**Frontend Development Plan — AI-Powered MERN Code Reviewer**

**Goal:** Build a React frontend for the MERN AI code reviewer project, allowing users to register/login, view PR reviews, and see AI feedback.

**Phase 0: Preparation**

**Objective:** Make sure the developer is ready to work with React, Node API, and the project environment.

**Tasks:**

1. Install Node.js (v18+) and npm.
2. Clone the project repository.
3. Navigate to the frontend/ folder.
4. Run npm install to install all dependencies.
5. Create a .env file in frontend/ with:
6. REACT\_APP\_API\_URL=http://localhost:5000/api
7. Start the frontend: npm start → should run at http://localhost:3000.

**Phase 1: Routing and Navigation**

**Objective:** Set up React Router and basic navigation.

**Tasks:**

1. Install React Router DOM: npm install react-router-dom.
2. Create the following pages in src/pages/:
   * Login.js
   * Register.js
   * StartExam.js (dashboard)
   * Exam.js
   * Result.js (PR detail view)
3. Create a ProtectedRoute.js component in src/components/ to protect routes requiring login.
4. Configure routes in App.js with <Routes> and <Route>.
5. Add basic navigation (Link) for login and register.

**Phase 2: Authentication**

**Objective:** Enable user registration and login with JWT.

**Tasks:**

1. Implement Register.js:
   * Form fields: name, email, password.
   * POST /api/auth/register using Axios.
2. Implement Login.js:
   * Form fields: email, password.
   * POST /api/auth/login using Axios.
   * Store JWT token in localStorage.
3. Ensure ProtectedRoute checks for JWT token and redirects to login if absent.
4. Test login and registration flow with backend.

**Phase 3: Dashboard — List of PR Reviews**

**Objective:** Display all PR reviews for the logged-in user.

**Tasks:**

1. Fetch data from backend: GET /api/prs.
2. Display list with:
   * PR title
   * Repo name
   * Date of PR
   * Overall AI score
3. Use a table or card-based layout for clarity.
4. Add navigation to detailed PR view (Result.js) on click.

**Phase 4: PR Review Detail Page**

**Objective:** Display AI feedback, inline comments, and suggestions for each PR.

**Tasks:**

1. Fetch PR detail: GET /api/prs/:prNumber/diff.
2. Display:
   * Overall score and category breakdown (lint, bugs, security, performance)
   * Inline comments (file path, line, comment body)
   * Fix suggestions (patch info)
3. Optional: Add visual chart using Chart.js or Recharts for score breakdown.

**Phase 5: UI and Styling**

**Objective:** Make the frontend visually clean and readable.

**Tasks:**

1. Decide on TailwindCSS or Material-UI.
2. Style:
   * Forms (login/register)
   * Dashboard list/cards
   * PR detail page
   * Inline comments and code snippets
3. Add responsive design (basic mobile support).

**Phase 6: API Integration and Testing**

**Objective:** Ensure frontend communicates correctly with backend and AI service.

**Tasks:**

1. Test registration and login.
2. Test dashboard PR list retrieval.
3. Test PR detail page for inline comments and fix suggestions.
4. Handle loading states and error messages.
5. Ensure JWT token is sent in Authorization header for protected routes.

**Phase 7: Deployment Preparation (Optional)**

**Objective:** Prepare frontend for production deployment.

**Tasks:**

1. Update .env to point to production backend URL.
2. Build React app: npm run build.
3. Serve frontend using a static server or integrate with Node backend.
4. Test deployment with live backend and AI service.

**Recommended Order of Work**

1. **Phase 0 → 1**: Environment setup + routing/navigation.
2. **Phase 2**: Authentication (login/register + JWT).
3. **Phase 3**: Dashboard listing PR reviews.
4. **Phase 4**: PR detail view with AI comments.
5. **Phase 5**: Styling and responsive UI.
6. **Phase 6**: API integration and end-to-end testing.
7. **Phase 7**: Deployment (optional, after full testing).

**Tips for the Beginner**

* Always check the backend API URL in .env.
* Use Axios interceptors or default headers to include JWT token.
* Start with static mock data if backend is not ready, then replace with real API calls.
* Test each component individually before integrating.
* Keep components small and reusable.