**Table Of Content**

[Tech Stack 1](#_Toc8129)

[Assumptions 2](#_Toc26896)

[Entity Relationship Diagram (Data Model) 3](#_Toc15777)

[1. The User Table 3](#_Toc19876)

[2. The Ride Request Table 3](#_Toc22249)

[3. The Ride Table 4](#_Toc10208)

[4. The Ride Rejection Table 4](#_Toc25756)

[5. The Payment Table 4](#_Toc32177)

[API Endpoint Overview 7](#_Toc27140)

[Real-Time Features 16](#_Toc19978)

[Error Handling 17](#_Toc30696)

**Design Document**

### ****Tech Stack****

**Backend:**  
**Node.js with Express**  
**Why:** Node.js is lightweight and highly efficient for building RESTful APIs. Express provides a simple interface for routing and middleware management, making it an excellent choice for rapid prototyping.

**Database:**  
**SQLite**  
**Why:** SQLite is a serverless, lightweight database well-suited for prototyping. It requires minimal configuration and integrates seamlessly with Node.js.

**Authentication:**  
**Basic Authentication**   
**Why:** Basic Auth is straightforward to implement and ideal for testing API endpoints during the development phase. It suits the current scope of this project. For Security Purpose I use hash passwords with bcrypt when users register, store the hash, and compare the hash with the entered password during login.

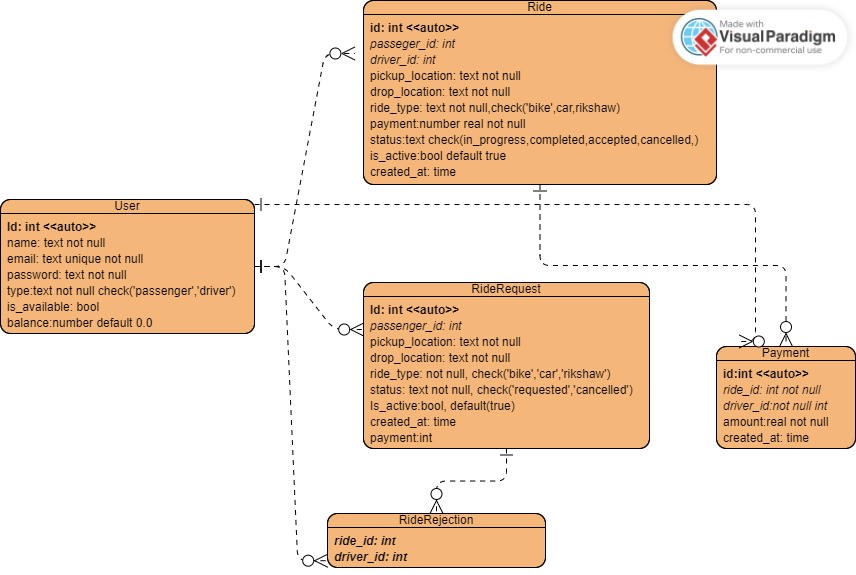
### ****Assumptions****

**Driver Availability Feature:**  
An additional feature has been added to manage driver availability. If a driver sets their status to "unavailable", they will not be able to view or accept ride requests.

**Single Active Ride Request per Passenger:**  
During development, it was observed that passengers could send multiple ride requests simultaneously. To address this, a mechanism was implemented to ensure that any existing request must be cancelled before a new one can be submitted.

**In-Memory Database Usage:**  
SQLite is configured to run in-memory for simplicity and ease of testing.

### ****Entity Relationship Diagram (Data Model)****



**Explantion of ERD:**

### 1. The User Table

This is the starting point—think of it as the people using the app. Every user gets their own spot here:

* **ID**: A unique number for each person, like a personal ID card, that automatically goes up (1, 2, 3, ...) as new users join.
* **Name**: Their name, because we need to know who they are!
* **Email**: Their email address, which has to be unique—no two people can use the same one.
* **Password**: A secret code they use to log in, which they can’t skip entering.
* **Type**: This tells us if they’re a passenger (looking for a ride) or a driver (offering a ride).
* **Is Available**: A yes/no flag, but only for drivers—shows if they’re ready to take a ride request.
* **Balance**: A running total of money earned, but this only grows for drivers when they complete rides. Passengers don’t use it (it stays at zero for them).

### 2. The Ride Request Table

This is where the ride journey begins. When a passenger wants a ride, they create a request, and it lands here:

* **ID**: Another unique number that auto-increments, like a ticket number for each request.
* **Passenger ID**: Links back to a user (the passenger who made the request).
* **Pickup Location**: Where they want to be picked up—gotta know the starting point!
* **Drop Location**: Where they’re heading.
* **Ride Type**: What kind of ride they want—bike, car, or rickshaw.
* **Payment**: How much they’re willing to pay, a flexible number they set (as long as it’s positive).
* **Status**: Either “requested” (waiting for a driver) or “cancelled” (they changed their mind).
* **Is Active**: A yes/no flag to show if the request is still open—passengers can only have one active request at a time.
* **Created At**: The timestamp when the request was made.

### 3. The Ride Table

Once a driver accepts a request, it moves to this box, turning into an active ride:

* **ID**: A unique number that auto-increments for each ride.
* **Passenger ID**: Links to the user who requested the ride.
* **Driver ID**: Links to the user (driver) who took the ride—this one’s required.
* **Pickup Location, Drop Location, Ride Type**: Same details as the request, carried over.
* **Payment**: The amount agreed upon, transferred from the ride request.
* **Status**: Can be “accepted” (driver’s on board), “in\_progress” (ride’s happening), “completed” (ride’s done), or “cancelled” (something went wrong).
* **Is Active**: Another yes/no flag—only one active ride per passenger or driver at a time.
* **Created At**: When the ride started.

### 4. The Ride Rejection Table

Sometimes a driver isn’t interested in a request, and this box keeps track of those “no thanks” moments:

* **Ride ID**: Links to the specific request they rejected.
* **Driver ID**: Links to the driver who said no.
* (No auto-incrementing ID here—it uses a combo of ride\_request\_id and driver\_id as the unique key.)

### 5. The Payment Table

When a ride is done, the money part gets recorded here:

* **ID**: A unique number that auto-increments for each payment.
* **Ride ID**: Links to the ride that was completed.
* **Driver ID**: Links to the driver who earned the money.
* **Amount**: The payment amount from the ride, recorded when it’s completed.
* **Created At**: When the payment was logged.

****Work Flow:****

· **User Registration**: Anyone can sign up as a passenger or driver with a name, email, and password, starting with a zero balance (for drivers only).

· **Login**: Users log in with their email and password to access the system securely.

· **Ride Request**: Passengers can request a ride by setting a pickup location, drop location, ride type (bike, car, or rickshaw), and a payment amount—only one active request at a time.

· **View Available Rides**: Drivers can see all open ride requests they haven’t rejected, but only if they’re marked as available.

· **Accept a Ride**: Drivers can accept a ride request, moving it to an active ride status—only one active ride per driver at a time.

· **Reject a Ride**: Drivers can reject a ride request, which gets logged so they won’t see it again.

· **Update Ride Status**: Drivers can change a ride’s status to “in\_progress,” “completed,” or “cancelled,” with payment processed and balance updated upon completion.

· **Cancel a Ride/Request**: Passengers can cancel their active ride request or accepted ride, marking it as inactive.

· **View Current Ride/Request**: Passengers can check their current active ride or request details.

· **View Ride History**: Passengers can see a history of their cancelled requests and completed rides, including payment amounts.

· **Check Driver Balance**: Drivers can view their total earnings (balance) updated after each completed ride.

· **Toggle Driver Availability**: Drivers can switch their availability on or off to accept or pause ride requests.

· **Track Total Payments History**: Drivers can check the history of all payments from the Payment table for verification.

### ****API Endpoint Overview****

#### 1. ****Register Users****

**Endpoint:** POST /register

**Purpose:** Register new users.

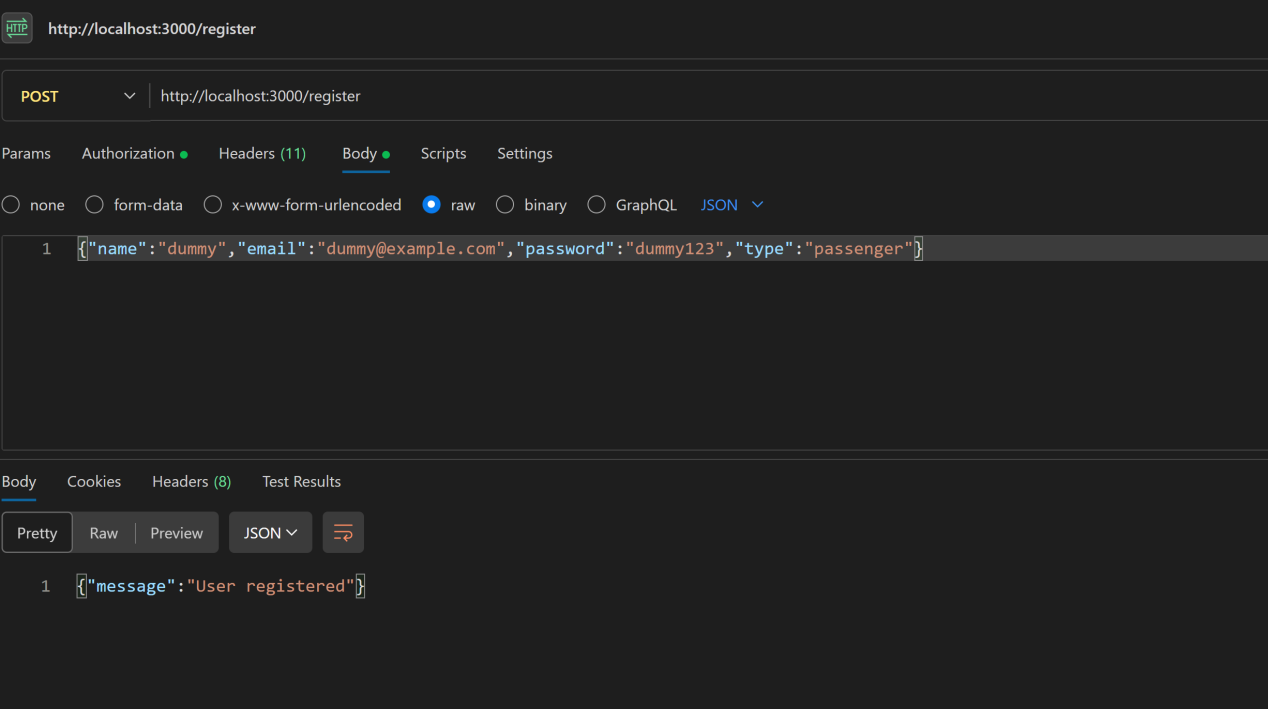
**Authentication:** Not required.

**Postman Setup:**

Method: POST

URL: http://localhost:3000/register

Body: JSON with four required fields.



#### 2. ****Test Login (Verify Basic Auth)****

**Endpoint:** POST /login

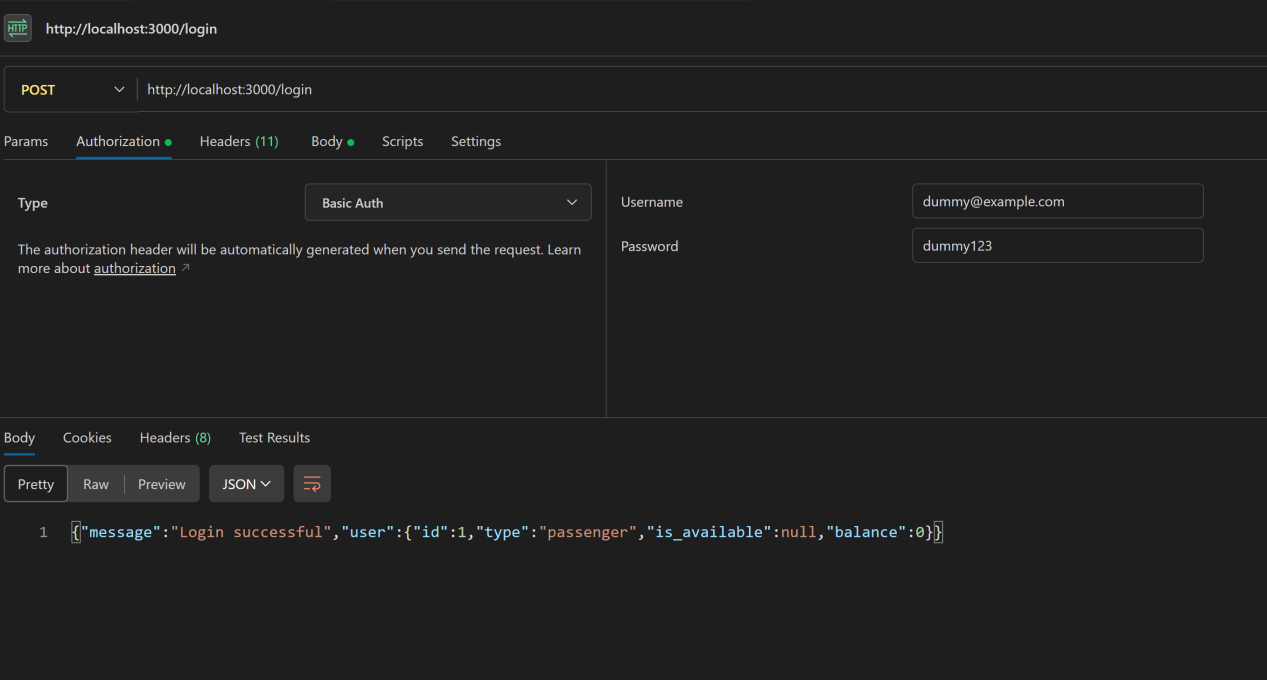
**Purpose:** Verify Basic Auth credentials.

**Postman Setup:**

Method: POST

URL: http://localhost:3000/login

**Authentication**: Select “Basic Auth” and provide credentials.



#### 3. ****Request a Ride (Passenger)****

**Endpoint:** POST /rides

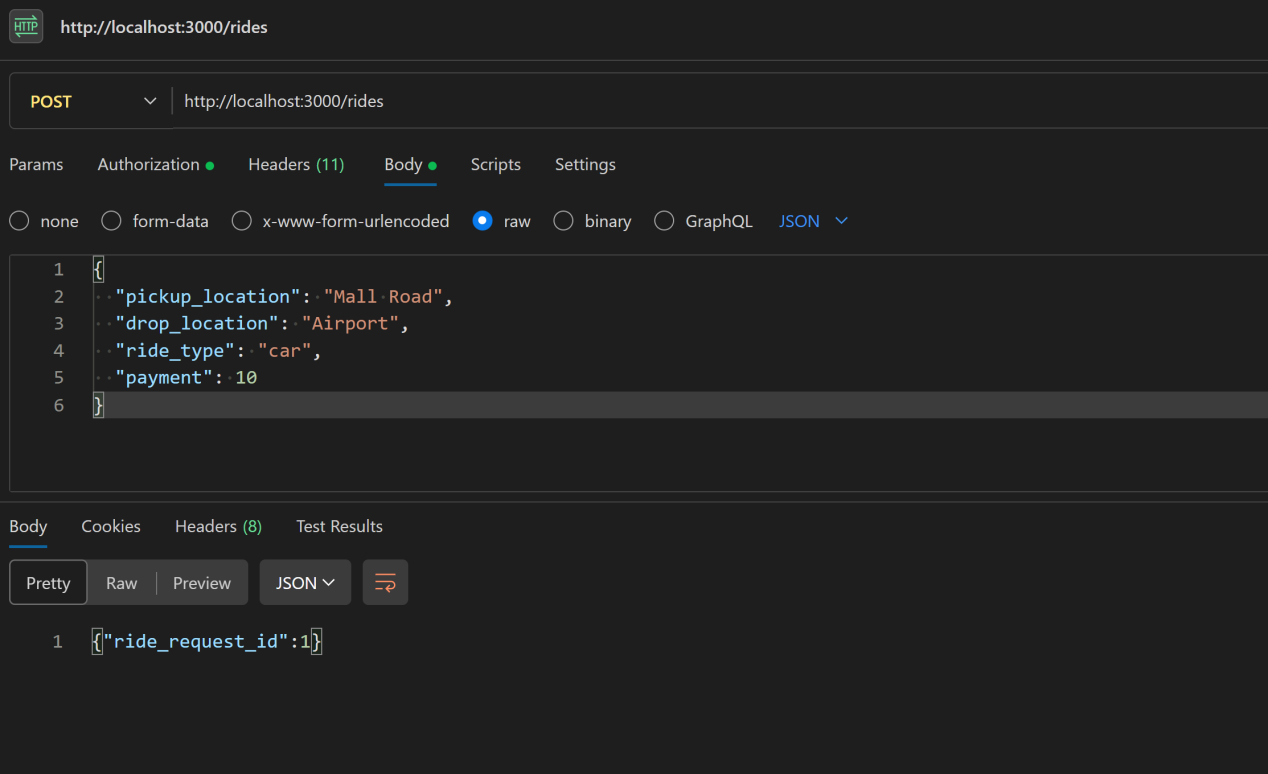
**Purpose:** Submit a ride request.

**Postman Setup:**

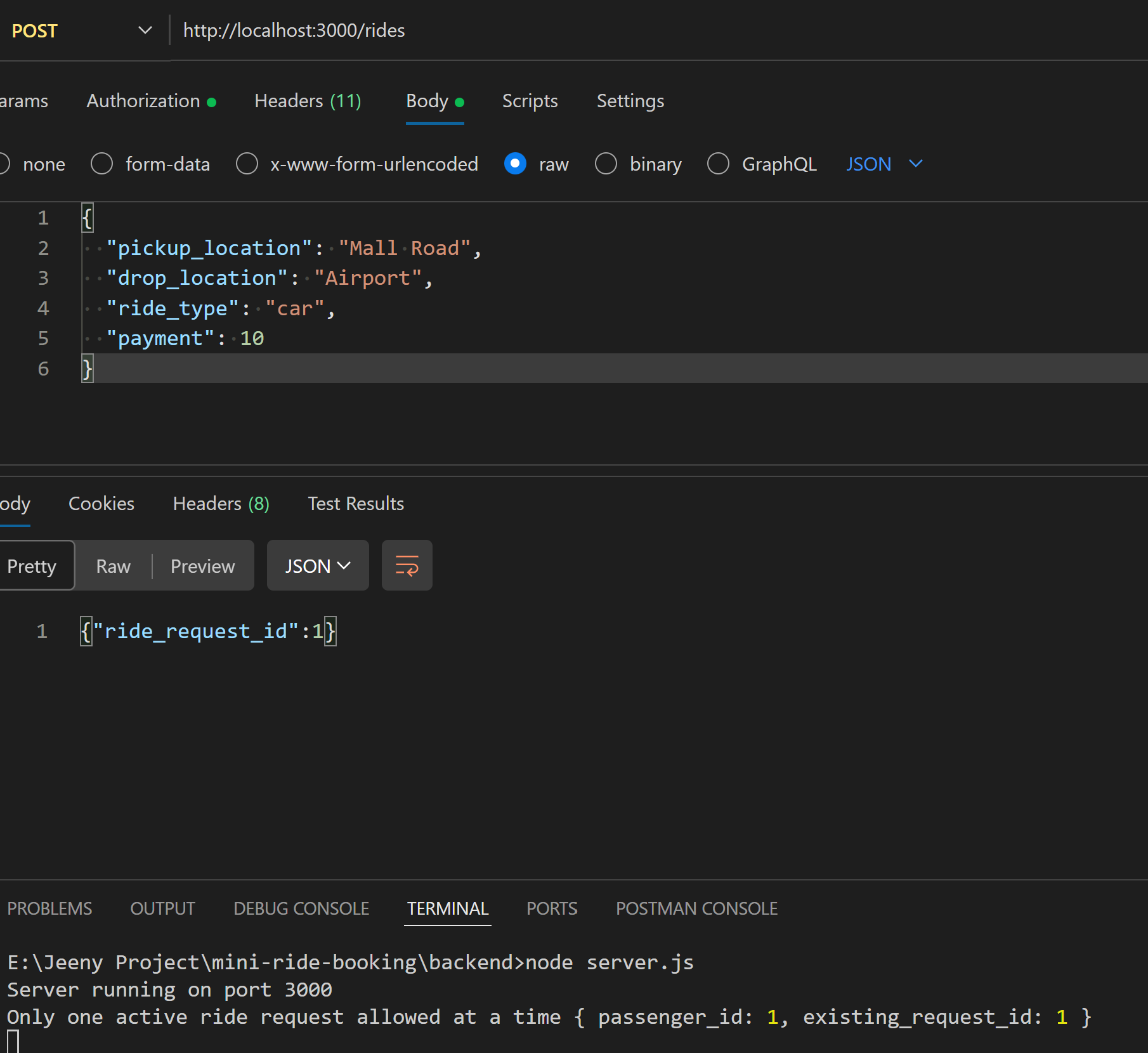
Method: POST

URL: http://localhost:3000/rides

Body: Raw JSON with pickup, destination, and ride\_type.



If a passenger create a ride request again he/she will not be able to do that because only one request at a time.



#### 4. ****View Available Rides (Driver)****

**Endpoint:** GET /rides/available

**Purpose:** Allow drivers to view available ride requests.

**Postman Setup:**

Method: GET

URL: http://localhost:3000/rides/available

**Authentication**: Basic Auth using driver credentials.



#### 5. ****Toggle Driver Availability****

**Endpoint:** PUT /users/availability

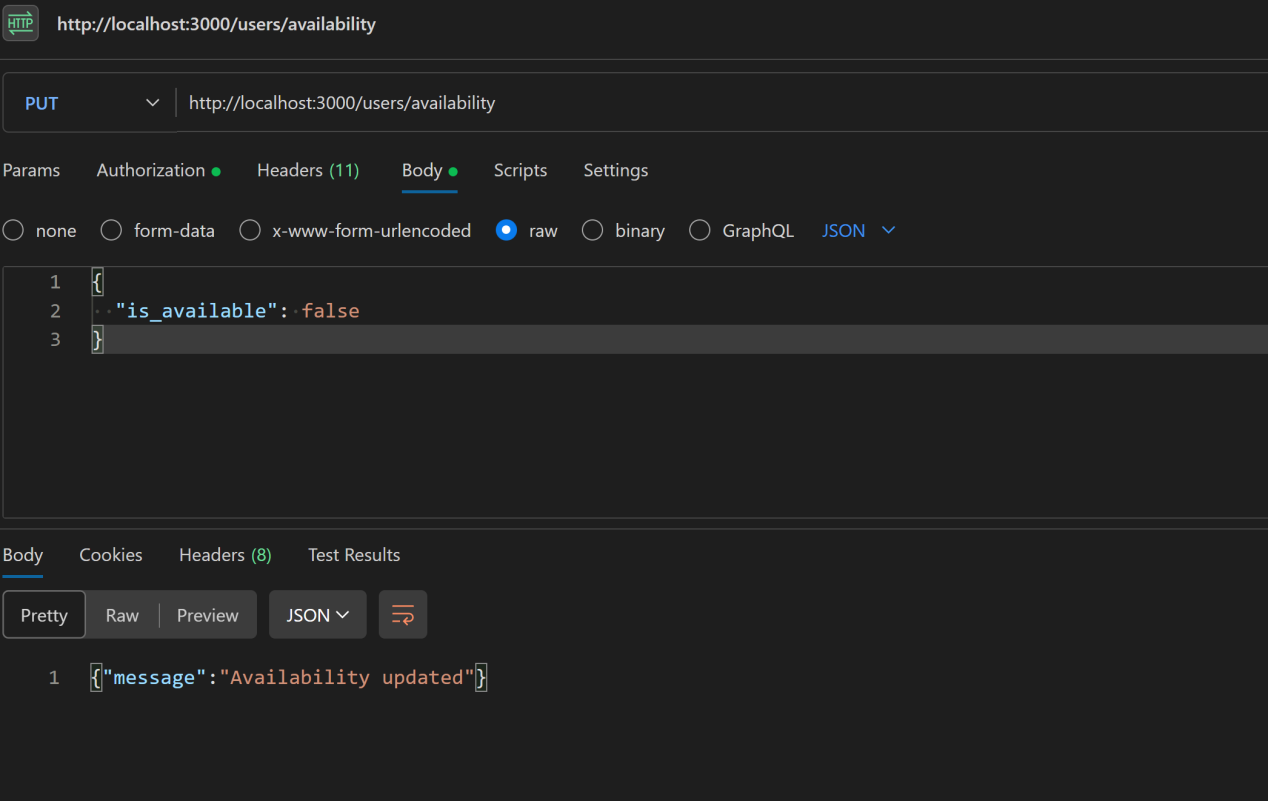
**Purpose:** Enable or disable driver availability.

**Postman Setup:**

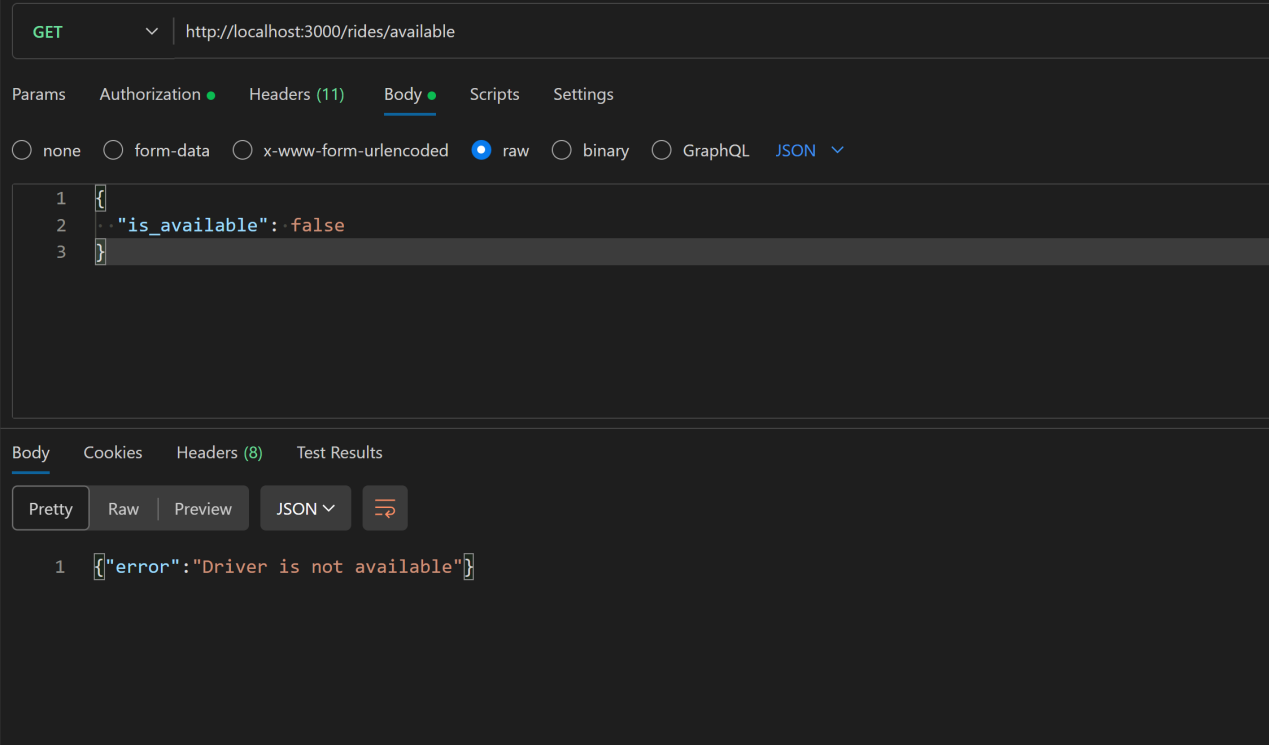
Method: PUT

URL: http://localhost:3000/users/availability

**Authentication**: Driver login credentials.



**Note:** When a driver marks themselves as unavailable, they will not see new ride requests. Once they toggle back to available, they can view and accept new requests.



#### 6. ****Accept a Ride (Driver)****

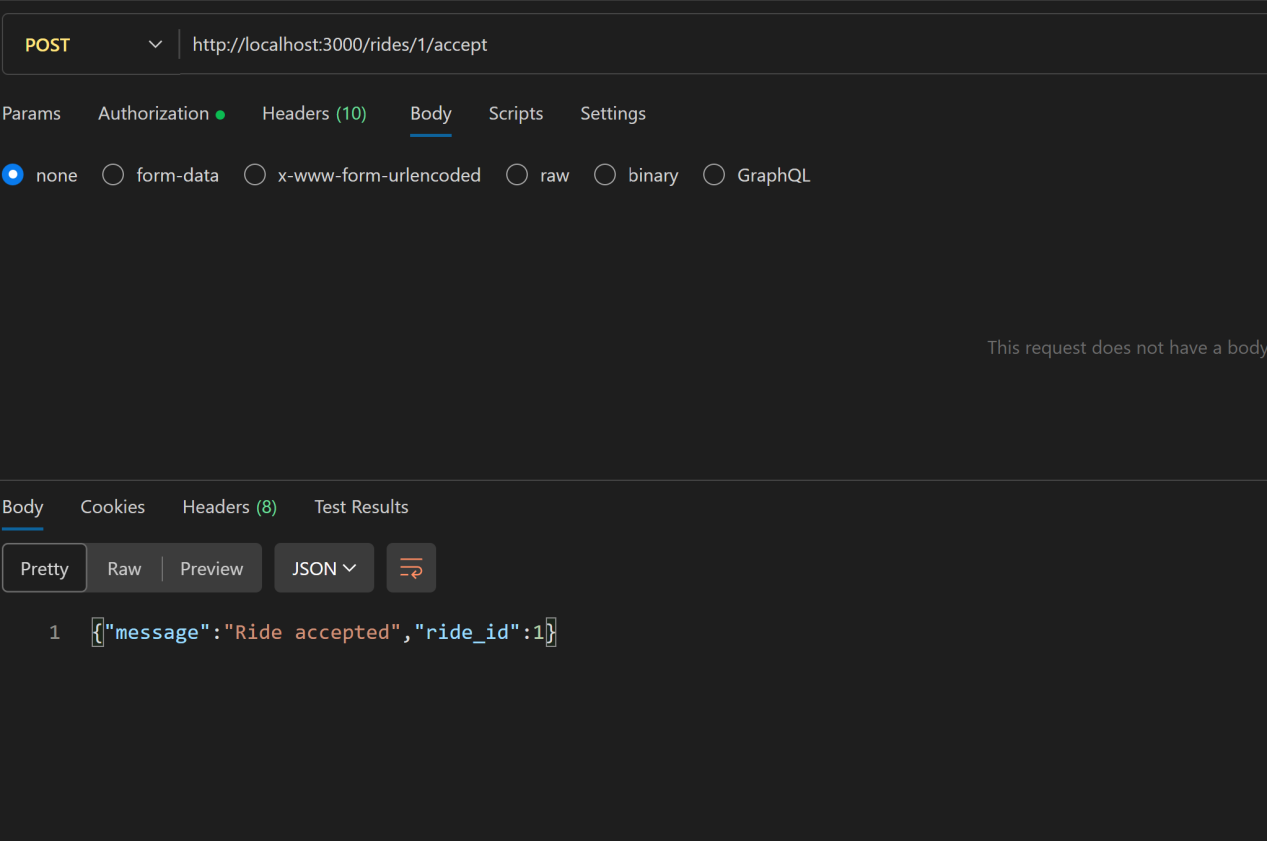
**Endpoint:** POST /rides/:id/accept

**Purpose:** Driver accepts a specific ride.

**Postman Setup:**

Method: POST

URL: <http://localhost:3000/rides/1/accept>



#### 7. ****Update Ride Status (Driver)****

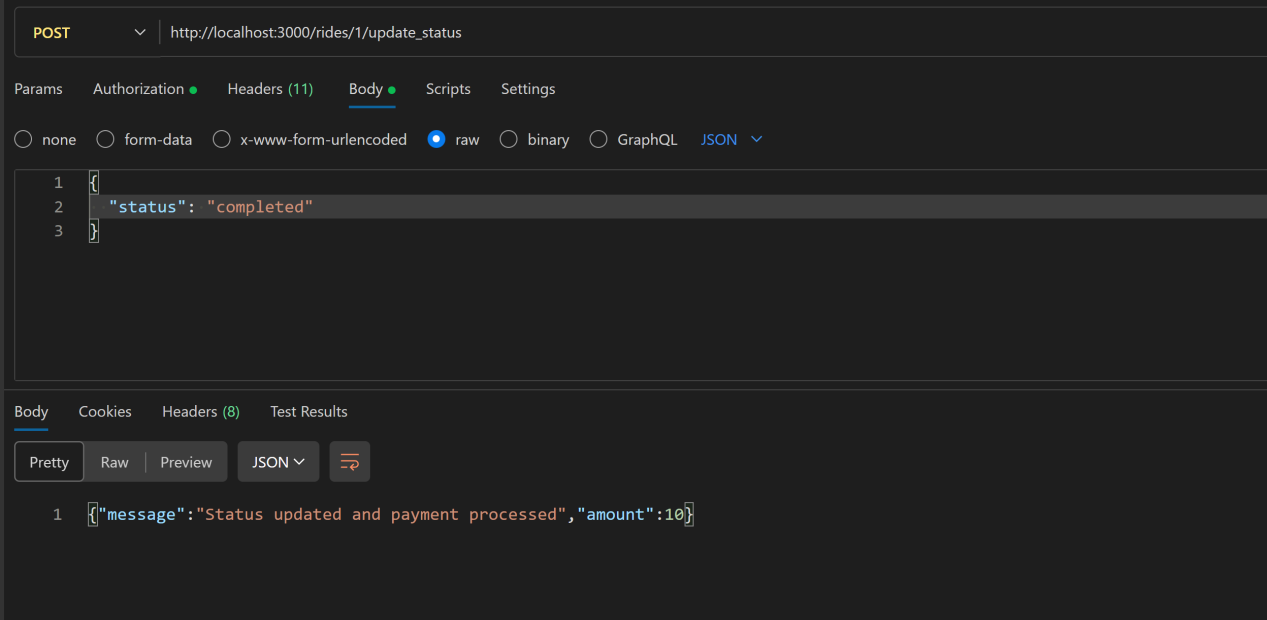
**Endpoint:** POST /rides/:id/update\_status

**Purpose:** Driver updates the status of a ride to in\_progress or completed.

**Postman Setup:**

Method: POST

URL: <http://localhost:3000/rides/1/update_status>



#### 8. ****Cancel a Ride Request (Passenger)****

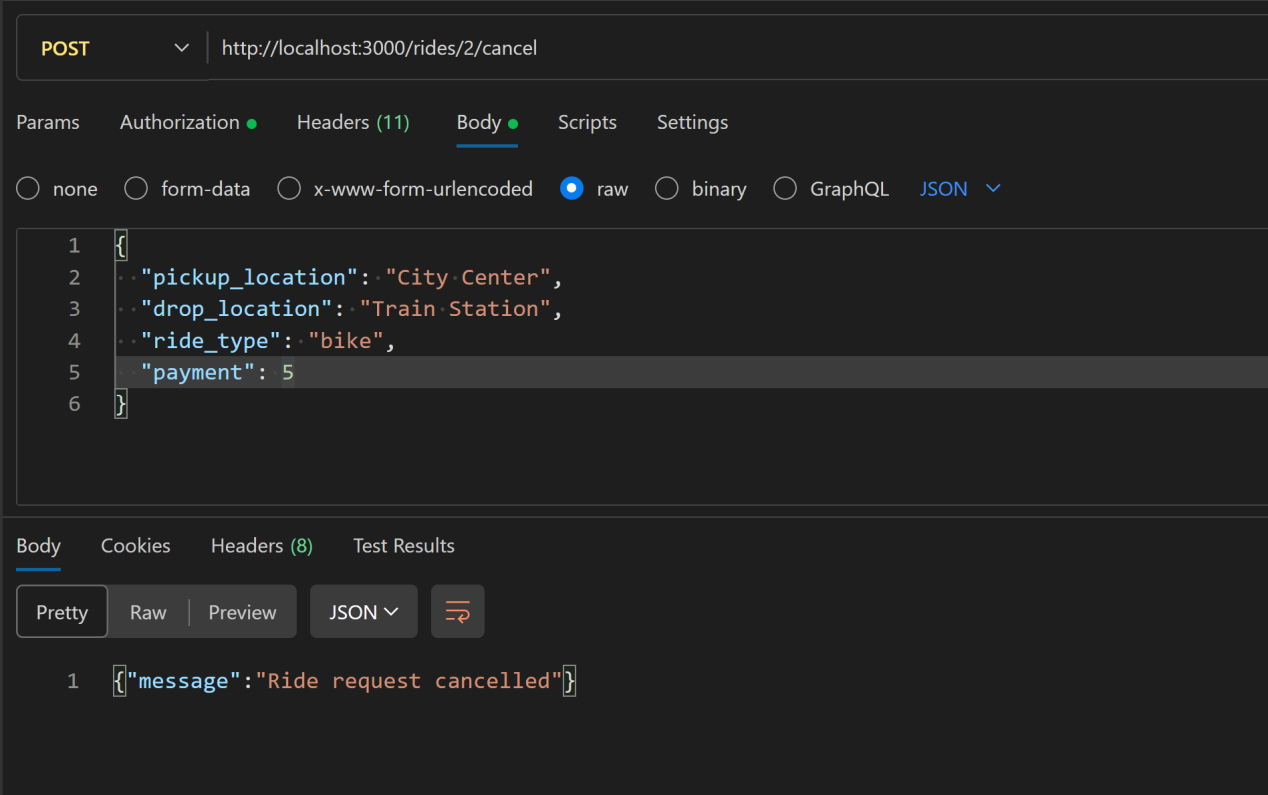
**Endpoint:** POST /rides/:id/cancel

**Purpose:** Passenger cancels a ride.

**Postman Setup:**

Method: POST

URL: <http://localhost:3000/rides/2/cancel>



#### 9. ****View Current Ride (Passenger)****

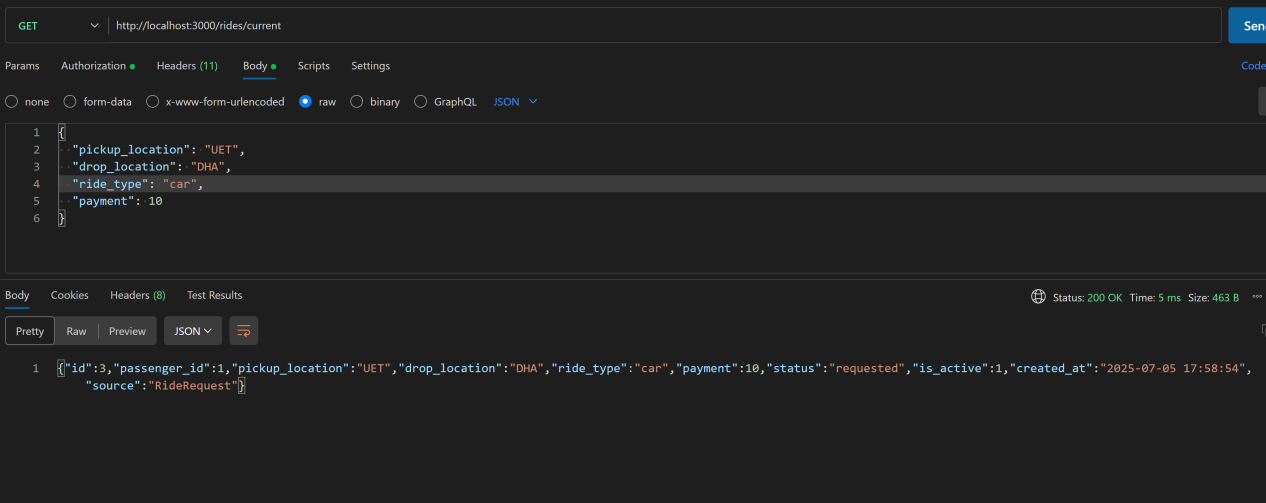
**Endpoint:** GET /rides/current

**Purpose:** View the current ride (returns none if no active ride).

**Postman Setup:**

Method: GET

URL: <http://localhost:3000/rides/current>



#### 10. ****View Ride History (Passenger)****

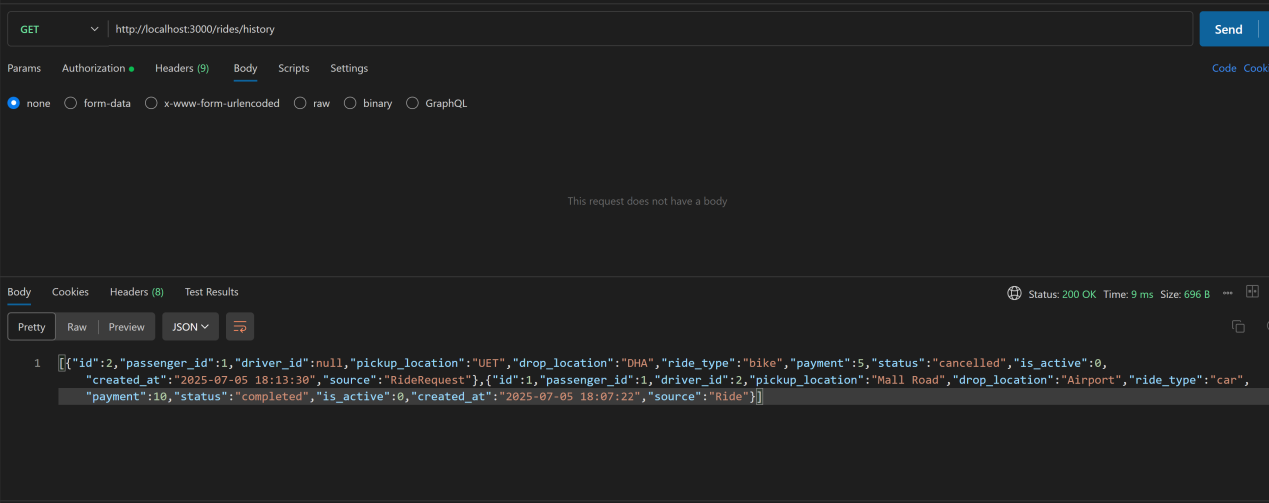
**Endpoint:** GET /rides/history

**Purpose:** Retrieve ride history for a passenger.

**Postman Setup:**

Method: GET

URL: <http://localhost:3000/rides/history>



#### 11. ****Optional Feature: Reject Ride (Driver)****

**Endpoint:** POST /rides/:id/reject

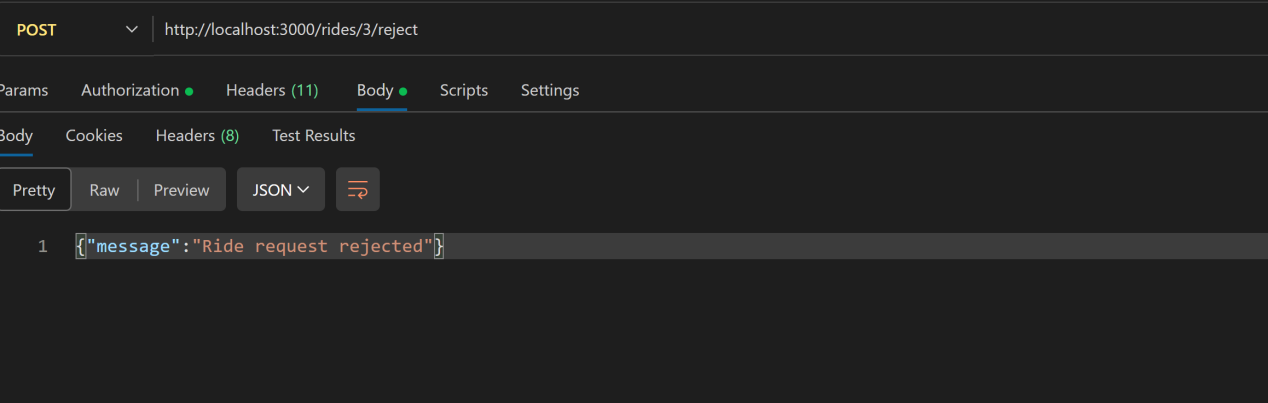
**Purpose:** Driver can reject a ride request.

**Postman Setup:**

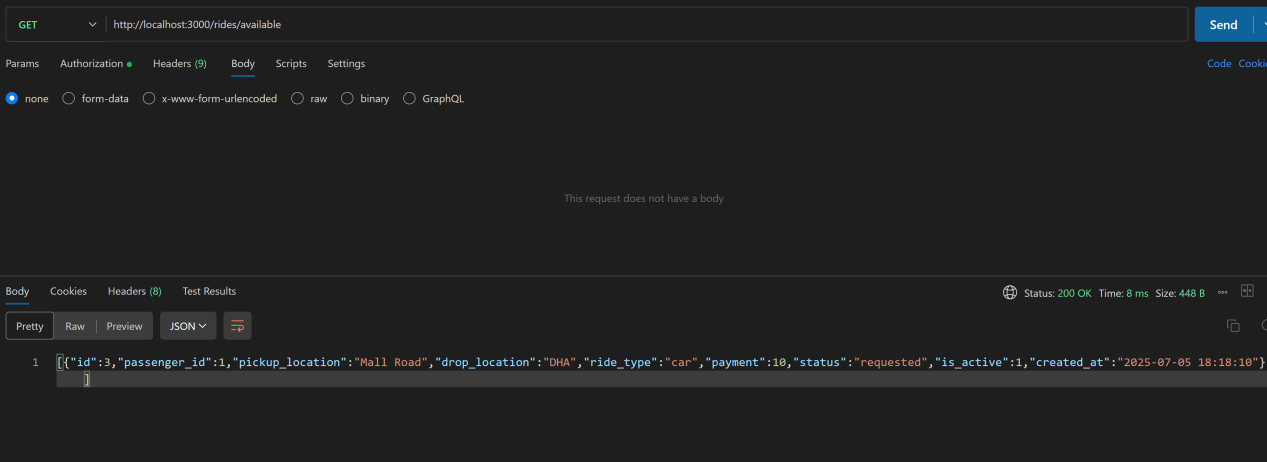
Method: POST

URL: http://localhost:3000/rides/3/reject

**Authentication**: Driver credentials

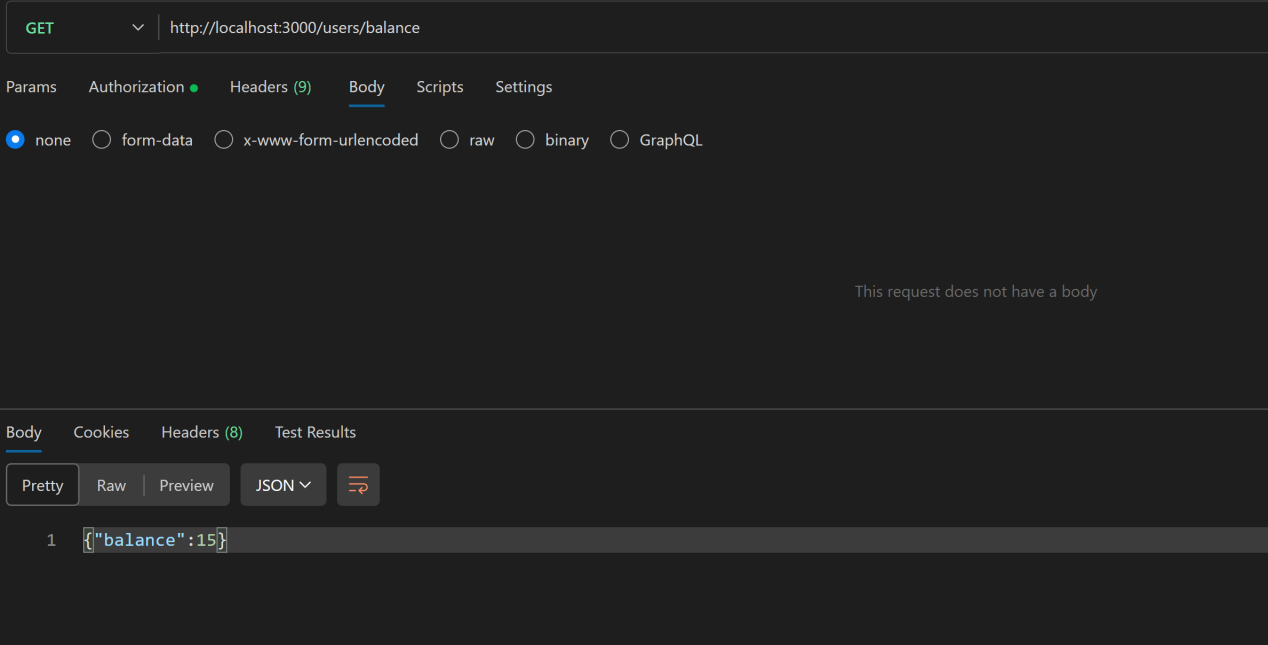


Note: Once a ride is rejected, it becomes available for other drivers.



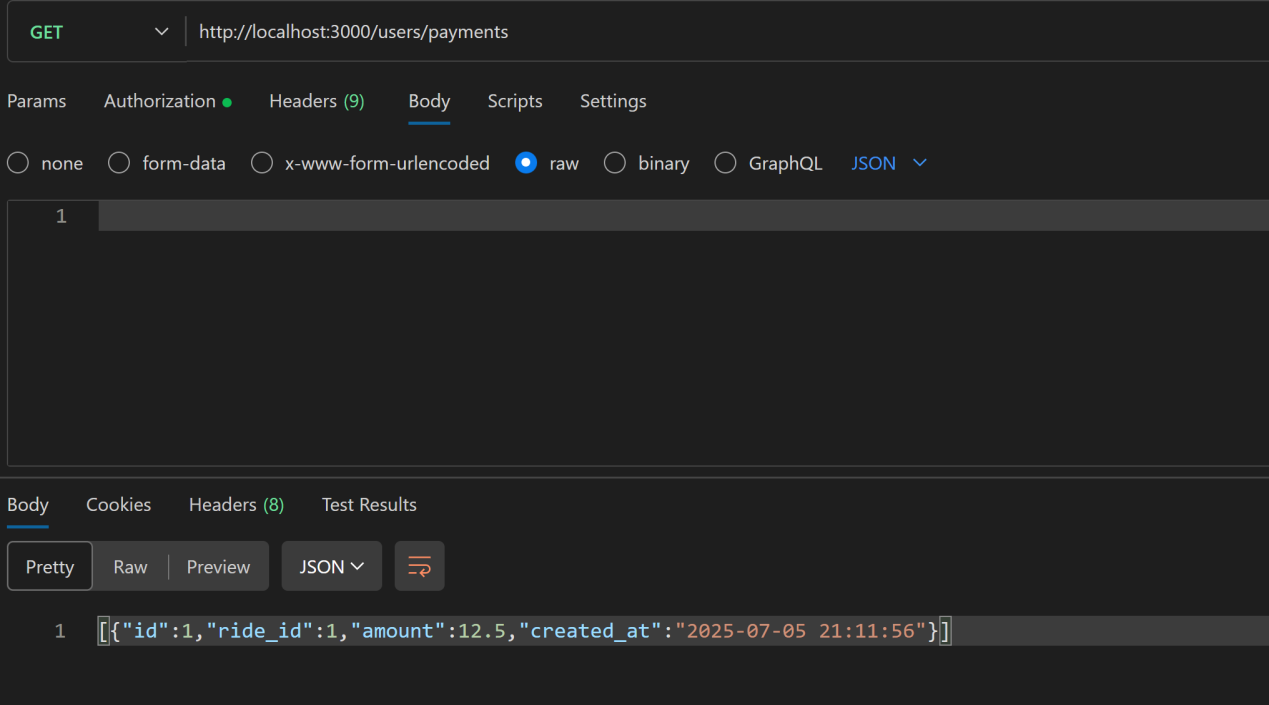
1. **Verify Driver Balance**

* **Endpoint**: GET http://localhost:3000/users/balance
* **Headers**: None
* **Authentication**: Basic Auth (Username: driver@example.com, Password: driver123)
* **Purpose**: Driver can check it’s balance
* **Response**:



1. **Track Total Payment:**

* **Endpoint**: GET http://localhost:3000/users/payments
* **Headers**: None
* **Authentication**: Basic Auth (Username: driver@example.com, Password: driver123)
* **Purpose**: Driver can check it’s balance history
* **Response**:



### ****Real-Time Features****

For real-time updates (e.g., ride request notifications), WebSocket integration is recommended.

### ****Error Handling****

Robust error handling has been implemented across all endpoints to ensure stability and meaningful feedback during failures.