Mini Project Documentation - User Management System with Encryption

# 1. Project Overview

This project is focused on a simple user management system integrated with encryption and decryption mechanisms for handling sensitive data such as emails. The application uses Node.js with Sequelize ORM to manage data storage in MySQL.

# 2. Setup and Configuration

Components Involved:  
- Node.js  
- Sequelize (ORM)  
- MySQL (Database)  
- crypto module (for encryption and decryption)  
- Environment variables (.env)

Process:  
1. Initialize Project:  
- Set up a Node.js project using `npm init`.  
- Install necessary dependencies:  
`npm install sequelize mysql2 crypto dotenv`  
  
2. Create a `.env` File for Storing Keys:  
- Define the encryption key and IV:  
`encryption\_key=32-byte-encryption-key`  
`iv\_key=16-byte-iv-key`  
  
3. Database Configuration:  
- Create a `database.js` to establish a connection using Sequelize.  
- Ensure MySQL is installed and a database is set up.

# 3. Models Setup

Components Involved:  
- Sequelize Models: `Customer` and `Order`

Process:  
1. Customer Model:  
- Fields: `id`, `name`, `email`  
- Hooks: `beforeCreate`, `beforeUpdate` for encrypting email.  
- Hook: `afterFind` for decrypting email on retrieval.  
  
2. Order Model:  
- Fields: `id`, `total`, `customerId`  
- Relation: `Customer` has many `Order`.

Encountered Issue:  
- Error in Email Decryption: When fetching customers, some customers' emails weren't decrypting properly.  
- Solution: Checked the encryption and decryption logic, ensured that proper keys and IV values were being used. Applied fixes to handle edge cases where decryption could fail.

# 4. Encryption and Decryption (Encryption Module - encr-decr.js)

Components Involved:  
- `crypto.createCipheriv` for encryption  
- `crypto.createDecipheriv` for decryption  
- Environment Variables for key and IV

Process:  
1. Encrypt Function:  
- Uses AES-256-CBC algorithm.  
- Takes `text`, returns encrypted hex string.  
  
2. Decrypt Function:  
- Reverses the encryption process using the same algorithm and keys.  
- Returns decrypted text.

Encountered Issue:  
- Invalid Encoding in Decryption: Received an error when trying to decrypt emails that had different encrypted lengths.  
- Solution: Ensure the `encryption\_key` and `iv\_key` are of correct size (32 bytes and 16 bytes respectively).

# 5. CRUD Operations (Performed in connection.js)

Components Involved:  
- `sequelize.sync()` to sync models with the database  
- Readline Interface for interactive command-line prompts.

Process:  
1. Select Operations:  
- Option 1: Select by Customer ID, with associated Orders.  
- Option 2: Select All Customers, along with their Orders.  
- Encountered Issue: There was a decryption issue when retrieving the `email` from the database.  
- Solution: Fixed by ensuring the `afterFind` hook is correctly decrypting the email.  
  
2. Insert Operation:  
- User provides customer name and email, which are inserted into the `Customer` model.  
- A corresponding order is created for that customer.  
  
3. Update Operation:  
- Customer details can be updated (name, email, or order total).  
- Encountered Issue: Unable to update the email field properly due to encryption.  
- Solution: Added hooks to handle encryption before saving the data.  
  
4. Delete Operation:  
- A customer and their associated orders are deleted.  
- Encountered Issue: None encountered with delete logic.

# 6. Enhancements and Finalization

Components Involved:  
- Asynchronous operations with Promises.  
- Sequelize hooks for pre and post data handling.

Process:  
1. Add Error Handling in CRUD Operations:  
- Ensure that each operation is wrapped in a try-catch block.  
- Provide clear error messages in case of failures during database interactions.  
  
2. Testing the Application:  
- Perform CRUD operations (Create, Read, Update, Delete) through the CLI.  
- Ensure encryption and decryption are working smoothly for sensitive fields like email.  
  
3. UI Enhancements:  
- Consider making the CLI prompts more user-friendly and interactive.  
- Format the output for better readability.

Encountered Issue:  
- Formatting of Output: The command-line output wasn’t readable due to raw object formatting.  
- Solution: Use `JSON.stringify()` for formatting complex objects.

# 7. Next Steps

1. User Authentication:  
- Implement user email validation using hashed passwords (bcrypt or argon2).  
  
2. Expand the Application:  
- Add more fields to `Customer` and `Order` models.  
- Add more complex relationships (many-to-many) if needed.  
  
3. Enhance Security:  
- Ensure that sensitive data like passwords are also encrypted.  
  
4. Optimization:  
- Optimize the encryption and decryption process to handle large datasets efficiently.

# 8. Key Concepts to Understand for Better Clarity

1. Asynchronous Programming:  
- Learn about Promises, async/await, and how to handle asynchronous code effectively.  
  
2. Sequelize ORM:  
- Understand how to define models, establish relationships, and perform CRUD operations.  
  
3. Encryption and Decryption:  
- Study the principles of symmetric encryption (AES) and how to use it securely with Node.js.  
  
4. Error Handling in Node.js:  
- Learn how to catch errors in asynchronous code, especially database interactions and file I/O.  
  
5. Environment Variables:  
- Understand how to securely manage secrets (encryption keys, database credentials) with `.env` files.