TBD - All endpoints should require authentication) **Content-Type:** `application/json`

Overview

The Driver Service is responsible for managing driver profiles, including creation, retrieval, updates, deletion, and searching. It handles driver information such as personal details, contact information, vehicle details, ratings, reviews, and location updates.

Data Structures

Driver Object

Represents a driver in the system. Used in GET responses, POST/PATCH responses.

```
{
  "driverId": "string (SSN Format: xxx-xx-xxxx)",
  "firstName": "string",
  "lastName": "string",
  "address": {
    "street": "string",
    "city": "string",
    "state": "string (Valid US State Abbreviation or Full Name)",
    "zipCode": "string (Format: xxxxx or xxxxx-xxxx)"
  },
  "phoneNumber": "string",
  "email": "string (email format)",
  "carDetails": {
    "make": "string",
    "model": "string",
    "year": "integer",
```

```
    "color": "string",
    "licensePlate": "string"
  },
  "rating": "number (float, 1.0-5.0)",
  "reviews": [ // Optional: Array of review snippets or IDs
    {
      "reviewId": "string",
      "customerId": "string (SSN Format)",
      "rating": "integer (1-5)",
      "comment": "string",
      "timestamp": "string (ISO 8601 Format)"
    }
    // ... more reviews
  ],
  "introduction": { // URLs to stored media
    "imageUrl": "string (URL)",
    "videoUrl": "string (URL)"
  },
  "ridesHistory": [ // Array of summary objects
     { "rideId": "string", "date": "string", "fare": number }
  ],
  "currentLocation": { // Optional: driver's last known location
      "latitude": "number",
      "longitude": "number",
      "timestamp": "string (ISO 8601 Format)"
  },
  "createdAt": "string (ISO 8601 Format)",
  "updatedAt": "string (ISO 8601 Format)"
}
```

Driver Input Object (for POST)

Used in the request body when creating a new driver. `driverId` must be provided and adhere to SSN format.

```
{
  "driverId": "string (SSN Format: xxx-xx-xxxx)", // Required
  "firstName": "string", // Required
  "lastName": "string", // Required
  "address": { // Required
    "street": "string", // Required
    "city": "string", // Required
    "state": "string (Valid US State Abbreviation or Full Name)", // Required
    "zipCode": "string (Format: xxxxx or xxxxx-xxxx)" // Required
  },
  "phoneNumber": "string", // Required
```

```
  "email": "string (email format)", // Required
  "carDetails": { // Required
    "make": "string", // Required
    "model": "string", // Required
    "year": "integer", // Required
    "color": "string", // Required
    "licensePlate": "string" // Required
  },
  "introduction": { // Optional
    "imageUrl": "string (URL)",
    "videoUrl": "string (URL)"
  }
  // Rating, Reviews, RidesHistory, Location are typically not set on creation
}
```

Driver Update Object (for PATCH)

Used in the request body when updating a driver. Only include fields that need to be changed.

```
{
  "firstName": "string", // Optional
  "lastName": "string", // Optional
  "address": { // Optional, include fields within address to change
    "street": "string",
    "city": "string",
    "state": "string (Valid US State Abbreviation or Full Name)",
    "zipCode": "string (Format: xxxxx or xxxxx-xxxx)"
  },
  "phoneNumber": "string", // Optional
  "email": "string (email format)", // Optional
  "carDetails": { // Optional, include fields within carDetails to change
    "make": "string",
    "model": "string",
    "year": "integer",
    "color": "string",
    "licensePlate": "string"
  },
  "introduction": { // Optional
    "imageUrl": "string (URL)",
    "videoUrl": "string (URL)"
  },
  // Note: Rating and Reviews might be updated by other services (e.g., Rides/Billing)
  // but this endpoint *could* allow admin updates if needed.
  "rating": "number (float, 1.0-5.0)", // Optional - Use with caution
  "reviews": [ /* ... */ ] // Optional - Use with caution
```

```
}
```

Location Update Object (for PATCH /location)

```
{
  "latitude": "number", // Required
  "longitude": "number" // Required
}
```

Error Response Object

Standard format for error responses.

```
{
  "error": "string (error code, e.g., 'invalid_input', 'not_found')",
  "message": "string (detailed error message)"
}
```

Endpoints

1. Create Driver

- **Description:** Registers a new driver in the system.
- **Endpoint:** `POST /drivers`
- **Request Body:** `Driver Input Object`
- **Responses:**
  - `201 Created`: Driver successfully created. Response body contains the created `Driver Object`. Includes `Location` header: `/drivers/{new_driver_id}`.
  - `400 Bad Request`: Invalid input data (e.g., missing required fields, invalid email format, invalid SSN format for `driverId`, invalid state, invalid zip code). Response body contains `Error Response Object`.
    - Example Error Codes: `invalid_input`, `missing_required_field`, `invalid_ssn_format`, `malformed_state`, `malformed_zipcode`.
  - `409 Conflict`: A driver with the provided `driverId` already exists. Response body contains `Error Response Object`.
    - Example Error Code: `duplicate_driver`.
  - `500 Internal Server Error`: Unexpected server error.

2. Get Driver by ID

- **Description:** Retrieves the full details of a specific driver.
- **Endpoint:** `GET /drivers/{driver_id}`
- **Path Parameters:**
  - `driver_id` (string, required): The SSN format ID of the driver to retrieve.

- **Responses:**
  - `200 OK`: Driver details successfully retrieved. Response body contains the `Driver Object`.
  - `400 Bad Request`: Invalid `driver_id` format provided in the path. Response body contains `Error Response Object`.
    - Example Error Code: `invalid_driver_id_format`.
  - `404 Not Found`: No driver found with the specified `driver_id`. Response body contains `Error Response Object`.
    - Example Error Code: `driver_not_found`.
  - `500 Internal Server Error`: Unexpected server error.

3. Update Driver Information

- **Description:** Updates specific fields for an existing driver. Supports partial updates.
- **Endpoint:** `PATCH /drivers/{driver_id}`
- **Path Parameters:**
  - `driver_id` (string, required): The SSN format ID of the driver to update.
- **Request Body:** `Driver Update Object` (containing only the fields to be updated).
- **Responses:**
  - `200 OK`: Driver successfully updated. Response body contains the updated `Driver Object`.
  - `400 Bad Request`: Invalid input data in the request body (e.g., invalid email format, invalid state, invalid zip code). Response body contains `Error Response Object`.
    - Example Error Codes: `invalid_input`, `malformed_state`, `malformed_zipcode`.
  - `400 Bad Request`: Invalid `driver_id` format provided in the path. Response body contains `Error Response Object`.
    - Example Error Code: `invalid_driver_id_format`.
  - `404 Not Found`: No driver found with the specified `driver_id`. Response body contains `Error Response Object`.
    - Example Error Code: `driver_not_found`.
  - `500 Internal Server Error`: Unexpected server error.

4. Delete Driver

- **Description:** Removes a driver from the system. (Consider soft delete vs hard delete).
- **Endpoint:** `DELETE /drivers/{driver_id}`
- **Path Parameters:**
  - `driver_id` (string, required): The SSN format ID of the driver to delete.
- **Responses:**

  - `204 No Content`: Driver successfully deleted. No response body.
  - `400 Bad Request`: Invalid `driver_id` format provided in the path. Response body contains `Error Response Object`.
    * Example Error Code: `invalid_driver_id_format`.
  - `404 Not Found`: No driver found with the specified `driver_id`. Response body contains `Error Response Object`.
    * Example Error Code: `driver_not_found`.
  - `500 Internal Server Error`: Unexpected server error.

5. List and Search Drivers

- **Description:** Retrieves a list of drivers, optionally filtered by provided criteria.
- **Endpoint:** `GET /drivers`
- **Query Parameters (Optional):**
  - `city` (string): Filter by city.
  - `state` (string): Filter by state abbreviation or full name.
  - `zipCode` (string): Filter by zip code.
  - `min_rating` (number): Filter drivers with rating >= this value.
  - `car_make` (string): Filter by car make.
  - `car_model` (string): Filter by car model.
  - `limit` (integer, default: 20): Maximum number of results to return (for pagination).
  - `offset` (integer, default: 0): Number of results to skip (for pagination).
- **Responses:**
  - `200 OK`: Successfully retrieved list of drivers. Response body contains a JSON array of `Driver Object`s matching the criteria. The array might be empty if no drivers match. Include pagination headers if implemented (e.g., `X-Total-Count`).
  - `400 Bad Request`: Invalid format for query parameters (e.g., non-numeric `min_rating`). Response body contains `Error Response Object`.
    * Example Error Code: `invalid_query_parameter`.
  - `500 Internal Server Error`: Unexpected server error.

6. Update Driver Location

- **Description:** Allows a driver (or the system simulating the driver's app) to update their current geographical location. This is crucial for matching drivers with nearby ride requests.
- **Endpoint:** `PATCH /drivers/{driver_id}/location`
- **Path Parameters:**
  - `driver_id` (string, required): The SSN format ID of the driver whose location is being updated.
- **Request Body:** `Location Update Object`

- **Responses:**
  - `200 OK`: Location successfully updated. Response body could optionally contain the updated `Driver Object` or just the updated location part.
  - `204 No Content`: Location successfully updated. No response body. (Choose one: 200 or 204).
  - `400 Bad Request`: Invalid input data (missing latitude/longitude, non-numeric values). Response body contains `Error Response Object`.
    - Example Error Code: `invalid_location_data`.
  - `400 Bad Request`: Invalid `driver_id` format provided in the path. Response body contains `Error Response Object`.
    - Example Error Code: `invalid_driver_id_format`.
  - `404 Not Found`: No driver found with the specified `driver_id`. Response body contains `Error Response Object`.
    - Example Error Code: `driver_not_found`.
  - `500 Internal Server Error`: Unexpected server error.

Validation Notes

- **SSN Format:** All `driverId` and `customerId` fields must strictly adhere to the `xxx-xx-xxxx` format.
- **State:** State fields must be validated against a list of valid US state abbreviations or full names. Return `malformed_state` error if invalid.
- **Zip Code:** Zip code fields must adhere to `xxxxx` or `xxxxx-xxxx` format. Return `malformed_zipcode` error if invalid.
- **Email:** Email fields should undergo basic format validation.
- **Phone Number:** Consider standardizing or validating phone number formats if necessary.
- **Rating:** Numeric ratings should be within the valid range (e.g., 1.0 to 5.0).

Customer Service API Documentation

Version: 1.0 **Base URL:** `/api/v1` **Authentication:** Bearer Token (Specifics TBD - All endpoints should require authentication) **Content-Type:** `application/json` (except for image upload)

Overview

The Customer Service is responsible for managing customer profiles, including creation, retrieval, updates, deletion, and searching. It handles customer infor-

mation such as personal details, contact information, payment details, ratings, reviews, and ride history associations.

Data Structures

Customer Object

Represents a customer in the system. Used in GET responses, POST/PATCH responses.

```
{
  "customerId": "string (SSN Format: xxx-xx-xxxx)",
  "firstName": "string",
  "lastName": "string",
  "address": {
    "street": "string",
    "city": "string",
    "state": "string (Valid US State Abbreviation or Full Name)",
    "zipCode": "string (Format: xxxxx or xxxxx-xxxx)"
  },
  "phoneNumber": "string",
  "email": "string (email format)",
  "creditCardDetails": { // **SECURITY NOTE:** Storing raw CC details is highly insecure. Us
    "last4Digits": "string",
    "cardType": "string (e.g., Visa, Mastercard)",
    "expiryMonth": "integer (1-12)",
    "expiryYear": "integer (YYYY)"
    // Raw number and CVV should NOT be stored directly in a real system.
  },
  "rating": "number (float, 1.0-5.0, given by drivers)", // Managed externally
  "reviews": [ // Optional: Array of review snippets or IDs (reviews *about* the customer)
    {
      "reviewId": "string",
      "driverId": "string (SSN Format)",
      "rating": "integer (1-5)",
      "comment": "string",
      "timestamp": "string (ISO 8601 Format)"
    }
    // ... more reviews
  ],
  "ridesHistory": [ // Array of summary objects
    { "rideId": "string", "date": "string", "fare": number }
    // ... more ride IDs/summaries
  ],
  "createdAt": "string (ISO 8601 Format)",
```

```
  "updatedAt": "string (ISO 8601 Format)"
}
```

Customer Input Object (for POST)

Used in the request body when creating a new customer. `customerId` must be provided and adhere to SSN format.

```
{
  "customerId": "string (SSN Format: xxx-xx-xxxx)", // Required
  "firstName": "string", // Required
  "lastName": "string", // Required
  "address": { // Required
    "street": "string", // Required
    "city": "string", // Required
    "state": "string (Valid US State Abbreviation or Full Name)", // Required
    "zipCode": "string (Format: xxxxx or xxxxx-xxxx)" // Required
  },
  "phoneNumber": "string", // Required
  "email": "string (email format)", // Required
  "creditCardDetails": { // Required - **SECURITY NOTE:** See Customer Object note. For simu
    "cardNumber": "string", // **Insecure for production**
    "expiryMonth": "integer (1-12)", // Required
    "expiryYear": "integer (YYYY)", // Required
    "cvv": "string" // **Highly Insecure for production**
  }
  // Rating, Reviews, RidesHistory are typically not set on creation
}
```

Customer Update Object (for PATCH)

Used in the request body when updating a customer. Only include fields that need to be changed.

```
{
  "firstName": "string", // Optional
  "lastName": "string", // Optional
  "address": { // Optional, include fields within address to change
    "street": "string",
    "city": "string",
    "state": "string (Valid US State Abbreviation or Full Name)",
    "zipCode": "string (Format: xxxxx or xxxxx-xxxx)"
  },
  "phoneNumber": "string", // Optional
  "email": "string (email format)", // Optional
  "creditCardDetails": { // Optional - **SECURITY NOTE:** Handle updates carefully.
    "cardNumber": "string", // **Insecure for production**
```

```
    "expiryMonth": "integer (1-12)",
    "expiryYear": "integer (YYYY)",
    "cvv": "string" // **Highly Insecure for production**
  }
  // Note: Rating and Reviews might be updated by other services (e.g., Rides/Billing)
}
```

Image Upload Response Object

```
{
  "imageUrl": "string (URL of the uploaded image)"
}
```

Error Response Object

Standard format for error responses.

```
{
  "error": "string (error code, e.g., 'invalid_input', 'not_found')",
  "message": "string (detailed error message)"
}
```

Endpoints

1. Create Customer

- **Description:** Registers a new customer in the system.
- **Endpoint:** `POST /customers`
- **Request Body:** `Customer Input Object`
- **Responses:**
  - `201 Created`: Customer successfully created. Response body contains the created `Customer Object` (with sensitive CC details masked/omitted as per security policy). Includes `Location` header: `/customers/{new_customer_id}`.
  - `400 Bad Request`: Invalid input data (e.g., missing required fields, invalid email format, invalid SSN format for `customerId`, invalid state, invalid zip code, invalid CC details). Response body contains `Error Response Object`.
    - Example Error Codes: `invalid_input`, `missing_required_field`, `invalid_ssn_format`, `malformed_state`, `malformed_zipcode`, `invalid_credit_card`.
  - `409 Conflict`: A customer with the provided `customerId` already exists. Response body contains `Error Response Object`.
    - Example Error Code: `duplicate_customer`.
  - `500 Internal Server Error`: Unexpected server error.

2. Get Customer by ID

- **Description:** Retrieves the full details of a specific customer.
- **Endpoint:** `GET /customers/{customer_id}`
- **Path Parameters:**
  - `customer_id` (string, required): The SSN format ID of the customer to retrieve.
- **Responses:**
  - `200 OK`: Customer details successfully retrieved. Response body contains the `Customer Object` (with sensitive CC details masked/omitted).
  - `400 Bad Request`: Invalid `customer_id` format provided in the path. Response body contains `Error Response Object`.
    - Example Error Code: `invalid_customer_id_format`.
  - `404 Not Found`: No customer found with the specified `customer_id`. Response body contains `Error Response Object`.
    - Example Error Code: `customer_not_found`.
  - `500 Internal Server Error`: Unexpected server error.

3. Update Customer Information

- **Description:** Updates specific fields for an existing customer. Supports partial updates.
- **Endpoint:** `PATCH /customers/{customer_id}`
- **Path Parameters:**
  - `customer_id` (string, required): The SSN format ID of the customer to update.
- **Request Body:** `Customer Update Object` (containing only the fields to be updated).
- **Responses:**
  - `200 OK`: Customer successfully updated. Response body contains the updated `Customer Object` (with sensitive CC details masked/omitted).
  - `400 Bad Request`: Invalid input data in the request body (e.g., invalid email format, invalid state, invalid zip code, invalid CC details). Response body contains `Error Response Object`.
    - Example Error Codes: `invalid_input`, `malformed_state`, `malformed_zipcode`, `invalid_credit_card`.
  - `400 Bad Request`: Invalid `customer_id` format provided in the path. Response body contains `Error Response Object`.
    - Example Error Code: `invalid_customer_id_format`.
  - `404 Not Found`: No customer found with the specified `customer_id`. Response body contains `Error Response Object`.
    - Example Error Code: `customer_not_found`.
  - `500 Internal Server Error`: Unexpected server error.

4. **Delete Customer**

- **Description:** Removes a customer from the system. (Consider soft delete vs hard delete).
- **Endpoint:** `DELETE /customers/{customer_id}`
- **Path Parameters:**
  - `customer_id` (string, required): The SSN format ID of the customer to delete.
- **Responses:**
  - `204 No Content`: Customer successfully deleted. No response body.
  - `400 Bad Request`: Invalid `customer_id` format provided in the path. Response body contains `Error Response Object`.
    - Example Error Code: `invalid_customer_id_format`.
  - `404 Not Found`: No customer found with the specified `customer_id`. Response body contains `Error Response Object`.
    - Example Error Code: `customer_not_found`.
  - `500 Internal Server Error`: Unexpected server error.

5. **List and Search Customers**

- **Description:** Retrieves a list of customers, optionally filtered by provided criteria.
- **Endpoint:** `GET /customers`
- **Query Parameters (Optional):**
  - `city` (string): Filter by city.
  - `state` (string): Filter by state abbreviation or full name.
  - `zipCode` (string): Filter by zip code.
  - `min_rating` (number): Filter customers with rating >= this value.
  - `limit` (integer, default: 20): Maximum number of results to return (for pagination).
  - `offset` (integer, default: 0): Number of results to skip (for pagination).
- **Responses:**
  - `200 OK`: Successfully retrieved list of customers. Response body contains a JSON array of `Customer Objects` (with sensitive CC details masked/omitted) matching the criteria. The array might be empty if no customers match. Include pagination headers if implemented (e.g., `X-Total-Count`).
  - `400 Bad Request`: Invalid format for query parameters (e.g., non-numeric `min_rating`). Response body contains `Error Response Object`.
    - Example Error Code: `invalid_query_parameter`.
  - `500 Internal Server Error`: Unexpected server error.

6. **Upload Image for a Ride Event**

- **Description:** Allows a customer to upload an image associated with a specific event during one of their rides. (**Note:** As per project requirement under Customer Module. Consider if this logically fits better under a dedicated Ride Service in a real-world scenario).
- **Endpoint:** `POST /customers/{customer_id}/rides/{ride_id}/images`
- **Content-Type:** `multipart/form-data`
- **Path Parameters:**
  - `customer_id` (string, required): The SSN format ID of the customer uploading the image. Must match the authenticated user.
  - `ride_id` (string, required): The ID of the ride the image relates to. The customer must be associated with this ride.
- **Request Body:** Form data containing the image file (e.g., under a key named `image`).
- **Responses:**
  - `201 Created`: Image successfully uploaded and associated with the ride event. Response body contains `Image Upload Response Object`.
  - `400 Bad Request`: Invalid `customer_id` or `ride_id` format, or missing image file in the request. Response body contains `Error Response Object`.
    - Example Error Codes: `invalid_customer_id_format`, `invalid_ride_id_format`, `missing_image_file`.
  - `403 Forbidden`: Authenticated customer does not match `customer_id` in the path or is not associated with the specified `ride_id`.
  - `404 Not Found`: No customer or ride found with the specified IDs. Response body contains `Error Response Object`.
    - Example Error Code: `customer_not_found`, `ride_not_found`.
  - `500 Internal Server Error`: Unexpected server error during file processing or storage.

Validation Notes

- **SSN Format:** All `customerId` and `driverId` fields must strictly adhere to the `xxx-xx-xxxx` format.
- **State:** State fields must be validated against a list of valid US state abbreviations or full names. Return `malformed_state` error if invalid.
- **Zip Code:** Zip code fields must adhere to `xxxxx` or `xxxxx-xxxx` format. Return `malformed_zipcode` error if invalid.
- **Email:** Email fields should undergo basic format validation.
- **Credit Card:** Validate expiry date (not in the past). **Crucially, implement secure handling (tokenization, etc.) in a real system and avoid storing raw card numbers and CVVs.** For simulation, ensure

basic format checks if storing temporarily. Return `invalid_credit_card` error for validation failures.
- **Rating:** Numeric ratings should be within the valid range (e.g., 1.0 to 5.0).

Security Considerations

- **Credit Card Data:** Storing raw Credit Card Numbers and CVVs is **highly insecure and violates PCI DSS compliance**. In a production environment, use a third-party payment processor and store only tokens or non-sensitive identifiers (like last 4 digits and card type). The API responses should *never* return full card numbers or CVVs. The provided data structures reflect simulation needs but must be adapted for real-world security.
- **Authentication/Authorization:** Ensure all endpoints are protected and that users can only access/modify their own data (or have appropriate admin privileges). For example, a customer should not be able to update another customer's profile. The image upload endpoint specifically needs checks to ensure the uploader is the customer associated with the ride.

Billing Service API Documentation

Version: 1.0 **Base URL:** `/api/v1` **Authentication:** Bearer Token (Specifics TBD - All endpoints should require authentication) **Content-Type:** `application/json`

Overview

The Billing Service is responsible for creating, managing, and retrieving billing information associated with completed rides. It links rides, customers, and drivers to specific financial transactions, including predicted and actual ride costs.

Note on ID Formats: The project specification requests SSN Format (xxx-xx-xxxx) for Billing ID, Ride ID, Customer ID, and Driver ID. While unconventional for Billing and Ride IDs (UUIDs are more common), this documentation adheres to the specification.

Data Structures

BillingInformation Object

Represents a billing record for a single ride.

```
{
  "billingId": "string (SSN Format: xxx-xx-xxxx)", // As per spec
```

```
  "rideId": "string (SSN Format: xxx-xx-xxxx)",    // As per spec, links to the Ride
  "customerId": "string (SSN Format: xxx-xx-xxxx)", // Links to the Customer
  "driverId": "string (SSN Format: xxx-xx-xxxx)",   // Links to the Driver
  "date": "string (ISO 8601 Date Format: YYYY-MM-DD)", // Date of the ride/billing event
  "pickupTime": "string (ISO 8601 DateTime Format)",
  "dropoffTime": "string (ISO 8601 DateTime Format)",
  "distanceCovered": "number (e.g., miles or km)",
  "sourceLocation": {
    "latitude": "number",
    "longitude": "number",
    "addressLine": "string (Optional: textual representation)" // Optional
  },
  "destinationLocation": {
    "latitude": "number",
    "longitude": "number",
    "addressLine": "string (Optional: textual representation)" // Optional
  },
  "predictedAmount": "number (Currency, e.g., USD)", // Fare prediction shown to user initia
  "actualAmount": "number (Currency, e.g., USD)", // Final calculated fare (Total amount for
  "paymentStatus": "string (Enum: PENDING, PAID, FAILED, VOID)", // Status of the payment
  "createdAt": "string (ISO 8601 Format)",
  "updatedAt": "string (ISO 8601 Format)"
}
```

Billing Input Object (for POST)

Used in the request body when creating a new bill. Often triggered internally by the Rides service upon ride completion.

```
{
  // billingId is typically generated by the service upon creation
  "rideId": "string (SSN Format: xxx-xx-xxxx)",    // Required
  "customerId": "string (SSN Format: xxx-xx-xxxx)", // Required
  "driverId": "string (SSN Format: xxx-xx-xxxx)",   // Required
  "date": "string (ISO 8601 Date Format: YYYY-MM-DD)", // Required
  "pickupTime": "string (ISO 8601 DateTime Format)", // Required
  "dropoffTime": "string (ISO 8601 DateTime Format)", // Required
  "distanceCovered": "number",                      // Required
  "sourceLocation": {                              // Required
    "latitude": "number",                          // Required
    "longitude": "number"                          // Required
  },
  "destinationLocation": {                         // Required
    "latitude": "number",                          // Required
    "longitude": "number"                          // Required
  },
```

```
  "predictedAmount": "number",                    // Required
  "actualAmount": "number"                        // Required
  // paymentStatus typically defaults to PENDING on creation
}
```

Error Response Object

Standard format for error responses.

```
{
  "error": "string (error code, e.g., 'invalid_input', 'not_found')",
  "message": "string (detailed error message)"
}
```

Endpoints

1. Create Bill

- **Description:** Creates a new billing record for a completed ride. This endpoint might be called internally by the Rides Service rather than directly by the client application.
- **Endpoint:** `POST /bills`
- **Request Body:** `Billing Input Object`
- **Responses:**
  - `201 Created`: Bill successfully created. Response body contains the created `BillingInformation Object`. Includes `Location` header: `/bills/{new_billing_id}`.
  - `400 Bad Request`: Invalid input data (e.g., missing required fields, invalid ID formats, non-numeric amounts/distance, invalid dates). Response body contains `Error Response Object`.
    - Example Error Codes: `invalid_input`, `missing_required_field`, `invalid_id_format`, `invalid_amount`, `invalid_date_format`.
  - `409 Conflict`: A billing record for the specified `rideId` might already exist (depending on business logic - can a ride be billed twice?). Or potentially if the provided `billingId` (if allowed in input) is already taken.
    - Example Error Code: `duplicate_bill_for_ride`, `duplicate_billing_id`.
  - `500 Internal Server Error`: Unexpected server error.

2. Get Bill by ID

- **Description:** Retrieves the details of a specific billing record.
- **Endpoint:** `GET /bills/{billing_id}`
- **Path Parameters:**
  - `billing_id` (string, required): The SSN format ID of the bill to retrieve.

- **Responses:**
  - `200 OK`: Bill details successfully retrieved. Response body contains the `BillingInformation Object`.
  - `400 Bad Request`: Invalid `billing_id` format provided in the path. Response body contains `Error Response Object`.
    - Example Error Code: `invalid_billing_id_format`.
  - `404 Not Found`: No bill found with the specified `billing_id`. Response body contains `Error Response Object`.
    - Example Error Code: `bill_not_found`.
  - `500 Internal Server Error`: Unexpected server error.

3. Search/List Bills

- **Description:** Retrieves a list of billing records, optionally filtered by provided criteria. Useful for admin reporting or displaying history to customers/drivers.
- **Endpoint:** `GET /bills`
- **Query Parameters (Optional):**
  - `customer_id` (string, SSN Format): Filter by customer ID.
  - `driver_id` (string, SSN Format): Filter by driver ID.
  - `ride_id` (string, SSN Format): Filter by ride ID.
  - `payment_status` (string, Enum: PENDING, PAID, FAILED, VOID): Filter by payment status.
  - `start_date` (string, YYYY-MM-DD): Filter bills on or after this date.
  - `end_date` (string, YYYY-MM-DD): Filter bills on or before this date.
  - `limit` (integer, default: 20): Maximum number of results to return (for pagination).
  - `offset` (integer, default: 0): Number of results to skip (for pagination).
- **Responses:**
  - `200 OK`: Successfully retrieved list of bills. Response body contains a JSON array of `BillingInformation Objects` matching the criteria. The array might be empty if no bills match. Include pagination headers if implemented (e.g., `X-Total-Count`).
  - `400 Bad Request`: Invalid format for query parameters (e.g., invalid ID format, invalid date format, invalid payment status enum). Response body contains `Error Response Object`.
    - Example Error Code: `invalid_query_parameter`, `invalid_id_format`, `invalid_date_format`, `invalid_payment_status`.
  - `500 Internal Server Error`: Unexpected server error.

4. Delete Bill

- **Description:** Removes a billing record from the system. **Note:** In real financial systems, deletion is rare. Marking a bill as "VOID" or

"CANCELLED" via a PATCH update is often preferred for audit trails. This endpoint is included as per the project requirements ("Delete an existing Bill"). Consider implications carefully.

- **Endpoint:** `DELETE /bills/{billing_id}`
- **Path Parameters:**
  - `billing_id` (string, required): The SSN format ID of the bill to delete.
- **Responses:**
  - `204 No Content`: Bill successfully deleted. No response body.
  - `400 Bad Request`: Invalid `billing_id` format provided in the path. Response body contains `Error Response Object`.
    - Example Error Code: `invalid_billing_id_format`.
  - `404 Not Found`: No bill found with the specified `billing_id`. Response body contains `Error Response Object`.
    - Example Error Code: `bill_not_found`.
  - `500 Internal Server Error`: Unexpected server error.
  - `403 Forbidden` (Optional): If deletion is restricted based on payment status or user roles.

Validation Notes

- **ID Formats:** All `billingId`, `rideId`, `customerId`, `driverId` fields must strictly adhere to the specified `xxx-xx-xxxx` format.
- **Dates/Times:** Validate adherence to ISO 8601 formats.
- **Numeric Fields:** Ensure `distanceCovered`, `predictedAmount`, `actualAmount` are valid numbers. Amounts should likely be non-negative.
- **Locations:** Validate latitude (-90 to 90) and longitude (-180 to 180).
- **Payment Status:** If provided in search filters or update requests, validate against the allowed enum values (PENDING, PAID, FAILED, VOID).

Considerations

- **Bill Creation Trigger:** Decide if bill creation (`POST /bills`) is triggered by the Rides Service upon ride completion or if another mechanism is used. The API is provided, but the trigger mechanism needs definition.
- **Payment Processing:** This service focuses on *recording* billing information. Actual payment processing (charging credit cards) would typically involve interaction with a separate Payment Gateway service, which might then update the `paymentStatus` on the bill (e.g., via a PATCH endpoint not detailed here, or internal updates).
- **Deletion vs. Voiding:** Carefully consider the implications of allowing hard deletion of billing records. An update mechanism to set `paymentStatus` to `VOID` is generally better practice for financial records.

Admin Service API Documentation

Version: 1.0 **Base URL:** `/api/v1/admin` **Authentication:** Bearer Token (Specifics TBD - **Strict Admin Role Required** for all endpoints) **Content-Type:** `application/json`

Overview

The Admin Service provides administrative functionalities for managing the Uber simulation system. This includes adding core entities (drivers, customers), reviewing accounts, viewing system-wide statistics and analytics, and searching billing records. Access to this service is restricted to users with administrative privileges.

Data Structures

Admin Object (Minimal)

Represents an administrator (less critical for simulation focus, but included for completeness).

```
{
  "adminId": "string (SSN Format: xxx-xx-xxxx)", // As per spec
  "firstName": "string",
  "lastName": "string",
  "email": "string (email format)",
  "createdAt": "string (ISO 8601 Format)"
  // Address, Phone etc. as needed, following project spec page 4
}
```

SystemStatistics Object

Represents aggregated statistics.

```
{
  "timePeriod": {
    "startDate": "string (YYYY-MM-DD)",
    "endDate": "string (YYYY-MM-DD)"
  },
  "areaFilter": "string (e.g., 'All', 'Zip:95123', 'City:San Jose')", // How the area was fi
  "totalRevenue": "number (Currency, e.g., USD)",
  "totalRides": "integer",
  "averageRideFare": "number (Currency, e.g., USD)",
  "averageRideDistance": "number (e.g., miles or km)"
  // Potentially add breakdowns per day/week within the period
}
```

ChartData Object

Represents data formatted for generating charts/graphs. Structure depends heavily on the charting library used.

```
{
  "chartType": "string (e.g., 'rides_per_area', 'revenue_per_driver', 'rides_over_time')",
  "labels": ["string", "string", "..."], // e.g., Area names, Driver names, Dates
  "datasets": [
    {
      "label": "string (e.g., 'Number of Rides', 'Total Revenue')",
      "data": ["number", "number", "..."] // Corresponding values
      // Add styling info if needed (colors, etc.)
    }
    // Potentially multiple datasets for comparison charts
  ]
}
```

Error Response Object

Standard format for error responses.

```
{
  "error": "string (error code, e.g., 'unauthorized', 'invalid_input', 'not_found')",
  "message": "string (detailed error message)"
}
```

(Note: This service might reuse `Driver Object`, `Customer Object`, and `BillingInformation Object` from other services for review/search responses)

Endpoints

1. Add Driver (Admin)

- **Description:** Allows an administrator to add a new driver to the system. Might have different validation rules or permissions than the public Driver Service endpoint.
- **Endpoint:** `POST /drivers`
- **Request Body:** `Driver Input Object` (from `driver_service.md`)
- **Responses:**
  - `201 Created`: Driver successfully created. Response body contains the created `Driver Object`.
  - `400 Bad Request`: Invalid input data. Response body contains `Error Response Object`.
  - `401 Unauthorized`: Request lacks valid admin credentials.
  - `403 Forbidden`: Authenticated user does not have admin privileges.

  - `409 Conflict`: Driver with this ID already exists. Response body contains `Error Response Object`.
  - `500 Internal Server Error`: Unexpected server error.

2. Add Customer (Admin)

- **Description:** Allows an administrator to add a new customer to the system.
- **Endpoint:** `POST /customers`
- **Request Body:** `Customer Input Object` (from `customer_service.md`)
- **Responses:**
  - `201 Created`: Customer successfully created. Response body contains the created `Customer Object` (sensitive CC details masked).
  - `400 Bad Request`: Invalid input data. Response body contains `Error Response Object`.
  - `401 Unauthorized`: Request lacks valid admin credentials.
  - `403 Forbidden`: Authenticated user does not have admin privileges.
  - `409 Conflict`: Customer with this ID already exists. Response body contains `Error Response Object`.
  - `500 Internal Server Error`: Unexpected server error.

3. Review Driver Account

- **Description:** Retrieves detailed information about a specific driver account for administrative review. May include more data than the public endpoint.
- **Endpoint:** `GET /drivers/{driver_id}`
- **Path Parameters:**
  - `driver_id` (string, required): The SSN format ID of the driver to review.
- **Responses:**
  - `200 OK`: Driver details retrieved. Response body contains `Driver Object` (potentially with additional admin-only fields).
  - `400 Bad Request`: Invalid `driver_id` format. Response body contains `Error Response Object`.
  - `401 Unauthorized`: Request lacks valid admin credentials.
  - `403 Forbidden`: Authenticated user does not have admin privileges.
  - `404 Not Found`: Driver not found. Response body contains `Error Response Object`.
  - `500 Internal Server Error`: Unexpected server error.

4. Review Customer Account

- **Description:** Retrieves detailed information about a specific customer account for administrative review.
- **Endpoint:** `GET /customers/{customer_id}`
- **Path Parameters:**

  - `customer_id` (string, required): The SSN format ID of the customer to review.
- **Responses:**
  - `200 OK`: Customer details retrieved. Response body contains `Customer Object` (sensitive CC details masked, potentially with additional admin-only fields).
  - `400 Bad Request`: Invalid `customer_id` format. Response body contains `Error Response Object`.
  - `401 Unauthorized`: Request lacks valid admin credentials.
  - `403 Forbidden`: Authenticated user does not have admin privileges.
  - `404 Not Found`: Customer not found. Response body contains `Error Response Object`.
  - `500 Internal Server Error`: Unexpected server error.

5. Get System Statistics

- **Description:** Retrieves aggregated statistics like total revenue and rides, filterable by date range and geographical area.
- **Endpoint:** `GET /statistics`
- **Query Parameters (Optional):**
  - `start_date` (string, YYYY-MM-DD): Start date for the statistics period. (Required if `end_date` is provided).
  - `end_date` (string, YYYY-MM-DD): End date for the statistics period. (Required if `start_date` is provided). Defaults to today if only `start_date` is given or covers all time if neither is given.
  - `area_type` (string, Enum: `city`, `zip`, `zone`, `all`): Type of area filter. Defaults to `all`.
  - `area_value` (string): The specific value for the area filter (e.g., "San Jose", "95123"). Required if `area_type` is not `all`.
- **Responses:**
  - `200 OK`: Statistics retrieved successfully. Response body contains `SystemStatistics Object`.
  - `400 Bad Request`: Invalid query parameters (e.g., invalid date format, missing `area_value`). Response body contains `Error Response Object`.
  - `401 Unauthorized`: Request lacks valid admin credentials.
  - `403 Forbidden`: Authenticated user does not have admin privileges.
  - `500 Internal Server Error`: Error during data aggregation.

6. Get Chart Data

- **Description:** Retrieves data formatted specifically for generating graphs/charts based on various criteria.
- **Endpoint:** `GET /charts`
- **Query Parameters (Required):**
  - `chart_type` (string, Enum: `rides_per_area`, `revenue_per_driver`,

`revenue_per_customer`, `rides_over_time`, ...): Specifies the type of chart data needed.

- **Query Parameters (Optional, context-dependent based on `chart_type`):**
  - `start_date` (string, YYYY-MM-DD): Start date for the data period.
  - `end_date` (string, YYYY-MM-DD): End date for the data period.
  - `area_type` (string, Enum: `city`, `zip`, `zone`, `all`): Filter data by area type.
  - `area_value` (string): Specific area value.
  - `driver_id` (string, SSN Format): Filter data for a specific driver.
  - `customer_id` (string, SSN Format): Filter data for a specific customer.
  - `time_granularity` (string, Enum: `day`, `week`, `month`): For time-series charts (e.g., `rides_over_time`). Defaults to `day`.
- **Responses:**
  - `200 OK`: Chart data retrieved successfully. Response body contains `ChartData Object`.
  - `400 Bad Request`: Invalid or missing query parameters required for the specific `chart_type`. Response body contains `Error Response Object`.
  - `401 Unauthorized`: Request lacks valid admin credentials.
  - `403 Forbidden`: Authenticated user does not have admin privileges.
  - `500 Internal Server Error`: Error during data aggregation or formatting.

7. Search Bills (Admin)

- **Description:** Allows an administrator to search for billing records using various attributes. This likely mirrors the functionality of `GET /bills` in the Billing Service but is accessed via the admin endpoint.
- **Endpoint:** `GET /bills`
- **Query Parameters:** Same as `GET /bills` in `billing_service.md` (e.g., `customer_id`, `driver_id`, `payment_status`, `start_date`, `end_date`, `limit`, `offset`).
- **Responses:**
  - `200 OK`: List of bills retrieved. Response body contains a JSON array of `BillingInformation Objects`.
  - `400 Bad Request`: Invalid query parameters. Response body contains `Error Response Object`.
  - `401 Unauthorized`: Request lacks valid admin credentials.
  - `403 Forbidden`: Authenticated user does not have admin privileges.
  - `500 Internal Server Error`: Unexpected server error.

8. **Display Bill Information (Admin)**

- **Description:** Allows an administrator to view the details of a specific bill. Mirrors `GET /bills/{billing_id}` in the Billing Service.
- **Endpoint:** `GET /bills/{billing_id}`
- **Path Parameters:**
  - `billing_id` (string, required): The SSN format ID of the bill to display.
- **Responses:**
  - `200 OK`: Bill details retrieved. Response body contains `BillingInformation Object`.
  - `400 Bad Request`: Invalid `billing_id` format. Response body contains `Error Response Object`.
  - `401 Unauthorized`: Request lacks valid admin credentials.
  - `403 Forbidden`: Authenticated user does not have admin privileges.
  - `404 Not Found`: Bill not found. Response body contains `Error Response Object`.
  - `500 Internal Server Error`: Unexpected server error.

Implementation Notes

- **Authorization:** Every endpoint in this service MUST rigorously check for administrative privileges.
- **Data Aggregation:** The statistics and charting endpoints (`/statistics`, `/charts`) require significant backend logic to query and aggregate data, potentially across multiple database tables or even services. Performance optimization (indexing, caching, potentially pre-calculated summaries) will be crucial.
- **Area Definition:** The system needs a consistent way to define "area" (e.g., based on zip codes, city boundaries, or predefined zones) for the statistics and charting endpoints to function correctly.
- **Service Interaction:** Consider whether the Admin service directly accesses the database tables of other services or communicates with the other services (Driver, Customer, Billing, Rides) via their APIs or Kafka events to gather data. Direct DB access might be simpler for reads but tightly couples the services. Service-to-service communication is generally preferred in microservice architectures.

Rides Service API Documentation

Version: 1.0 **Base URL:** `/api/v1` (Example, adjust as needed) **Authentication:** Bearer Token (Specifics TBD - All endpoints should require authentication, usually Customer or Driver context) **Content-Type:** `application/json`

Overview

The Rides Service manages the lifecycle of a ride request and the subsequent trip. It handles ride creation (requests), matching customers with nearby drivers, tracking ride status, retrieving ride details, and managing cancellations. It interacts with other services like Pricing (for fare estimation), Driver Service (for location and status), Customer Service, and potentially triggers events for Billing and Rating upon completion.

Note on ID Formats: The project specification requests SSN Format (xxx-xx-xxxx) for Ride ID, Customer ID, and Driver ID. This documentation adheres to that specification.

Data Structures

Ride Object

Represents a ride instance in the system.

```
{
  "rideId": "string (SSN Format: xxx-xx-xxxx)", // As per spec
  "customerId": "string (SSN Format: xxx-xx-xxxx)",
  "driverId": "string (SSN Format: xxx-xx-xxxx) | null", // Assigned after matching
  "pickupLocation": {
    "latitude": "number",
    "longitude": "number",
    "addressLine": "string (Optional: textual representation)" // Optional
  },
  "dropoffLocation": {
    "latitude": "number",
    "longitude": "number",
    "addressLine": "string (Optional: textual representation)" // Optional
  },
  "status": "string (Enum: REQUESTED, ACCEPTED, DRIVER_ARRIVED, IN_PROGRESS, COMPLETED, CANC
  "requestTimestamp": "string (ISO 8601 Format)",
  "acceptTimestamp": "string (ISO 8601 Format) | null",
  "pickupTimestamp": "string (ISO 8601 Format) | null", // When ride actually starts
  "dropoffTimestamp": "string (ISO 8601 Format) | null", // When ride ends
  "predictedFare": "number (Currency, e.g., USD) | null", // Calculated at request time
  "actualFare": "number (Currency, e.g., USD) | null", // Calculated after completion
  "distance": "number (e.g., miles or km) | null", // Calculated after completion
  "createdAt": "string (ISO 8601 Format)",
  "updatedAt": "string (ISO 8601 Format)"
}
```

Ride Request Input Object (for POST)

Used by a customer to request a new ride.

```
{
  // customerId is typically derived from the authenticated user token
  "pickupLocation": {          // Required
    "latitude": "number",      // Required
    "longitude": "number"    // Required
  },
  "dropoffLocation": {         // Required
    "latitude": "number",      // Required
    "longitude": "number"    // Required
  }
  // passenger_count might be added here if needed for pricing/matching
}
```

Nearby Driver Object (Response from GET /drivers/nearby)

Simplified representation of a driver for matching purposes.

```
{
    "driverId": "string (SSN Format)",
    "currentLocation": {
        "latitude": "number",
        "longitude": "number"
    },
    "estimatedArrivalTimeSeconds": "number | null", // Optional: calculated ETA to pickup
    "rating": "number (float)", // Optional: for matching preference
    "carDetails": { // Optional: for display/preference
        "make": "string",
        "model": "string",
        "color": "string"
    }
}
```

Error Response Object

Standard format for error responses.

```
{
  "error": "string (error code, e.g., 'invalid_input', 'not_found', 'no_drivers', 'ride_not_
  "message": "string (detailed error message)"
}
```

Endpoints

1. Request Ride

- **Description:** A customer requests a new ride from their current location to a destination. This initiates the driver matching process.
- **Endpoint:** `POST /rides`
- **Request Body:** `Ride Request Input Object`
- **Responses:**
  - `202 Accepted`: Ride request successfully received and driver matching initiated. Response body contains the initial `Ride Object` with status `REQUESTED` and a `predictedFare`.
  - `400 Bad Request`: Invalid input data (e.g., missing locations, invalid coordinates). Response body contains `Error Response Object`.
    - Example Error Codes: `invalid_input`, `missing_location`, `invalid_coordinates`.
  - `401 Unauthorized`: Customer authentication failed.
  - `404 Not Found` (or a specific 4xx/5xx): Could indicate failure during initial processing like price prediction or inability to even start the matching process. Could also represent "No drivers available nearby" if checked synchronously (though often handled asynchronously).
    - Example Error Code: `pricing_error`, `service_unavailable`.
  - `503 Service Unavailable`: If the matching system or dependent services (like Pricing, Driver location) are temporarily unavailable.
  - `500 Internal Server Error`: Unexpected server error during request processing or matching initiation.

2. Get Ride Details

- **Description:** Retrieves the current status and details of a specific ride. Accessible by the customer or the assigned driver (or admin).
- **Endpoint:** `GET /rides/{ride_id}`
- **Path Parameters:**
  - `ride_id` (string, required): The SSN format ID of the ride to retrieve.
- **Responses:**
  - `200 OK`: Ride details successfully retrieved. Response body contains the `Ride Object`.
  - `400 Bad Request`: Invalid `ride_id` format. Response body contains `Error Response Object`.
    - Example Error Code: `invalid_ride_id_format`.
  - `401 Unauthorized`: Authentication failed.
  - `403 Forbidden`: Authenticated user is not the customer or assigned driver for this ride (and not an admin).
  - `404 Not Found`: No ride found with the specified `ride_id`. Response body contains `Error Response Object`.
    - Example Error Code: `ride_not_found`.
  - `500 Internal Server Error`: Unexpected server error.

3. **List Rides**

- **Description:** Retrieves a list of rides associated with the authenticated user (customer or driver), optionally filtered.
- **Endpoint:** `GET /rides`
- **Query Parameters (Optional):**
  - `status` (string, Enum: See Ride Object status): Filter by ride status. Can potentially accept multiple statuses (e.g., `status=COMPLETED,CANCELLED`).
  - `limit` (integer, default: 20): Maximum number of results to return (for pagination).
  - `offset` (integer, default: 0): Number of results to skip (for pagination).
  - `start_date` (string, YYYY-MM-DD): Filter rides requested on or after this date.
  - `end_date` (string, YYYY-MM-DD): Filter rides requested on or before this date.
  - `for_customer_id` (string, SSN Format - Admin Only): Filter by customer ID (requires admin privileges).
  - `for_driver_id` (string, SSN Format - Admin Only): Filter by driver ID (requires admin privileges).
- **Responses:**
  - `200 OK`: Successfully retrieved list of rides. Response body contains a JSON array of `Ride Objects` matching the criteria. The array might be empty. Include pagination headers if implemented (e.g., `X-Total-Count`).
  - `400 Bad Request`: Invalid format for query parameters (e.g., invalid status enum, invalid date format). Response body contains `Error Response Object`.
    - Example Error Code: `invalid_query_parameter`.
  - `401 Unauthorized`: Authentication failed.
  - `403 Forbidden`: Non-admin user attempting to use `for_customer_id` or `for_driver_id`.
  - `500 Internal Server Error`: Unexpected server error.

4. **Cancel Ride (Delete Ride)**

- **Description:** Cancels an active ride request or an accepted ride *before* it enters the `IN_PROGRESS` state. Fulfills the "Delete an existing ride" requirement.
- **Endpoint:** `DELETE /rides/{ride_id}`
- **Path Parameters:**
  - `ride_id` (string, required): The SSN format ID of the ride to cancel.
- **Responses:**
  - `204 No Content`: Ride successfully cancelled.
  - `400 Bad Request`: Invalid `ride_id` format. Response body contains

`Error Response Object`.
    * Example Error Code: `invalid_ride_id_format`.
  - `401 Unauthorized`: Authentication failed.
  - `403 Forbidden`: Authenticated user is not authorized to cancel this ride (e.g., not the customer or assigned driver in an appropriate state).
  - `404 Not Found`: No ride found with the specified `ride_id`. Response body contains `Error Response Object`.
    * Example Error Code: `ride_not_found`.
  - `409 Conflict`: The ride is not in a cancellable state (e.g., already `COMPLETED` or `IN_PROGRESS`). Response body contains `Error Response Object`.
    * Example Error Code: `ride_not_cancellable`.
  - `500 Internal Server Error`: Unexpected server error.

5. Find Nearby Drivers

- **Description:** Retrieves a list of available drivers within a specified radius of a given location. Useful for the client UI to show nearby cars or for the backend matching algorithm. Fulfills "Display the location of drivers within 10 miles".
- **Endpoint:** `GET /drivers/nearby`
- **Query Parameters (Required):**
  - `latitude` (number): Latitude of the center point (e.g., customer pickup location).
  - `longitude` (number): Longitude of the center point.
- **Query Parameters (Optional):**
  - `radius_miles` (number, default: 10): Search radius in miles.
- **Responses:**
  - `200 OK`: Successfully retrieved list of nearby available drivers. Response body contains a JSON array of `Nearby Driver Objects`. The array might be empty.
  - `400 Bad Request`: Invalid or missing latitude/longitude, invalid radius. Response body contains `Error Response Object`.
    * Example Error Code: `invalid_coordinates`, `missing_coordinates`.
  - `401 Unauthorized`: Authentication failed (if required for this endpoint).
  - `500 Internal Server Error`: Unexpected server error querying driver locations.

State Transitions & Events (Conceptual)

The `status` field follows a state machine. Transitions are triggered by events (often via Kafka or direct internal calls):

1. `POST /rides` -> Creates Ride with status `REQUESTED`.

2. Matching Algo finds driver -> Driver Accepts (via Driver App interaction, potentially another API endpoint not listed here) -> Status changes to `ACCEPTED`. -> *Event: RideAccepted* (Payload: rideId, driverId, customerId).
3. Driver reaches pickup -> Driver marks arrival (via Driver App) -> Status changes to `DRIVER_ARRIVED`. -> *Event: DriverArrived* (Payload: rideId, driverId).
4. Customer gets in, ride starts -> Driver starts trip (via Driver App) -> Status changes to `IN_PROGRESS`. -> *Event: RideStarted* (Payload: rideId).
5. Ride finishes -> Driver ends trip (via Driver App) -> Status changes to `COMPLETED`. -> Calculate `actualFare` & `distance`. -> *Event: RideCompleted* (Payload: rideId, customerId, driverId, actualFare, distance, start/end times & locations). -> Billing Service consumes this. Rating mechanism might be triggered.
6. `DELETE /rides/{ride_id}` called by Customer before ACCEPTED/IN_PROGRESS -> Status changes to `CANCELLED_CUSTOMER`. -> *Event: RideCancelled* (Payload: rideId, cancelledBy="customer").
7. Driver cancels before pickup (via Driver App) -> Status changes to `CANCELLED_DRIVER`. -> *Event: RideCancelled* (Payload: rideId, cancelledBy="driver"). -> Potentially re-initiate matching.
8. Matching Algo fails after timeout -> Status changes to `NO_DRIVERS_AVAILABLE`. -> *Event: MatchingFailed* (Payload: rideId, customerId).

Considerations

- **Matching Logic:** The core driver matching algorithm is complex and happens behind the scenes after `POST /rides`. It needs efficient querying of driver locations and availability (potentially using Geo-spatial indexing and the Driver Service or a dedicated location cache).
- **Driver Actions:** Driver interactions like accepting a ride, marking arrival, starting/ending the trip are typically handled via endpoints exposed specifically for the driver application, not usually listed in the primary customer-facing API docs. These actions trigger status updates and events within the Rides Service.
- **Pricing Integration:** Needs to call the dynamic pricing logic/service during the `POST /rides` request to get the `predictedFare` and after completion to calculate the `actualFare` (or receive it in the `RideCompleted` event payload).
- **Kafka Integration:** This service is a prime candidate for producing events (RideAccepted, RideCompleted, etc.) and potentially consuming events (e.g., DriverLocationUpdated).
- **Concurrency:** Handling multiple simultaneous ride requests, driver location updates, and status changes requires careful design to avoid race conditions.