COLLEGE CODE :9111

COLLEGE NAME :SRM MCET

DEPARTMENT :CSE

STUDENTNMROLLNO: F527DD6EC9EEBE39B9E46539353F079E

DATE :15/09/2025

Completed the project named as

Phase__3 TECHNOLOGY PROJECT

NAME :SINGLE PAGE APPILICATION

SUBMITTED BY,

NAME:SAMIKSHA .R
MOBILENO:9962408168

IBM-FE — Single Page Application

Phase 3 — MVP Implementation (Deadline – Week 8)

Project Setup

Project setup is the foundation of a successful MVP. During this phase the team creates a robust, repeatable development environment that reduces friction for new contributors and prevents common integration issues. Key outcomes for Project Setup include a reliable directory structure, consistent dependency management, and a bootstrap codebase that contains routing, basic components, and global styles.

Objectives

1. Establish a reproducible development environment (node version, package manager).
2. Create initial project scaffolding (components, pages, routing).
3. Integrate linters and formatters to ensure code quality (ESLint, Prettier).
4. Configure secure environment variable management and secrets for local dev and CI.

Environment & Tooling

Select the primary frontend framework (React recommended for SPA), pick a CSS strategy (CSS Modules / Tailwind), and set up a local development server. Document required Node/NPM versions, install key dependencies, and add developer scripts (start, build, test, lint). This minimizes 'works on my machine' problems.

Core Features Implementation

The MVP must deliver the smallest set of features that provide measurable value to users. Implementation focuses on modular, testable components and a clear mapping between user stories and code deliverables. Prioritize features by impact and development effort and keep the UI lightweight and responsive.

Feature Prioritization & Planning

Use a simple prioritization technique (MoSCoW or RICE) to decide which features enter the MVP. Each feature should have acceptance criteria and a definition of done. Break features into small tasks and assign owners.

Example: Patient Registration Flow (Hospital Management SPA)

Goal: Allow front-desk staff to register a new patient quickly and reliably.

Steps:

- Open 'New Patient' form (modal or page).
- Capture essential fields: name, DOB, contact, ID proof type, emergency contact.
- Validate input on client side and show clear inline errors.
- Submit to API and show success state with patient ID.
- Optionally, navigate to appointment scheduling after registration.

Component Design & Reusability

Design small, focused components (FormInput, Modal, DataTable). Favor composition: build complex UIs by combining simple components. Keep styling consistent using design tokens or a utility-first framework. Document component API and usage examples to speed future development.

Data Storage (Local State / Database)

A single-page application requires careful decisions about where and how data is stored and synchronized. Use local state for ephemeral UI state and a centralized store for shared or complex state.

Local State Strategies

- useState/useReducer for local form state and transient UI values.
- Context API for lightweight cross-component state sharing (auth, theme).
- Redux / Zustand for predictable, testable global state in larger apps.

Persistent Storage & API Design

Design clean REST or GraphQL APIs with explicit contracts. For the MVP, choose a managed backend or BaaS (e.g., Firebase, Supabase) if you want to minimize backend work. Otherwise, a simple Node/Express or serverless endpoint can handle CRUD operations. Apply input validation and authentication at the API layer.

Testing Core Features

Testing should run in parallel with development. Automate where possible and target the most valuable tests first. A practical testing pyramid for an SPA includes unit tests at the base, some integration tests in the middle, and a few end-to-end tests at the top.

```
Import React from “react”;

Import { render, screen, fireEvent } from “@testing-library/react”;

Import “@testing-library/jest-dom”;

Import PatientRegistrationForm from “./PatientRegistrationForm”;



Describe(“Patient Registration Form”, () => {

  Test(“renders form fields correctly”, () => {

    Render(<PatientRegistrationForm />);

      // Check if essential fields are present

    Expect(screen.getByLabelText(/Full Name/i)).toBeInTheDocument();
```

```
  Expect(screen.getByLabelText(/Date of Birth/i)).toBeInTheDocument();

  Expect(screen.getByLabelText(/Contact Number/i)).toBeInTheDocument();

  Expect(screen.getByRole(“button”, { name: /Register/I })).toBeInTheDocument();

});



Test(“shows validation errors for empty fields”, () => {

  Render(<PatientRegistrationForm />);



  fireEvent.click(screen.getByRole(“button”, { name: /Register/I }));



  // Validation messages should appear

  Expect(screen.getByText(/Full Name is required/i)).toBeInTheDocument();

  Expect(screen.getByText(/Date of Birth is required/i)).toBeInTheDocument();

});
```

```
Test(“submits form with valid inputs”, () => {
  Const mockSubmit = jest.fn();
  Render(<PatientRegistrationForm onSubmit={mockSubmit} />);

  fireEvent.change(screen.getByLabelText(/Full Name/i), { target: { value: “John Doe” } });
  fireEvent.change(screen.getByLabelText(/Date of Birth/i), { target: { value: “1990-01-01” } });
  fireEvent.change(screen.getByLabelText(/Contact Number/i), { target: { value: “9876543210” } });

  fireEvent.click(screen.getByRole(“button”, { name: /Register/I }));

  expect(mockSubmit).toHaveBeenCalledWith({
    fullName: “John Doe”,
    dob: “1990-01-01”,
    contact: “9876543210”
  });
```

```
  });

});
```

Testing Plan & Tools

- Unit testing: Jest + React Testing Library for components and utilities.
- Integration testing: test network interactions and component composition.
- E2E testing: Cypress or Playwright to simulate real user flows (registration to appointmentbooking).
- CI integration: run tests on each PR and block merges on failing tests.

Quality Gates & Acceptance Criteria

Define clear acceptance criteria per feature. Examples: 'Patient registration completes with valid inputs and returns patient ID', 'Dashboard loads within 1.5s under average load', 'No critical linting errors on build'. Use these criteria to mark the MVP as ready.

Version Control (GitHub)

GitHub is the single source of truth for code. Adopt a simple branching strategy such as 'main' (production), 'develop' (staging), and feature branches. Use descriptive commit messages and enforce code reviews via pull requests to maintain code quality.

Branching & PR Workflow

- Feature branches: feature/-short-description
- PRs: include description, screenshots if UI changes, and testing notes.
- Reviews: require at least one approver and run automated checks before merge.

CI/CD & Deployment

Integrate GitHub Actions (or another CI) to run tests, linting, and builds on each PR. Automate deployments to a staging environment on successful merges to 'develop' and to production on merges to 'main'. Use feature flags for controlled rollouts and include health checks and basic monitoring.

MVP Acceptance Checklist

- Core features implemented and validated against acceptance criteria.
- Automated tests in place and passing in CI.
- Documentation: README, setup, and component usage guides available.
- Deployment pipeline defined and tested (staging + production).
- Version control discipline followed with PRs, reviews, and changelog entries.

Next Steps & Roadmap

After MVP acceptance, plan iterative releases focusing on the highest-impact features deferred from the MVP. Collect user feedback, track usage analytics, and prioritize the backlog accordingly. Maintain a cadence of short sprints and continuous improvement.