ASLSteamHub – A Full Overview

1. Overview

ASLSteamHub is a Node.js + Express application that uses **Sequelize** (with a MariaDB/MySQL dialect) for data persistence. Users can register, sign in, and create or manage lists of words (ASL signs). The application has an administrative layer to manage users and their lists.

Key technologies and libraries include:

- **Express** for HTTP server and routing
- **EJS** templates for server-side rendering
- **Sequelize** ORM for database interaction
- **CSRF**-protection via csrf-csrf
- **JWT** for token-based authentication (in some modules)
- **bcryptjs** for password hashing

2. Application Entry Point

app.js

1. **Environment & Middleware**
    - Loads environment variables from .env using dotenv.
    - Configures **CSRF** protection with the doubleCsrf library.
    - Sets up cookieParser and express.json() / express.urlencoded() for parsing requests.
2. **Database Connection**
    - Imports and initializes sequelize from ./util/database.js.
    - Syncs models with the database (sequelize.sync()).
3. **Routes**
    - Mounts core routes:
        - "/" → pages.js
        - "/auth" → auth.js
        - "/admin" → admin.js
        - "/library" → library.js
        - "/user" → user.js
4. **Server Listening**
    - Listens on port 8000 and logs a success message upon database connection.

3. Data Models & Database

3.1 util/database.js

- Sets up a Sequelize instance configured via environment variables.

3.2 models/user.js

- Defines a User model with fields such as id, email, password, etc.

3.3 models/list.js

- Represents a "List" entity, used to group words (ASL signs) under a certain collection.
- Likely contains fields for id, title (list name), and references a UserId foreign key if the list belongs to a user.

3.4 models/word.js

- Defines an ASL word/sign.
- Key fields might be id, wordName, imageURL, etc.

3.5 models/words-list.js

- A **join table** (through Sequelize) that associates many Words to many Lists.
- References wordId and listId.

3.6 models/associations.js

- Central file to define relationships among User, List, Word, etc.

4. Controllers

4.1 controllers/auth.js

Handles user authentication tasks such as registration, login, and password management.
Typical methods:

- **register**: Creates a new User in the database (password is hashed via bcryptjs).

- **login**: Finds the User by email, compares hashed passwords, and manages session tokens or cookies.
- **logout**: Possibly clears cookies or session info.

4.2 controllers/library.js

Manages the logic for word lists and library functionalities:

- **createList**: Creates a new List associated with the current user.
- **addWordToList**: Adds a Word to a particular list.
- **removeWordFromList**: Removes a Word from a list.
- **fetchLists** or **fetchSingleList**: Retrieves user-specific lists.

4.3 controllers/userController.js

Handles user-related actions outside of auth (like profile updates or admin tasks):

- **getAllUsers**: For admins to list all users in the system.
- **editUser**: Admin or user can update user info (like name, role, password).
- **deleteUser**: Remove a User from the system (may also remove associated lists, etc.).

5. Routes

1. **routes/pages.js**
    - Typically renders the main pages of the site (home, about, contact, etc.).
    - Example endpoints:
        - GET / → Renders index.ejs
        - GET /faq → Renders faq.ejs
2. **routes/auth.js**
    - Houses the authentication endpoints:
        - POST /register → Calls authController.register
        - POST /login → Calls authController.login
        - GET /logout → Calls authController.logout
    - Integrates CSRF tokens in forms for security.
3. **routes/admin.js**
    - Contains routes restricted to administrator-level access:
        - GET /dashboard → Admin dashboard page
        - POST /user/:id/edit → Edit user info
        - DELETE /user/:id → Delete user

4. **routes/library.js**
    - Manages user library endpoints:
        - POST /create-list → Create a new list
        - POST /add-word → Add word to a list
        - GET /my-lists → Retrieve current user's lists
5. **routes/user.js**
    - Additional user routes (profile or settings):
        - GET /profile → Renders user profile page
        - POST /profile → Updates user profile

6. Views

All templates are **EJS** files located in the views/ directory. Some notable views:

- **index.ejs**: Homepage layout. Often includes navbars, footers, or partials (e.g. views/partials/footer.ejs).
- **signup.ejs / signin.ejs**: Registration and login forms.
- **admin-dashboard.ejs**: Display administrative information such as lists of users, word lists, and site stats.
- **user-dashboard.ejs**: The user's personalized space, showing their lists, words, and possible actions.
- **list.ejs / add-list.ejs / edit-list.ejs**: Templates for viewing and managing word lists.
- **word-list.ejs**: Showcases the collection of words in a particular list.

The partials folder contains reusable snippets like headers, footers, or nav bars.

7. Security & Middleware

1. **CSRF Protection**
    - Implemented via doubleCsrf. The app.js sets up app.use(doubleCsrfProtection) to guard against Cross-Site Request Forgery.
    - Tokens can be retrieved from GET /csrf-token and are embedded in form fields.
2. **Auth Checks**
    - Some routes require the user to be logged in (or have admin privileges). This can be enforced through a custom middleware (e.g. verifying session or JWT tokens).

3. **Password Hashing**
    - Uses bcryptjs to hash passwords upon registration or update.
4. **Cookie Parsing**
    - cookie-parser helps with reading or setting cookies, including session tokens.

8. Notable Features & Flows

8.1. Registration and Login

1. **Registration**
    - POST /auth/register
    - Takes email, password, and username.
    - Hashes the password and stores the user record in the database.
2. **Login**
    - POST /auth/login
    - Verifies the user credentials using bcryptjs.
    - If successful, sets a session cookie or sends back some token for subsequent requests.

8.2. Creating and Managing Lists

1. **Create a List**
    - POST /library/create-list
    - Persists a new List record, linking to the currently logged-in user.
2. **Add a Word**
    - POST /library/add-word
    - Associates a new or existing Word with a particular List in the words-list join table.

8.3. Admin Panel

- **Admin Dashboard**: /admin/dashboard
    - Shows site statistics, user lists, etc.
- **Edit/Delete Users**: Allows the admin to update user roles or remove them entirely.

9. Local Setup & Run

Please refer to “**How to set up the dev environment.docx**” to have your local machine set up and ready to run.

10. Gitflow Workflow Guide

Please refer to “**Gitflow Workflow Guide.docx**” for a revision of your knowledge on working with github.

11. Test Environment Setup

Please refer to “**How to set up the test environment.docx**” for setting up the testing environment.