resolver.js

```
import jwt from 'jsonwebtoken';
import { getData, getUsers, getdataMap, addData } from "./database.js";

// Load data and users from the database
let data = getData();
let users = getUsers();

const resolvers = {
  Query: {
    // Fetch all data only for authenticated users
    getAllData: (parent, args, context) => {
      if (!context.user) {
        throw new Error('Not authenticated'); // Ensure authentication
      }
      return data; // Return all data
    },

    // Fetch a specific record by ID, restricted to authenticated users
    getDatabyId: (parent, args, context) => {
      if (!context.user) {
        throw new Error('Not authenticated'); // Ensure authentication
      }
      return data.find(p => p.id === args.id); // Find and return the data by ID
    },

    // Fetch user-specific data based on their username
    getUserData: (parent, args) => {
      const dataMap = getdataMap(); // Map of users to their data IDs
      const userId = dataMap[args.username]; // Get data IDs for the user
      if (!userId) return []; // Return empty array if no data IDs are found
```

```
    return data.filter(person => userId.includes(person.id)); // Return user's specific data
  },

  // Fetch all users (This may not be safe in production without restrictions)
  getUsers: () => users,
},

// Additional resolver for User type (if implemented in schema)
User: {
  // Fetch data owned by a specific user
  userOwnData: (parent) => {
    const dataMap = getdataMap(); // Map of users to their data IDs
    const userId = dataMap[parent.username]; // Get data IDs for the user
    if (!userId) return []; // Return empty array if no data IDs are found
    return data.filter(person => userId.includes(parent.id)); // Return user's specific data
  }
},

Mutation: {
  // Add new data entry to the database
  addData: (parent, args, context) => {
    if (!context.user) {
      throw new Error('Not authenticated'); // Ensure authentication
    }

    // Check if data with the same ID already exists
    if (data.find(b => b.id === args.id)) {
      throw new Error('Record already exists'); // Prevent duplicate records
    } else {
      const newData = { ...args }; // Create new data object
      addData(newData); // Persist new data to the database
```

```
        data = data.concat(newData); // Update in-memory data
        return newData; // Return the added data
      }
    },

    // Authenticate a user and provide a JWT token
    login: (parent, { username, password }) => {
      // Verify user credentials
      const user = users.find(user => user.username === username && user.password ===
password);
      if (!user) throw new Error('Invalid credentials'); // Invalid credentials

      // Generate a JWT token
      const token = jwt.sign({ username: username }, 'my_secret_key', { expiresIn: '1d' });
      const bearer_token = 'Bearer ' + token;

      // Save token to the user object (non-persistent, temporary storage)
      user.token = token;

      return { "token": bearer_token, username }; // Return token and username
    }
  }
};

export default resolvers;
```

database.j

This code manages a simple JSON-based database using fs for file I/O, allowing persistent storage of users, data, and their associations.

```
import fs from 'fs';

const DATABASE_FILE = './database.json'; // Path to the JSON database file

// Initial database content for first-time setup
const initialData = {
  users: [
    { "username": "jk", "password": "sala", 'token': '', "rateLimiting": { "window": 0, "requestCounter": 0 } },
    { "username": "pl", "password": "pass", 'token': '', "rateLimiting": { "window": 0, "requestCounter": 0 } }
  ],
  data: [
    { "id": "1", "Firstname": "Jyri", "Surname": "Kemppainen" },
    { "id": "2", "Firstname": "Petri", "Surname": "Laitinen" },
    { "id": "3", "Firstname": "Heikki", "Surname": "Helppo" }
  ],
  dataMap: {
    jk: ["1", "3"], // "jk" owns data with IDs 1 and 3
    pl: ["2"]      // "pl" owns data with ID 2
  }
};

// Load data from the database file or initialize if missing
const loadDatabase = () => {
  if (fs.existsSync(DATABASE_FILE)) { // Check if the database file exists
    try {
```

```
        return JSON.parse(fs.readFileSync(DATABASE_FILE, 'utf8')); // Parse and return file content
      } catch (error) {
        console.error('Error reading database file:', error);
      }
    }
    saveDatabase(initialData); // If file is missing or invalid, initialize with default data
    return initialData;
};

// Save in-memory database to the file
const saveDatabase = (data) => {
    fs.writeFileSync(DATABASE_FILE, JSON.stringify(data, null, 2), 'utf8'); // Write JSON data with 2-space indentation
};

// Load the database into memory on startup
let db = loadDatabase();

// Retrieve all users
const getUsers = () => {
    return db.users; // Return users from the in-memory database
};

// Retrieve all data records
const getData = () => {
    return db.data; // Return data from the in-memory database
};

// Retrieve the user-to-data mapping
const getdataMap = () => {
    return db.dataMap; // Return dataMap from the in-memory database
```

```
};

// Add a new data record and save it persistently
const addData = (newData) => {
  db.data.push(newData); // Add new record to in-memory data array
  saveDatabase(db); // Persist the updated database to the file
};

// Update user information (e.g., token or rate limiting)
const updateUser = (username, userUpdates) => {
  const userIndex = db.users.findIndex(user => user.username === username); // Find user by username
  if (userIndex >= 0) {
    db.users[userIndex] = { ...db.users[userIndex], ...userUpdates }; // Merge updates with existing user
    saveDatabase(db); // Persist the updated database to the file
  }
};

export {
  getUsers,    // Export function to retrieve users
  getData,     // Export function to retrieve data
  getdataMap,  // Export function to retrieve the user-to-data mapping
  addData,     // Export function to add new data
  updateUser   // Export function to update user details
};
```

Code Functionality

1. **Initial Database Content**:
   - The `initialData` object contains default data for users, their associated records (`data`), and a mapping (`dataMap`) of which users own which records.
2. **Database File Management**:

    - `DATABASE_FILE` specifies the JSON file used for storing the database (`database.json`).
    - The system attempts to load the database from the file at startup. If the file doesn't exist, it initializes it with `initialData`.
3. **Core Functions**:
    - **`loadDatabase`**: Reads data from `database.json`. If the file is missing or invalid, it initializes with `initialData`.
    - **`saveDatabase`**: Writes the current in-memory database to `database.json`.
    - **`getUsers`**: Returns the list of users from the in-memory database.
    - **`getData`**: Returns the list of data records.
    - **`getdataMap`**: Returns the mapping of users to their owned data.
    - **`addData`**: Adds a new record to the database and saves it persistently.
    - **`updateUser`**: Updates the details of a specific user (e.g., token or rate-limiting info) and persists the change.
4. **Persistent Storage**:
    - Every modification (e.g., `addData`, `updateUser`) updates the in-memory database (`db`) and saves the changes to `database.json`.