# Web Development Assignment (Frontend + Backend)

If you are applying for only Frontend Developer position, you can do only PHASE 1 and skip PHASE 2.
If you are applying as a full stack developer, complete both phases.

### Overview

This project consists of two phases and aims to evaluate the candidate's ability to build a simple web application. The goal is to assess basic frontend skills using React, and backend integration using Node.js with the NestJS framework.

---

## Technologies to Use

- **Frontend**
  - React
  - TypeScript
  - Vite
  - ESLint
- **Backend**
  - Node.js
  - NestJS (TypeScript-based)
  - Express (within NestJS)

---

### PHASE 1: React + Vite Frontend with Sample Data

#### Objective

To evaluate the candidate's ability to build a basic UI using React and TypeScript, and render sample data.

**Requirements**

- Create a new React + TypeScript project using Vite.
- Build a simple "Homepage" component which shows links for Users and Posts.

- Fetch sample data from https://jsonplaceholder.typicode.com/ and show following lists:

    - **User List**: id, name, username, email
    - **Post List**: userId, id, title

- User should be able to do CRUD operations in both list and also show relation between lists through userId field.
- Apply basic styling for readability and layout; having a user-friendly UI/UX is a plus.
- Code must follow ESLint rules and should not contain linting errors.

**Deliverables**

- A `README.md` file describing how to set up and run the project.
- Code should be pushed to a GitHub repository.

Deploying the application to a free platform such as Netlify or Vercel is a big plus. If you can deploy it, please share the public accessible link too.

---

## PHASE 2: Backend Integration with NestJS

**Objective**

To fetch data from a backend API (built with NestJS) instead of using static data.

**Requirements**

- Create a simple backend server using NestJS and TypeScript.
- Implement the CRUD API endpoints for both users and posts.

We need services to fetch user and post data, create new users, create new posts, update users' and posts' properties, delete users and posts, and add new posts to specific users.

- The initial data should be hardcoded inside the service files (no database required).
- Update the frontend to fetch data from these API endpoints using `fetch` or `axios`.
- Update frontend update services to update data on backend.
- Frontend and backend should run on separate ports.
- Backend code should follow ESLint rules and be free of linting issues.

**Deliverables**

The project should have a folder structure similar to:

```
/project-root
  /frontend
  /backend
```

- Both `frontend` and `backend` folders must contain their own `README.md` files with setup and run instructions.
- Everything should be pushed to a single GitHub repository.

---

## Evaluation Criteria

- Functional correctness of both frontend and backend
- Project structure and code organization
- Proper usage of TypeScript types
- Code quality and ESLint compliance
- Basic but clean UI and good user experience