Project 1 — Beginner Progress Report

Created by: Anshika Bhadauriya
Date: 2026-02-18

Summary
I built a minimal full-stack feature that demonstrates user registration, login, and a dashboard. The backend uses Node.js with Express (described below) and server-side rendering with EJS. The frontend contains simple HTML forms for user input. Data is persisted in a lightweight database and shown on the dashboard after login.

1) Create an HTML structure with forms for user input
- Build clear, accessible forms using standard HTML elements: form, label, input, and button.
- Include client-side validation where helpful (required fields, simple length checks) to improve user experience.

Example (login form):
Email

Password

Sign In

2) Set up a simple Node.js server using Express
- Initialize a Node.js project (npm init -y) and install Express (npm install express).
- Create an index.js (or server.js) that imports Express, sets up middleware for parsing application/x-www-form-urlencoded and JSON, and starts listening on a port.

Example (server bootstrap):
```
const express = require('express');
const app = express();
app.use(express.urlencoded({ extended: false }));
app.use(express.json());
app.listen(3000, () => console.log('Server started on http://localhost:3000'));
```

3) Create server-side endpoints to handle form submissions
- Define POST routes for registration and login. Validate inputs on the server, hash passwords before storing, and provide appropriate responses (redirect, JSON, or rendered page).
- Use a minimal data store (SQLite, JSON file, or any DB) for persistence.

Example endpoints (Express):
```
app.post('/register', async (req, res) => {
const { name, email, password } = req.body;
// validate, hash password, store user
res.redirect('/login');
});

app.post('/login', async (req, res) => {
const { email, password } = req.body;
// find user, compare hash, establish session
res.redirect('/dashboard');
});
```

4) Use server-side rendering (EJS) to dynamically generate HTML
- Install EJS (npm install ejs) and configure Express to use it: app.set('view engine', 'ejs').
- Create EJS templates that receive data from the server and render views (e.g., pass the logged-in user's name to the dashboard template).

Example (EJS render):
```
// in route
res.render('dashboard', { user: { name: 'Anshika' }, items: [...] });

// in views/dashboard.ejs
Welcome, <%= user.name %>
```

Project structure (suggested)
- package.json
- index.js

- views/ (EJS templates: login.ejs, register.ejs, dashboard.ejs)
- public/ (static css/js)
- data/ or db/ (database files)

Notes & best practices
- Hash passwords using a battle-tested library such as bcrypt (npm install bcrypt) before storing them.
- Keep server-side validation even if client-side validation exists.
- Organize routes into modules for maintainability.
- Use environment variables for secrets and configuration.

This report explains the small project steps and provides clear code skeletons to implement each requirement using Node.js, Express, and EJS in a beginner-friendly way.