MAJORPROJECT Notes

Overview

MAJORPROJECT is a Node.js web application built with Express.js, MongoDB (via Mongoose), and EJS templating. It implements user authentication, listing and review management, and uses various middleware and utilities for validation, session management, and error handling.

Packages Used and Their Purpose

- **express**: Web framework for routing and middleware.
- **mongoose**: ODM for MongoDB, defines schemas and models.
- **passport**: Authentication middleware.
- **passport-local**: Local username/password strategy for Passport.
- **passport-local-mongoose**: Mongoose plugin simplifying Passport local authentication integration.
- **joi**: Data validation library used for validating request bodies.
- **connect-flash**: Flash messages middleware for Express.
- **ejs**: Embedded JavaScript templating engine.
- **ejs-mate**: Layout support for EJS templates.
- **express-session**: Session middleware for Express.
- **method-override**: Middleware to support HTTP verbs like PUT and DELETE via query parameter.

---

app.js

- Sets up Express app with EJS as view engine using ejs-mate for layouts.
- Connects to MongoDB using Mongoose.

- Uses middleware: `express.urlencoded` for parsing form data, `method-override` for HTTP verbs, and serves static files from `public` directory.
- Configures session with `express-session` and flash messages with `connect-flash`.
- Initializes Passport for authentication, using `passport-local` strategy with `User` model.
- Mounts routers for listings, reviews, and user routes.
- Includes error handling middleware rendering an error page.

---

middleware.js

- Exports `isLoggedIn` middleware that checks if a user is authenticated using Passport's `req.isAuthenticated()`.
- If not authenticated, flashes an error and redirects to login page.
- Used to protect routes requiring login.

---

routes/user.js

- Defines routes for user signup and login.
- Uses `passport` for authentication and `wrapAsync` utility to handle async errors.
- Signup route registers new users with `User.register` (passport-local-mongoose).
- Login route authenticates users with Passport local strategy.
- Flash messages used to notify success or errors.

---

schema.js

- Defines Joi validation schemas:
    - `listingSchema`: Validates listing data (title, description, location, country, price, image).

    - `reviewSchema`: Validates review data (rating 1-5, comment).

---

models/user.js

- Mongoose schema for User with email field.
- Uses `passport-local-mongoose` plugin to add username, password hashing, and authentication methods.

---

models/listing.js

- Mongoose schema for Listing with fields: title, description, image (with default), price, location, country.
- Contains array of references to Review documents.
- Post middleware on `findOneAndDelete` to delete associated reviews when a listing is deleted.

---

utils/ExpressError.js

- Custom error class extending JavaScript `Error`.
- Includes `statusCode` and `message` properties for HTTP error handling.

---

utils/wrapAsync.js

- Utility function to wrap async route handlers.
- Catches errors and passes them to Express error handling middleware to avoid repetitive try-catch blocks.

Authentication Flow and Session Management

- Passport.js configured with local strategy using `passport-local-mongoose` User model.
- Sessions managed with `express-session`.
- Flash messages used to communicate login/signup success or failure.
- `isLoggedIn` middleware protects routes requiring authentication.

Validation with Joi

- Incoming listing and review data validated against Joi schemas before processing.
- Ensures data integrity and proper error handling.

Database Models and Relationships

- User model stores user credentials and email.
- Listing model stores listing details and references multiple reviews.
- Review model (not detailed here) likely stores individual review data.
- Cascade delete implemented to remove reviews when a listing is deleted.

Error Handling Strategy

- Custom `ExpressError` class used to create errors with status codes.
- Async route handlers wrapped with `wrapAsync` to forward errors.
- Centralized error handling middleware renders error page with error details.

End of Notes