EC Site Project

Technical Documentation & Project Schedule

Full Stack Engineer Portfolio Project

Next.js + NestJS + AWS (EC2, RDS, S3, CloudFront) + Stripe

2025

1. Project Overview

This document describes the technical architecture, project structure, requirements, and 6-month development schedule for a full-stack EC (E-Commerce) site. The project is designed as a portfolio piece targeting Japanese companies in startup and enterprise markets.

1.1 Goals

- Build a production-quality EC site from scratch
- Demonstrate fullstack skills: Next.js + NestJS + AWS
- Learn AWS services: EC2, RDS, S3, CloudFront
- Integrate Stripe payment processing
- Deploy with CI/CD via GitHub Actions

1.2 Tech Stack

Frontend	Next.js 14 + TypeScript	React framework with App Router, SSR/SSG
Styling	Tailwind CSS	Utility-first CSS framework
State	Zustand	Lightweight cart state management
Data Fetching	React Query (TanStack)	API caching and loading states
Backend	NestJS + TypeScript	Structured Node.js API framework
ORM	Prisma	Type-safe database access
Database	AWS RDS (PostgreSQL)	Managed relational database
Storage	AWS S3	Product image storage
CDN	AWS CloudFront	Fast image/asset delivery
Hosting (BE)	AWS EC2 (t2.micro)	Backend server
Hosting (FE)	Vercel	Frontend deployment

Auth	JWT (JSON Web Token)	Stateless authentication
Payments	Stripe	Payment processing + webhooks
CI/CD	GitHub Actions	Automated deployment to EC2
Process Mgr	PM2	Keep NestJS running on EC2
Reverse Proxy	Nginx	Port forwarding + HTTPS on EC2

1.3 Architecture Diagram

User Flow:

```
User Browser
  ↓  HTTPS
Vercel (Next.js Frontend)
  ↓  API calls
AWS EC2 — Nginx → NestJS Backend
  ↓              ↓
AWS RDS          AWS S3 ← CloudFront
(PostgreSQL)     (Images)
  ↓
Stripe (Payment)
```

1.4 Cost Estimate

AWS EC2 t2.micro	Stop when not developing	¥100 ~ ¥500
AWS RDS t2.micro	Stop temporarily when idle	¥2,000 ~ ¥3,000
AWS S3 + CloudFront	Always Free tier	¥0
Vercel	Free tier	¥0
Stripe	Pay per transaction only	¥0 (dev)
GitHub	Free public repository	¥0
Total		¥2,100 ~ ¥3,500/month

2. Project Structure

2.1 Repository Structure (Monorepo)

```
ec-site/                            ← GitHub monorepo
```

```
├── frontend/                  ← Next.js App Router
├── backend/                   ← NestJS API
├── .github/
│   └── workflows/
│       └── deploy.yml         ← GitHub Actions CI/CD
└── README.md
```

2.2 Frontend Structure (Next.js)

```
frontend/
├── app/
│   ├── (auth)/
│   │   ├── login/page.tsx
│   │   └── register/page.tsx
│   ├── (shop)/
│   │   ├── page.tsx           ← Product listing (SSG)
│   │   ├── products/[id]/page.tsx← Product detail (SSR)
│   │   ├── cart/page.tsx
│   │   ├── checkout/page.tsx
│   │   └── orders/page.tsx    ← Order history
│   ├── admin/
│   │   ├── products/page.tsx  ← Add/edit products
│   │   └── orders/page.tsx    ← Manage orders
│   └── layout.tsx
├── components/
│   ├── ui/                    ← Button, Input, Modal...
│   ├── product/               ← ProductCard, ProductList...
│   ├── cart/                  ← CartItem, CartSummary...
│   └── admin/                 ← AdminTable, AdminForm...
├── lib/
│   ├── api.ts                 ← API call helpers
│   └── stripe.ts              ← Stripe client setup
├── store/
│   └── cartStore.ts           ← Zustand cart state
└── .env.local
    ├── NEXT_PUBLIC_API_URL     ← EC2 backend URL
    └── NEXT_PUBLIC_STRIPE_KEY  ← Stripe publishable key
```

2.3 Backend Structure (NestJS)

```
backend/
├── src/
```

```
│   ├── auth/
│   │   ├── auth.module.ts
│   │   ├── auth.controller.ts    ← POST /auth/login, /register
│   │   ├── auth.service.ts
│   │   └── jwt.strategy.ts       ← JWT validation
│   ├── users/
│   │   ├── users.module.ts
│   │   ├── users.controller.ts
│   │   └── users.service.ts
│   ├── products/
│   │   ├── products.module.ts
│   │   ├── products.controller.ts← GET/POST/PUT/DELETE /products
│   │   └── products.service.ts
│   ├── orders/
│   │   ├── orders.module.ts
│   │   ├── orders.controller.ts  ← GET/POST /orders
│   │   └── orders.service.ts
│   ├── payments/
│   │   ├── payments.module.ts
│   │   ├── payments.controller.ts← Stripe webhook handler
│   │   └── payments.service.ts
│   ├── upload/
│   │   ├── upload.module.ts
│   │   ├── upload.controller.ts  ← POST /upload (S3)
│   │   └── upload.service.ts     ← AWS S3 SDK
│   ├── app.module.ts
│   └── main.ts
├── prisma/
│   └── schema.prisma             ← Database schema
└── .env
    ├── DATABASE_URL              ← RDS connection string
    ├── JWT_SECRET
    ├── STRIPE_SECRET_KEY
    ├── STRIPE_WEBHOOK_SECRET
    ├── AWS_ACCESS_KEY_ID
    ├── AWS_SECRET_ACCESS_KEY
    ├── AWS_REGION
    └── AWS_S3_BUCKET_NAME
```

2.4 Database Schema (Prisma)

```
model User {
  id        String   @id @default(uuid())
  email     String   @unique
  password  String
  role      Role     @default(USER)
  orders    Order[]
  createdAt DateTime @default(now())
}

model Product {
  id          String      @id @default(uuid())
  name        String
  description String
  price       Int
  stock       Int
  imageUrl    String      // CloudFront URL
  orderItems  OrderItem[]
  createdAt   DateTime    @default(now())
}

model Order {
  id          String      @id @default(uuid())
  user        User        @relation(fields: [userId], references: [id])
  userId      String
  status      OrderStatus @default(PENDING)
  totalPrice  Int
  items       OrderItem[]
  createdAt   DateTime    @default(now())
}

model OrderItem {
  id        String  @id @default(uuid())
  order     Order   @relation(fields: [orderId], references: [id])
  orderId   String
  product   Product @relation(fields: [productId], references: [id])
  productId String
  quantity  Int
  price     Int
}

enum Role         { USER  ADMIN }
enum OrderStatus  { PENDING  PAID  SHIPPED  DELIVERED  CANCELLED }
```

3. Features & Requirements

3.1 User Features

User Registration	Email + password signup with validation	High
User Login	JWT-based authentication, token stored in httpOnly cookie	High
Product Listing	Browse all products with search and filter	High
Product Detail	View product details, images, price, stock	High
Shopping Cart	Add/remove items, adjust quantity (Zustand state)	High
Checkout	Review cart, enter shipping info, proceed to payment	High
Stripe Payment	Secure payment via Stripe hosted checkout	High
Order Confirmation	Success page after payment with order summary	High
Order History	View past orders and their status	Medium
User Profile	View and update account info	Low

3.2 Admin Features

Product Create	Add new product with name, price, stock, image upload to S3	High
Product Edit	Update product details and images	High
Product Delete	Remove products (soft delete)	High
Order Management	View all orders, update status (Shipped, Delivered etc.)	High
Image Upload	Upload product images directly to AWS S3	High
Dashboard	Overview of sales, orders, products count	Medium

3.3 API Endpoints (NestJS)

POST	/auth/register	Register new user	None
POST	/auth/login	Login, returns JWT token	None
GET	/products	Get all products (with search)	None
GET	/products/:id	Get single product	None
POST	/products	Create product (admin)	Admin JWT
PUT	/products/:id	Update product (admin)	Admin JWT
DELETE	/products/:id	Delete product (admin)	Admin JWT
POST	/orders	Create new order	User JWT
GET	/orders	Get user's orders	User JWT
GET	/orders/:id	Get single order	User JWT
GET	/admin/orders	Get all orders (admin)	Admin JWT
PUT	/admin/orders/:id	Update order status	Admin JWT
POST	/upload	Upload image to S3	Admin JWT
POST	/payments/webhook	Stripe webhook handler	Stripe Sig
POST	/payments/checkout	Create Stripe session	User JWT

3.4 Payment Flow

Step-by-step Stripe payment process:

- User clicks 'Checkout' button on cart page
- Frontend calls POST /payments/checkout to NestJS
- NestJS creates Stripe Checkout Session, returns URL
- Frontend redirects user to Stripe hosted checkout page
- User completes payment on Stripe page
- Stripe redirects user back to success page
- Stripe sends webhook event (payment_intent.succeeded) to POST /payments/webhook
- NestJS verifies webhook signature, updates order status to PAID in RDS

4. AWS Setup Guide

4.1 AWS Services Summary

EC2 t2.micro	Run NestJS backend	Ubuntu 22.04, Node.js 20, PM2, Nginx
RDS t2.micro	PostgreSQL database	PostgreSQL 15, private subnet

S3 Bucket	Store product images	Private bucket, accessed via CloudFront
CloudFront	CDN for images	Origin: S3 bucket, HTTPS enabled

4.2 EC2 Setup Steps

- Launch EC2 t2.micro instance (Ubuntu 22.04)
- Create Security Group: allow port 22 (SSH), 80 (HTTP), 443 (HTTPS)
- Assign Elastic IP for fixed public IP address
- SSH into instance and install: Node.js 20, PM2, Nginx
- Clone GitHub repository on EC2
- Set environment variables (.env file)
- Configure Nginx as reverse proxy (port 80 → port 3000)
- Install SSL certificate with Certbot (free HTTPS)
- Start NestJS with PM2: pm2 start dist/main.js --name ec-site

4.3 RDS Setup Steps

- Create RDS instance: PostgreSQL 15, t2.micro
- Place in same VPC as EC2
- Create Security Group: allow port 5432 from EC2 Security Group only
- Note connection endpoint, username, password
- Add DATABASE_URL to EC2 .env file
- Run Prisma migrations: npx prisma migrate deploy
- Stop RDS instance when not developing to save cost

4.4 S3 + CloudFront Setup Steps

- Create S3 bucket (private, block all public access)
- Create CloudFront distribution with S3 as origin
- Create IAM user with S3 access, generate Access Key + Secret
- Add AWS credentials to EC2 .env file
- Test image upload via POST /upload endpoint
- Images served via: https://[cloudfront-id].cloudfront.net/[filename]

4.5 Cost Control Tips

- Set AWS billing alert at $10 immediately after account creation
- Stop EC2 instance when not developing (only pay for storage ~$0.10/month)

- Stop RDS temporarily when not developing (saves ~70% cost)
- After 6 months, migrate RDS to Supabase free tier (only change DATABASE_URL)
- Monitor usage at: AWS Console → Billing → Free Tier

5. GitHub & CI/CD

5.1 Branch Strategy

main	Production-ready code	EC2 + Vercel (auto)
develop	Development / testing	Local only
feature/xxx	Individual features	Local only

5.2 GitHub Secrets Required

EC2_HOST	EC2 Elastic IP address	SSH connection
EC2_SSH_KEY	EC2 private key (.pem content)	SSH authentication
AWS_ACCESS_KEY_ID	IAM user access key	S3 upload
AWS_SECRET_ACCESS_KEY	IAM user secret key	S3 upload
DATABASE_URL	RDS connection string	Prisma database
JWT_SECRET	Random secure string	JWT token signing
STRIPE_SECRET_KEY	Stripe secret key	Payment processing
STRIPE_WEBHOOK_SECRET	Stripe webhook secret	Webhook validation

5.3 GitHub Actions Workflow

```
# .github/workflows/deploy.yml
name: Deploy to EC2
on:
  push:
    branches: [main]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Deploy to EC2 via SSH
        uses: appleboy/ssh-action@v0.1.5
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ubuntu
          key: ${{ secrets.EC2_SSH_KEY }}
          script: |
            cd ec-site/backend
            git pull origin main
            npm install
            npm run build
            pm2 restart all
```

6. 6-Month Development Schedule

Month 1 — Foundation Setup

Goal: Local development environment fully working with AWS RDS connection

Week 1	Create GitHub monorepo, setup Next.js frontend	Running Next.js locally
Week 2	Setup NestJS backend, install Prisma	Running NestJS locally
Week 3	Create AWS RDS instance, connect via Prisma	DB connected, migrations running
Week 4	Build Products API (GET all, GET by ID), test with Postman	Working Products API

AWS tasks this month: Create RDS instance, note credentials

Earn AWS credit: Complete 'Configure RDS database' activity (+$20)

Month 2 — Products + Image Upload

Goal: Products fully working with images stored on S3

Week 1	Setup AWS S3 bucket + CloudFront distribution	S3 + CDN configured
Week 2	Build image upload API (NestJS → S3)	Images uploading to S3
Week 3	Build product listing page (Next.js SSG)	Product list page working
Week 4	Build product detail page (Next.js SSR)	Product detail page working

AWS tasks this month: S3 bucket, CloudFront, IAM user for S3 access

Earn AWS credit: Complete 'Launch EC2 instance' activity (+$20)

Month 3 — Authentication + Cart

Goal: Users can register, login, and manage shopping cart

Week 1	Build user registration + login API (JWT)	Auth API working
Week 2	Build login/register pages (Next.js)	Auth pages working
Week 3	Build cart with Zustand (add, remove, quantity)	Cart state working
Week 4	Build checkout page, connect cart to checkout	Checkout page working

Month 4 — Payments + Orders

Goal: Real payments working with Stripe, orders saved to RDS

Week 1	Setup Stripe account, integrate Stripe checkout session	Stripe checkout working
Week 2	Build Stripe webhook handler in NestJS	Payment confirmation working
Week 3	Build order creation and save to RDS after payment	Orders saving to DB
Week 4	Build order history page for users	Order history working

Earn AWS credit: Complete 'Build Lambda function' activity (+$20)

Month 5 — Admin Panel + Polish

Goal: Admin can manage products and orders, UI is polished

Week 1	Build admin product management (create, edit, delete)	Admin products page
Week 2	Build admin order management (view, update status)	Admin orders page
Week 3	UI polish: loading states, error handling, responsive design	Production-quality UI
Week 4	Setup GitHub Actions CI/CD workflow	Auto-deploy on git push

Earn AWS credit: Complete 'Use Amazon Bedrock' activity (+$20)

Month 6 — Deployment + Job Apply

Goal: Site fully deployed on AWS, ready to show to employers

Week 1	Launch EC2, setup Nginx + PM2, deploy NestJS	NestJS live on EC2
Week 2	Deploy Next.js to Vercel, connect to EC2 API	Full site live
Week 3	Setup Route 53 domain, SSL certificate (Certbot)	Custom domain + HTTPS
Week 4	Write README, record demo video, update portfolio	Ready to apply!

Earn AWS credit: Complete 'Set up AWS Budget' activity (+$20)

7. After 6 Months — Migration Plan

After the AWS free tier period ends, migrate RDS to Supabase to reduce cost to nearly zero.

7.1 Migration Steps

- Create Supabase account and new project (free)
- Export data from RDS: pg_dump [rds-url] > backup.sql

- Import to Supabase: psql [supabase-url] < backup.sql
- Change DATABASE_URL in GitHub Secrets to Supabase URL
- Redeploy via GitHub Actions — no code changes needed

7.2 Cost After Migration

EC2 t2.micro (stop when idle)	¥100 ~ ¥500
Supabase (replaces RDS)	¥0 (free forever up to 500MB)
S3 + CloudFront (always free)	¥0
Vercel	¥0
Total	¥100 ~ ¥500/month

8. Resume & Interview Notes

8.1 Skills to List on Resume

Frontend	Next.js, React, TypeScript, Tailwind CSS, Zustand, React Query
Backend	NestJS, Node.js, TypeScript, REST API, JWT Auth, Prisma ORM
Database	PostgreSQL, AWS RDS, Prisma migrations, SQL
AWS	EC2, RDS, S3, CloudFront, IAM
DevOps	GitHub Actions, CI/CD, PM2, Nginx, Linux, SSH
Payment	Stripe, Webhook handling
Tools	Git, GitHub, Postman, VS Code

8.2 Interview Topics to Prepare

- Why did you choose Next.js App Router over Pages Router?
- Explain SSR vs SSG — when do you use each?
- How does JWT authentication work?
- What is Prisma and why use it over raw SQL?
- Explain the Stripe webhook flow
- What is the difference between EC2 and S3?
- How does CloudFront improve performance?

- What is Nginx and why do you use it in front of NestJS?
- How does GitHub Actions automate your deployment?
- How would you scale this application if traffic increased?