**✅ A Development Checklist**

**1. Core Architecture**

* Backend in **.NET 8 (WebAPI, Clean Architecture, Microservices)**
* Frontend in **React + Vite + Tailwind**
* Database: **MongoDB** (for problems, users, discussions)
* Vector DB: **Qdrant** (for AI-powered search & recommendations)
* Queue: **BullMQ (Redis)** for async tasks (e.g., parsing, notifications)
* Authentication: **JWT + Refresh Tokens**
* RBAC (Role-based access control) → user, admin
* API Gateway (optional for scaling later)

**2. User Features**

* **Auth & Profiles**
  + Register / Login (Email, Social, SSO future)
  + Email verification
  + Profile page with solved problems, streaks
* **Problem Bank**
  + Browse problems by category, pattern, difficulty, company tags
  + Problem statement, examples, constraints, hints
  + Code editor with test cases (frontend integration)
  + Solutions (user’s + editorial)
* **Practice Features**
  + Track progress
  + Streaks (future add-on)
  + AI hints/explanations (AI integration)
* **Search**
  + Normal search (Mongo text index)
  + Semantic search (Qdrant + embeddings)

**3. Admin Dashboard (instead of service)**

* **Users Management**
  + View/search users
  + Ban/unban users
  + Promote/demote roles
* **Problem Management**
  + Approve/reject custom user-uploaded problems
  + Edit/update tags, difficulty
* **Discussion Moderation**
  + Flagged posts view
  + Ban toxic users
* **System Insights (AI-assisted, optional)**
  + Popular problems, hardest problems
  + Trends (“Sliding Window is trending”)

**4. Discussions**

* Add per-problem discussion thread
* Users can comment, reply, upvote
* Admin moderation via dashboard

**5. Future Add-ons**

* Premium subscription (Stripe/Razorpay)
* Daily streaks + rewards
* Discussion forums (beyond per-problem)
* Custom problem uploads → Admin approves → goes live
* AI-powered dashboard insights (auto-difficulty, tagging)

**6. DevOps & Infra**

* **CI/CD**: GitHub Actions (build, test, deploy)
* **Dockerized** backend, frontend, redis, qdrant, mongo
* **Kubernetes (future scaling)**
* **Monitoring**: Prometheus + Grafana (system health)
* **Logging**: ELK stack (ElasticSearch, Logstash, Kibana)
* **Arena UI** for job queues (already in place)

**7. SEO & Performance**

* React **Vite** build for performance
* Meta tags, OpenGraph tags
* Sitemap + robots.txt
* Lazy loading + Code splitting
* Pre-render popular pages (for SEO)

**8. AI Integration**

* Problem search → semantic embeddings (Qdrant)
* AI hints (“think about sliding window”)
* AI explanations for solutions
* AI insights in Admin Dashboard (problem difficulty predictions)

**9. APIs (High-level)**

* **Auth** → /api/auth/register, /api/auth/login, /api/auth/refresh
* **Problems** → /api/problems, /api/problems/:id, /api/problems/upload
* **Search** → /api/search?q=…, /api/search/semantic
* **Discussions** → /api/discussions/:problemId
* **Admin** → /api/admin/users, /api/admin/problems/pending, /api/admin/discussions
* **Jobs** → /api/jobs/:id
* **Extract (OCR)** → /api/extract/pdf, /api/extract/image, /api/extract/text
* **LLM** → /api/llm/ask

**10. UI Pages**

* **Public**: Home, Login, Register
* **User**: Dashboard, Problem bank, Problem detail (editor + discussion)
* **Admin Dashboard**: Users, Problems, Discussions, Insights
* **Future**: Premium subscription page, Forum, AI dashboard

## Structure 1. System Overview

**AlgoClone** is a microservice-based platform for **DSA learning, practice, and tracking**. It uses a hybrid stack with:

* **Backend:** .NET 8 microservices with MongoDB
* **Frontend:** Next.js + React + Ant Design
* **Infra:** Docker + K8s (optional)
* **Auth & Config:** Nexus (Auth), Kagi (Config)
* **Messaging:** Kafka
* **Search:** Solr (can switch to ElasticSearch paid tier)
* **AI Assistant:** Perplexity AI (paid) fallback to OpenAI Free
* **Analytics:** Google Analytics (Free), Firebase Analytics (Paid)

## 2. Architecture & Flow

* **User Flow:**
  1. User signs up / logs in → Nexus Auth (JWT + refresh + SSO).
  2. Config values loaded from Kagi (service discovery & secrets).
  3. Requests → API Gateway → .NET microservices.
  4. MongoDB stores problems, users, progress.
  5. Kafka handles events (e.g., problem solved, leaderboard update).
  6. Solr / Elastic handles problem search.
  7. Perplexity AI provides explanations & hints.
  8. Analytics recorded → Google/Firebase.

## 3. Folder / File Structure

algoclone/

├─ .editorconfig

├─ .gitattributes

├─ .gitignore

├─ README.md

├─ LICENSE

├─ .env.example

│

├─ docs/

│ ├─ architecture.md

│ ├─ api-design.md

│ ├─ db-schema.md

│ └─ setup.md

│

├─ shared/

│ ├─ contracts/

│ │ ├─ openapi/

│ │ │ ├─ auth.v1.yaml

│ │ │ ├─ problems.v1.yaml

│ │ │ ├─ submissions.v1.yaml

│ │ │ ├─ discussions.v1.yaml

│ │ │ └─ admin.v1.yaml

│ │ └─ events/

│ │ ├─ user.registered.v1.json

│ │ ├─ problem.created.v1.json

│ │ └─ submission.created.v1.json

│ └─ schemas/

│ ├─ problem.schema.json

│ ├─ submission.schema.json

│ └─ discussion.schema.json

│

├─ backend/

│ ├─ AlgoClone.sln

│ │

│ ├─ AlgoClone.Api/ # Web API

│ │ ├─ AlgoClone.Api.csproj

│ │ ├─ Program.cs

│ │ ├─ appsettings.json.example

│ │ ├─ appsettings.Development.json.example

│ │ ├─ appsettings.Production.json.example

│ │ ├─ Properties/launchSettings.json.example

│ │ ├─ Controllers/

│ │ │ ├─ HealthController.cs

│ │ │ ├─ AuthController.cs

│ │ │ ├─ UsersController.cs

│ │ │ ├─ ProblemsController.cs

│ │ │ ├─ SubmissionsController.cs

│ │ │ ├─ DiscussionsController.cs

│ │ │ ├─ AdminController.cs

│ │ │ └─ SearchController.cs

│ │ ├─ Filters/ApiExceptionFilter.cs

│ │ ├─ Middlewares/

│ │ │ ├─ ErrorHandlingMiddleware.cs

│ │ │ └─ CorrelationMiddleware.cs

│ │ ├─ Auth/

│ │ │ ├─ JwtExtensions.cs

│ │ │ ├─ RbacPolicies.cs

│ │ │ ├─ GoogleOAuthExtensions.cs

│ │ │ └─ CurrentUserAccessor.cs

│ │ ├─ Configs/

│ │ │ ├─ MongoOptions.cs

│ │ │ ├─ JwtOptions.cs

│ │ │ ├─ KafkaOptions.cs

│ │ │ └─ CorsOptions.cs

│ │ ├─ Dtos/

│ │ │ ├─ Auth/

│ │ │ ├─ Problems/

│ │ │ ├─ Submissions/

│ │ │ ├─ Discussions/

│ │ │ └─ Users/

│ │ ├─ Mapping/ApiMappingProfile.cs

│ │ ├─ Swagger/swagger-extensions.cs

│ │ └─ Extensions/

│ │ ├─ ServiceCollectionExtensions.cs

│ │ └─ ApplicationBuilderExtensions.cs

│ │

│ ├─ AlgoClone.Application/

│ │ ├─ Interfaces/

│ │ ├─ Services/

│ │ ├─ Validators/

│ │ ├─ Models/

│ │ └─ Events/

│ │

│ ├─ AlgoClone.Domain/

│ │ ├─ Entities/

│ │ ├─ Enums/

│ │ └─ ValueObjects/

│ │

│ ├─ AlgoClone.Infrastructure/

│ │ ├─ Persistence/

│ │ ├─ Identity/

│ │ ├─ Security/

│ │ ├─ Kafka/

│ │ ├─ Search/

│ │ └─ MCP/

│ │

│ └─ tests/

│ ├─ AlgoClone.Api.Tests/

│ ├─ AlgoClone.Application.Tests/

│ └─ AlgoClone.Infrastructure.Tests/

│

├─ mcp/

│ └─ AlgoClone.Mcp/

│ ├─ AlgoClone.Mcp.csproj

│ ├─ Program.cs

│ ├─ appsettings.json.example

│ ├─ Clients/PerplexityClient.cs

│ ├─ Endpoints/

│ │ ├─ AskEndpoint.cs

│ │ └─ GenTestcasesEndpoint.cs

│ └─ Models/

│ ├─ AskRequest.cs

│ ├─ AskResponse.cs

│ └─ TestcaseGenerateRequest.cs

│

├─ frontend/

│ └─ web/

│ ├─ package.json

│ ├─ vite.config.ts

│ ├─ tsconfig.json

│ ├─ index.html

│ ├─ .env.example

│ ├─ public/

│ │ ├─ favicon.svg

│ │ └─ robots.txt

│ └─ src/

│ ├─ main.tsx

│ ├─ App.tsx

│ ├─ routes.tsx

│ ├─ styles/globals.css

│ ├─ api/

│ ├─ context/

│ ├─ hooks/

│ ├─ components/

│ └─ pages/

│ ├─ Home.tsx

│ ├─ Auth/

│ ├─ Problems/

│ ├─ Learn/

│ ├─ Payments/

│ ├─ Admin/

│ └─ Profile.tsx

│

├─ infra/

│ ├─ docker/

│ │ ├─ backend.Dockerfile

│ │ ├─ frontend.Dockerfile

│ │ └─ mcp.Dockerfile

│ ├─ docker-compose.dev.yml

│ ├─ docker-compose.local.yml

│ ├─ seed/

│ │ ├─ problems.json

│ │ └─ seed-instructions.md

│ └─ scripts/

│ ├─ dev-up.bat

│ ├─ dev-down.bat

│ ├─ build-all.bat

│ └─ publish-api.ps1

│

├─ k8s/

│ ├─ namespaces.yaml

│ ├─ algoclone-gateway.yaml

│ ├─ algoclone-api-deployment.yaml

│ ├─ algoclone-api-service.yaml

│ ├─ algoclone-mcp-deployment.yaml

│ ├─ algoclone-mcp-service.yaml

│ ├─ algoclone-frontend-deployment.yaml

│ ├─ algoclone-frontend-service.yaml

│ └─ config/

│ ├─ cm-appsettings.yaml

│ └─ secret.env.example

│

└─ .github/

└─ workflows/

├─ ci.yml

├─ cd-frontend.yml

└─ cd-backend.yml

## 4. Design Patterns Used

* **Repository Pattern** → MongoDB DAO layer.
* **Factory Pattern** → Service provider switching (Free ↔ Paid).
* **Strategy Pattern** → AI Assistant service selection (Perplexity vs OpenAI).
* **Observer Pattern** → Kafka event-driven communication.
* **Singleton** → DB Context, Config loader.

## 5. System Design Details

* **Scalability**: Kafka + microservices allow horizontal scaling.
* **Resilience**: Circuit breaker for AI/Analytics fallback.
* **Extensibility**: Free-to-Paid switch is plug-and-play.
* **Security**: JWT + refresh + encrypted password in Nexus.
* **Monitoring**: Kibana dashboards + custom metrics.

## 6. Free → Paid Service Switching Playbook

### 🔑 Config: Kagi → Paid Secrets Manager (AWS Secrets Manager)

**Free (local .env):**

builder.Configuration.AddEnvironmentVariables();

**Paid (AWS):**

builder.Configuration.AddSecretsManager(options =>

{

options.SecretsManagerConfig = new AmazonSecretsManagerConfig

{

RegionEndpoint = RegionEndpoint.USEast1

};

});

### 🔒 Auth: Nexus (Password JWT) → Paid SSO (Google, Apple, etc.)

**Free JWT:**

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true

};

});

**Paid (Google SSO):**

services.AddAuthentication()

.AddGoogle(googleOptions =>

{

googleOptions.ClientId = Configuration["GoogleAuth:ClientId"];

googleOptions.ClientSecret = Configuration["GoogleAuth:ClientSecret"];

})

.AddApple(appleOptions =>

{

appleOptions.ClientId = Configuration["AppleAuth:ClientId"];

appleOptions.KeyId = Configuration["AppleAuth:KeyId"];

});

### 🤖 AI: OpenAI Free → Perplexity AI (Paid)

**Free (OpenAI):**

import OpenAI from "openai";

const client = new OpenAI({ apiKey: process.env.OPENAI\_KEY });

**Paid (Perplexity):**

import axios from "axios";

async function askPerplexity(query) {

const response = await axios.post(

"https://api.perplexity.ai/chat/completions",

{

model: "sonar-medium-online",

messages: [{ role: "user", content: query }],

},

{ headers: { Authorization: `Bearer ${process.env.PERPLEXITY\_KEY}` } }

);

return response.data;

}

### 📊 Analytics: Google Analytics (Free) → Firebase (Paid)

**Free:**

import ReactGA from "react-ga4";

ReactGA.initialize("G-XXXXX");

**Paid:**

import { getAnalytics, logEvent } from "firebase/analytics";

const analytics = getAnalytics();

logEvent(analytics, "problem\_solved", { problemId });

### 🔍 Search: Solr → ElasticSearch Cloud

**Free (Solr):**

var solr = ServiceLocator.Current.GetInstance<ISolrOperations<Problem>>();

solr.Add(problem);

**Paid (Elastic):**

var settings = new ConnectionSettings(new Uri("https://elastic-cloud"))

.DefaultIndex("problems");

var client = new ElasticClient(settings);

client.Index(problem);

## 7. One-Page Structure Diagram (ASCII)

┌──────────┐

│ Frontend│

│ Next.js │

└─────┬────┘

│

┌──────▼──────┐

│ API Gateway │

└──────┬──────┘

┌─────────────┼─────────────┐

│ │ │

┌──▼───┐ ┌───▼───┐ ┌───▼───┐

│ Auth │ │ User │ │Problem│

│Svc │ │Svc │ │Svc │

└──┬───┘ └───┬───┘ └───┬───┘

│ │ │

▼ ▼ ▼

JWT/Nexus MongoDB Solr/Elastic

│ │

└─────────┬──────────┬──────┘

│ │

┌─────▼────┐ ┌───▼────────┐

│ Kafka │ │ AI (Perplx)│

└──────────┘ └────────────┘