Service Layers

Overview

Using Node.js with Express to build the backend of the WanderLogixs web application is a pragmatic choice. Node.js, known for its speed and non-blocking I/O, paired with the minimalist and flexible Express.js framework, offers a solid foundation for creating a performant and user-friendly application.

Here's an overview of how the backend of the WanderLogixs web application will operate, split into service layers:

- **Routing and Request Handling Layer:**
  At the core of the backend is the routing and request handling layer, powered by Express.js. This layer defines the API endpoints (routes) and their associated controllers. Each route corresponds to a specific service, such as user registration, trip creation, and expense recording. When a request (e.g., a POST request to create a new trip) is made to a specific endpoint, Express routes it to the corresponding controller for further processing.

- **Controller Layer:**
  The controller layer is responsible for handling the business logic associated with each API endpoint. For instance, when a user wants to create a new trip, the trip controller processes the request. It communicates with the database, ensuring that the data is stored correctly, and then sends an appropriate response back to the client. Additionally, controllers may perform authentication and validation tasks.
- **Data Access Layer:**
  The data access layer is where the backend interacts with the PostgreSQL database. This layer includes functions and queries for connecting to the database, fetching data, and updating

records. For example, when a user creates a new trip, the trip controller delegates tasks related to database operations to this layer. It ensures data consistency and integrity by handling operations like inserting trip details into the database.

- **Authentication and Security Layer:**
  Security is a critical aspect of the backend. User authentication, encryption of sensitive data, and other security measures are implemented in this layer. It ensures that user data is protected and that only authorized users can access certain services. For example, user registration and login services rely on this layer to confirm user identities securely.

Specifications

To design the service layer for the WanderLogixs web application, I'll follow a structured approach, identifying the service endpoints, describing their purpose, and providing example requests and responses. Additionally, I'll create a diagram to illustrate how communication will flow from the user interface pages to these service endpoints.

User Authentication

GET /api/auth

- **Method:** GET
- **Purpose:** Retrieve user information or verify authentication status.
- **Example Request:**

```
GET /api/auth
```

- **Example Response (Success):**

```
{
  "message": "User information retrieved.",
  "user": {
    "userID": "12345",
    "username": "exampleuser",
    "email": "user@example.com"
  }
}
```

- **Error Response (Unauthorized):**

```
{
  "error": "Unauthorized. Please log in."
}
```

POST /api/auth/register

- **Method:** POST
- **Purpose:** Register a new user.
- **Example Request:**

```
POST /api/auth/register
Content-Type: application/json

{
  "username": "exampleuser",
  "email": "user@example.com",
  "password": "securepassword"
}
```

- **Example Response (Success):**

```
{
  "message": "Registration successful. User ID: 12345"
}
```

- **Error Response (User Already Exists):**

```
{
  "error": "User already exists."
}
```

POST /api/auth/login

- **Method:** POST
- **Purpose:** Log in a user.
- **Example Request:**

```
POST /api/auth/login
Content-Type: application/json

{
  "username": "exampleuser",
  "password": "securepassword"
}
```

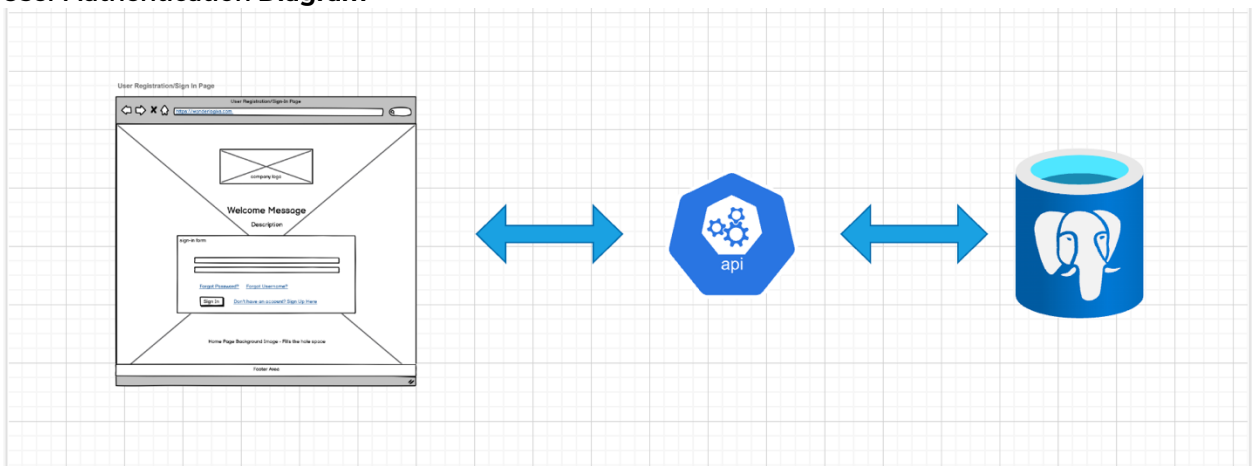
- **Example Response (Success):**

```
{
  "message": "Login successful. User ID: 12345"
}
```

- **Error Response (User Already Exists):**

```
{
  "error": "Invalid username or password."
}
```

User Authentication Diagram



Trips Management

GET /api/trips

- **Method:** GET
- **Purpose:** Retrieve a list of the user's trips.
- **Example Request:**

```
GET /api/trips
```

- **Example Response (Success):**

```
{
  "message": "Trips retrieved successfully.",
  "trips": [
    {
      "tripID": "123",
      "destination": "Paris",
      "startDate": "2023-01-15",
      "endDate": "2023-01-20",
      "purpose": "Vacation"
    },
    // More trip objects...
  ]
}
```

- **Error Response (Unauthorized):**

```
{
    "error": "Unauthorized. Please log in."
  }
```

GET /api/trips/:tripID

- **Method:** GET
- **Purpose:** Retrieve details of a specific trip.
- **Example Request:**

```
GET /api/trips/123
```

- **Example Response (Success):**

```
{
  "message": "Trip details retrieved successfully.",
  "trip": {
    "tripID": "123",
    "destination": "Paris",
    "startDate": "2023-01-15",
    "endDate": "2023-01-20",
    "purpose": "Vacation"
  }
}
```

- **Error Response (Trip Not Found):**

```
{
  "error": "Trip not found."
}
```

POST /api/trips

- **Method:** POST
- **Purpose:** Create a new trip.
- **Example Request:**

```
POST /api/trips
Content-Type: application/json

{
  "destination": "New York",
  "startDate": "2023-02-10",
  "endDate": "2023-02-15",
  "purpose": "Business"
}
```

- **Example Response (Success):**

```
{
    "message": "Trip created successfully. Trip ID: 456"
 }
```

- **Error Response (Validation Error):**

```
 {
   "error": "Invalid date format."
 }
```

PUT /api/trips/:tripID

- **Method:** PUT
- **Purpose:** Update trip details.
- **Example Request:**

```
{
  "destination": "London",
  "startDate": "2023-03-05",
  "endDate": "2023-03-10",
  "purpose": "Vacation"
}
```

- **Example Response (Success):**

```
{
  "message": "Trip details updated successfully."
}
```

- **Error Response (Trip Not Found):**

```
{
  "error": "Trip not found."
}
```

DELETE /api/trips/:tripID

- **Method:** DELETE
- **Purpose:** Delete a trip.
- **Example Request:**

```
DELETE /api/trips/123
```

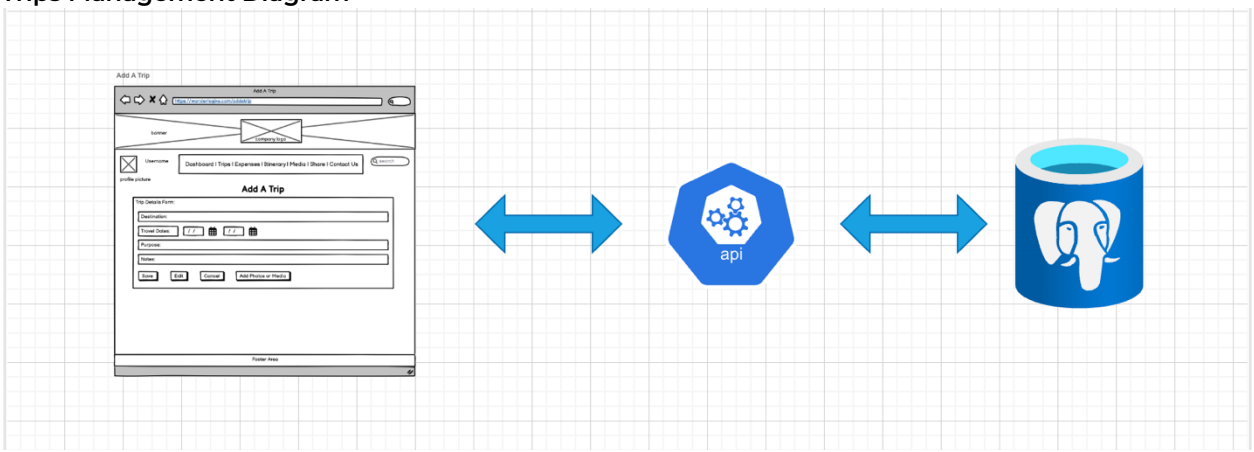
- **Example Response (Success):**

```
{
  "message": "Trip deleted successfully."
}
```

- **Error Response (Trip Not Found):**

```
{
  "error": "Trip not found."
}
```

Trips Management Diagram



Expenses Tracking

GET /api/expenses/:tripID

- **Method:** GET
- **Purpose:** Retrieve expenses for a specific trip.
- **Example Request:**

```
GET /api/expenses/123
```

- **Example Response (Success):**

```
{
  "message": "Expenses retrieved successfully.",
  "expenses": [
    {
      "expenseID": "789",
      "date": "2023-01-16",
      "category": "Dining",
      "amount": 75.50,
      "description": "Dinner at a local restaurant."
    },
    // More expense objects...
  ]
}
```

- **Error Response (Unauthorized):**

```
{
    "error": "Unauthorized. Please log in."
  }
```

POST /api/expenses

- **Method:** POST
- **Purpose:** Create a new expense.
- **Example Request:**

```
POST /api/expenses
Content-Type: application/json

{
  "tripID": "123",
  "date": "2023-01-16",
  "category": "Dining",
  "amount": 75.50,
  "description": "Dinner at a local restaurant."
}
```

- **Example Response (Success):**

```
{
  "message": "Expense added successfully. Expense ID: 789"
}
```

- **Error Response (Validation Error):**

```
{
  "error": "Invalid expense category."
}
```

PUT /api/expenses/:expenseID

- **Method:** PUT
- **Purpose:** Update an expense.
- **Example Request:**

```
PUT /api/expenses/789
Content-Type: application/json

{
  "date": "2023-01-17",
  "category": "Entertainment",
  "amount": 50.00,
  "description": "Movie tickets."
}
```

- **Example Response (Success):**

```
{
  "message": "Expense updated successfully."
}
```

- **Error Response (Expense Not Found):**

```
{
  "error": "Expense not found."
}
```

DELETE /api/expenses/:expenseID

- **Method:** DELETE
- **Purpose:** Delete an expense.
- **Example Request:**

```
DELETE /api/expenses/789
```

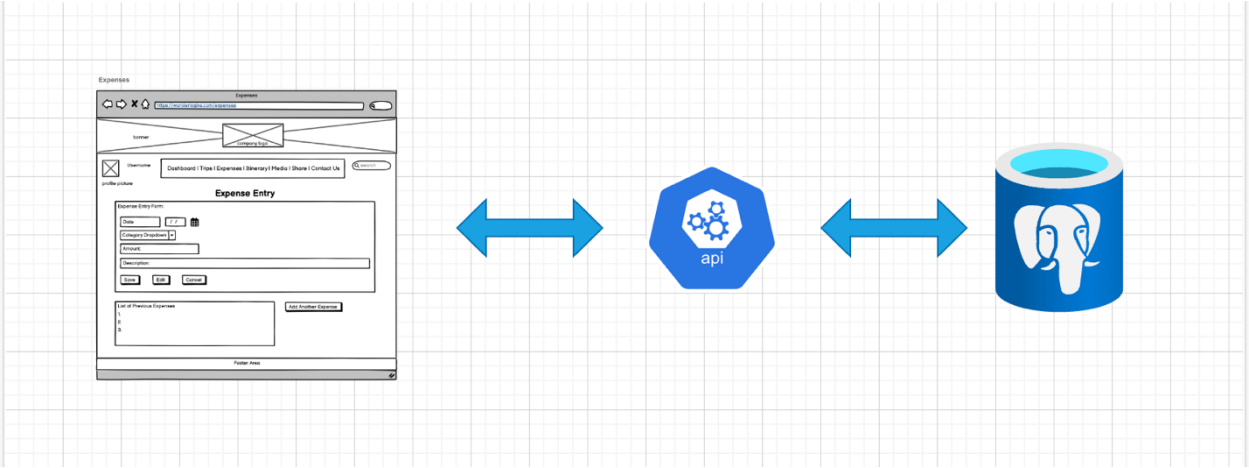
- **Example Response (Success):**

```
{
  "message": "Expense deleted successfully."
}
```

- **Error Response (Expense Not Found):**

```
{
  "error": "Expense not found."
}
```

Expenses Tracking Diagram



Itinerary Management

GET /api/itineraries/:tripID

- **Method:** GET
- **Purpose:** Retrieve day-to-day itineraries for a specific trip.
- **Example Request:**

```
GET /api/itineraries/123
```

- **Example Response (Success):**

```
{
  "message": "Itineraries retrieved successfully.",
  "itineraries": [
    {
      "itineraryID": "456",
      "date": "2023-01-16",
      "activity": "Visit Eiffel Tower",
      "notes": "Don't forget the camera!"
    },
    // More itinerary objects...
  ]
}
```

- **Error Response (Unauthorized):**

```
{
  "error": "Unauthorized. Please log in."
}
```

POST /api/itineraries

- **Method:** POST
- **Purpose:** Add a new itinerary entry.
- **Example Request:**

```
POST /api/itineraries
Content-Type: application/json

{
  "tripID": "123",
  "date": "2023-01-16",
  "activity": "Visit Eiffel Tower",
  "notes": "Don't forget the camera!"
}
```

- **Example Response (Success):**

```
{
  "message": "Itinerary added successfully. Itinerary ID: 456"
}
```

- **Error Response (Validation Error):**

```
{
  "error": "Invalid date format."
}
```

PUT /api/itineraries/:itineraryID

- **Method:** PUT
- **Purpose:** Update an itinerary entry.
- **Example Request:**

```
PUT /api/itineraries/456
Content-Type: application/json

{
  "date": "2023-01-17",
  "activity": "Visit Louvre Museum",
  "notes": "Museum opens at 9 AM."
}
```

- **Example Response (Success):**

```
{
  "message": "Itinerary updated successfully."
}
```

- **Error Response (Itinerary Not Found):**

```
{
  "error": "Itinerary not found."
}
```

DELETE /api/itineraries/:itineraryID

- **Method:** DELETE
- **Purpose:** Delete an itinerary entry.
- **Example Request:**

```
DELETE /api/itineraries/456
```

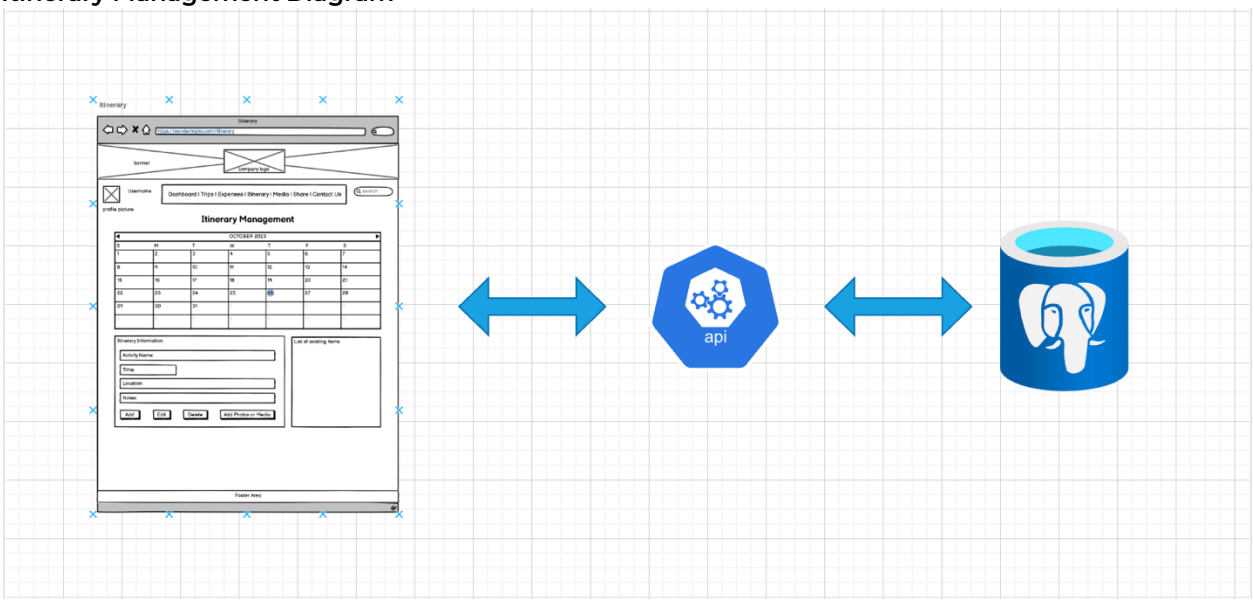
- **Example Response (Success):**

```
{
  "message": "Itinerary deleted successfully."
}
```

- **Error Response (Itinerary Not Found):**

```
{
  "error": "Itinerary not found."
}
```

Itinerary Management Diagram



Media Upload

POST /api/media

- **Method:** POST
- **Purpose:** Upload media content and associate them with a trip or specific day.
- **Example Request:**

```
POST /api/media
Content-Type: application/json

{
  "tripID": "123",
  "date": "2023-01-16",
  "mediaType": "photo",
  "fileURL": "https://example.com/photo.jpg",
  "description": "View from the Eiffel Tower."
}
```

- **Example Response (Success):**

```
{
  "message": "Media uploaded successfully. Media ID: 789"
}
```

- **Error Response (Validation Error):**

```
{
  "error": "Invalid media type."
}
```

PUT /api/media/:mediaID

- **Method:** PUT
- **Purpose:** Update media details or associations.
- **Example Request:**

```
PUT /api/media/789
Content-Type: application/json

{
  "date": "2023-01-17",
  "description": "At the Louvre Museum."
}
```

- **Example Response (Success):**

```
{
  "message": "Media details updated successfully."
}
```

- **Error Response (Validation Error):**

```
{
  "error": "Media not found."
}
```

DELETE /api/media/:mediaID

- **Method:** DELETE
- **Purpose:** Delete media content.
- **Example Request:**

```
DELETE /api/media/789
```

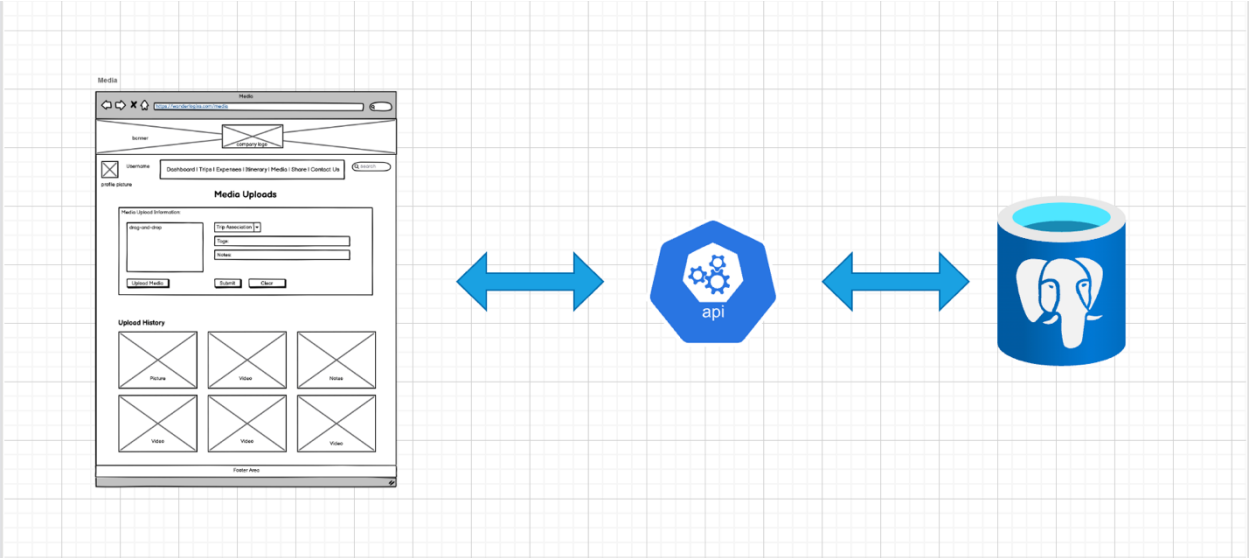
- **Example Response (Success):**

```
{
  "message": "Media deleted successfully."
}
```

- **Error Response (Media Not Found):**

```
{
  "error": "Media not found."
}
```

Media Upload Diagram



Share/Memory

GET /api/memories/:tripID

- **Method:** GET
- **Purpose:** Retrieve shared memories for a specific trip.
- **Example Request:**

```
GET /api/memories/123
```

- **Example Response (Success):**

```
{
  "message": "Memories retrieved successfully.",
  "memories": [
    {
      "memoryID": "101",
      "title": "Eiffel Tower Visit",
      "text": "Unforgettable experience at the Eiffel Tower."
    },
    // More memory objects...
  ]
}
```

- **Error Response (Unauthorized):**

```
{
  "error": "Unauthorized. Please log in."
}
```

POST /api/memories

- **Method:** POST
- **Purpose:** Add a new memory entry.
- **Example Request:**

```
POST /api/memories
Content-Type: application/json

{
  "tripID": "123",
  "title": "Louvre Museum Visit",
  "text": "Spent the day exploring art at the Louvre."
}
```

- **Example Response (Success):**

```
{
  "message": "Memory added successfully. Memory ID: 202"
}
```

- **Error Response (Validation Error):**

```
{
  "error": "Memory title is required."
}
```

PUT /api/memories/:memoryID

- **Method:** PUT
- **Purpose:** Update a memory entry.
- **Example Request:**

```
PUT /api/memories/202
Content-Type: application/json

{
  "title": "Louvre Museum Visit",
  "text": "Spent the day exploring art at the Louvre. Must-visit!"
}
```

- **Example Response (Success):**

```
{
  "message": "Memory updated successfully."
}
```

- **Error Response (Memory Not Found):**

```
{
  "error": "Memory not found."
}
```

DELETE /api/memories/:memoryID

- **Method:** DELETE
- **Purpose:** Delete a memory entry.
- **Example Request:**

```
DELETE /api/memories/202
```

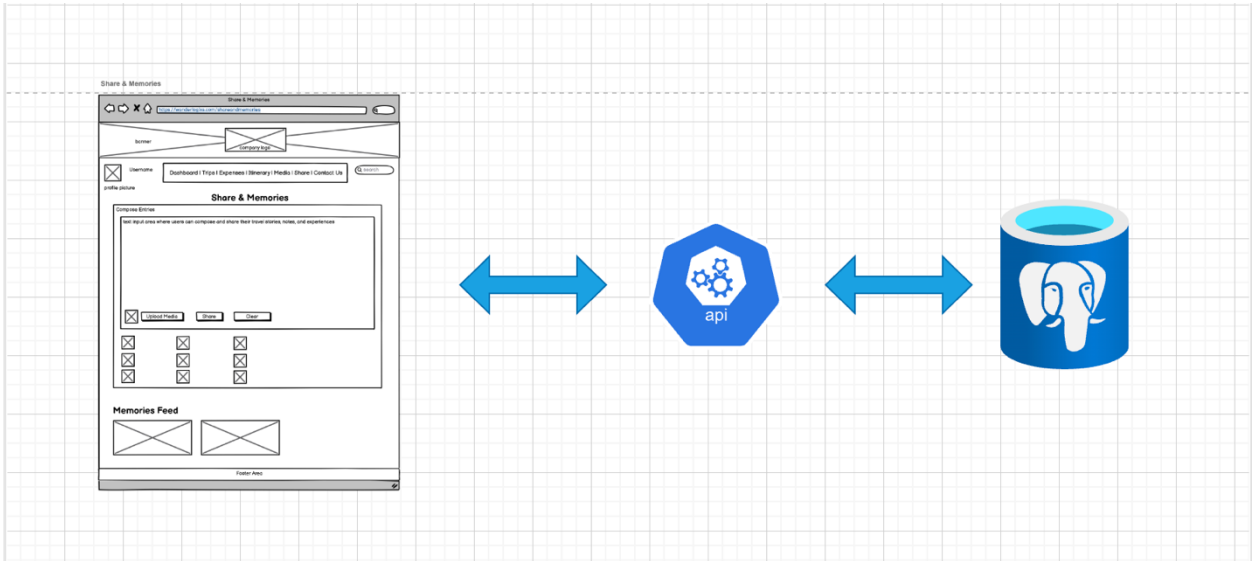
- **Example Response (Success):**

```
{
  "message": "Memory deleted successfully."
}
```

- **Error Response (Memory Not Found):**

```
{
  "error": "Memory not found."
}
```

Share/Memory Diagram



Concluding Information

RESTful Principles: Yes, the application follows RESTful principles where appropriate. Each endpoint is designed to perform a specific action related to the resources it manages, such as retrieving trip details, adding expenses, updating memories, and more. The design adheres to RESTful principles in structuring the API.

HTTP Verbs: HTTP verbs are used appropriately and consistently. For example, GET is used for retrieving data, POST for creating new records, PUT for updating existing records, and DELETE for removing records. This consistency makes it easy for developers to understand and work with the API.

Logical Paths: The paths are designed to logically correspond to their purpose and effectively communicate that purpose. For example, the path `/api/trips` is used to retrieve trip data, which is intuitive and aligns with user expectations. Similarly, paths like `/api/expenses` and `/api/memories` clearly indicate their intended functionality.

Completeness: The design document, in combination with the mockups and ERD, provides a comprehensive blueprint for the application. It outlines all the necessary service endpoints required for the MVP and details how they will be used by the user interface pages.