Seamate Lite Architecture & Development Plan

Architecture Overview

Clean Architecture

- **Presentation Layer**: Next.js (React-based UI)
- **Application Layer**: .NET Core Web API (business logic, use cases)
- **Domain Layer**: Entity classes, interfaces
- **Infrastructure Layer**: DB access, external services

Deployment Architecture

- **Shore Side**: Hosted on secure cloud/in-premise server (Azure/AWS)
- **Vessel Side**: Access via browser over low bandwidth
- **Data Sync**: All real-time via REST.

Project Setup & Structure

Folder Structure

Frontend (Next.js)

```
/components
/app
    /auth
   /crew
/layouts
/hooks
/context
/services
/utils
/public
```

Backend (.NET Core API)

```
/Controllers
/Services
/Repositories
/Models
/DTOs
/Mappings
/Middleware
/Configurations (for settings like JWT keys, connection strings)
```

State Management

- **Local State**: React useState/useReducer
- **Global State**: Zustand (lightweight)
- **Persistent State**: LocalStorage & Cookies + SSR hydration (only for login/session)

Rendering Strategy

- **Default: SSR (Server-Side Rendering)** to reduce payload.
- **Critical Pages**: SSR (Server-Side Rendering) – Login, SEO-sensitive.
- **Caching**: Incremental Static Regeneration (ISR).

API & Data Handling

API Approach

- **Use REST APIs** via .NET Core Web API
- API returns minimal payloads, supports pagination.

Caching Strategy

- Frontend: SWR
- Backend: Memory cache (IMemoryCache)

Authentication & Security

- **Authentication**: JWT (preferred) with access + refresh token flow
- **RBAC**: Role-based guards at API and UI route level
- **Environment Variables**: Use .env.local, never commit secrets
- **API Protection**: Rate-limiting, IP whitelisting, CORS policies
- **Data Security**: HTTPS, encryption at rest (SQL), secure headers

Development Strategy

Technical Documents

- Architecture Diagrams
- API Docs (Swagger)
- Coding Guidelines
- Release Notes

UI Strategy

- Use **native HTML + TailwindCSS** (best for low bandwidth)
- Evaluate third-party controls only if necessary (lightweight ones)

Frontend

- Next.js with Typescript
- TailwindCSS or simple CSS modules
- Responsive UI (mobile & tablet friendly)

Backend

- .NET Core 8.0 Web API
- RESTful principles
- Swagger docs enabled

Coding Standards

- C# + TypeScript linting rules
- Prettier + ESLint
- Use interfaces & dependency injection

Testing & QA

- **Unit Tests**: Development Team
- **End-to-End**: Selenium with JAVA
- **Static Code Analysis**: SonarQube / GitHub Advanced Security
- **VAPT**: Burp Suite
- **Error Handling**: Use global exception middleware and frontend toast/log (also log to backend/monitoring system)

Database Strategy

- **Database**: SQL Server
- **Approach**: Normalized tables, secure with views/stored procedures
- **Backup Strategy**: Daily backups, geo-redundant if cloud-hosted

Version Control & CI/CD

Version Control

- GitHub / Azure DevOps
- Git flow (feature → develop → release → main)
- Pull Request policies with code review

CI/CD

- GitHub Actions / Azure DevOps Pipelines
- Automatic build, test, deploy
- Slack/Teams notification on failure

Rollback Strategy

- Maintain backup deployments (Blue/Green or Canary releases)
- Infra as Code (ARM/Bicep or Terraform for Azure)

Deployment & Hosting

- **Shore Hosting**: Azure App Service / Linux VM
- **Static Assets**: Azure Blob/CDN
- **SSL**: Use Azure-managed certificates
- **Monitoring**: App Insights + Log Analytics

Performance Optimization

- Lazy load modules and components
- Code-splitting (Next.js automatic)
- Image optimization using next/image
- GZIP compression
- SQL tuning and indexing

POC Scope

Objective:

Build a simple **Login Page** and a **Crew List Page** using new architecture

Features:

- Login with JWT
- Display crew list from backend API
- Role-based restriction
- Use clean UI with loading states