High level Architecture

P04:TradeUp

<team member names & ids>

Student ID	Name
26100355	M Umar Zubair
26100216	M rayyan khan
25100137	M Raiyaan Junaid Hamid
26100200	M Ahmad
26100204	M.Shahmir Sher Qazi

Table of Contents

1. Introduction

The project is a **stock trading simulation platform** that enables hands-on learning for people who want to gain trading experience without the risk of losing money. The platform is intended to provide experiential learning value to users, to supplement their stock trading learning journeys.

Our platform's objectives are to provide a safe and risk-free trading environment that pulls real-time stock market data to simulate the real thing – meaning our environment will capture and simulate the market's movement as it moves in real-time. Furthermore, we aim to provide valuable learning experiences to users with our platform that engage and empower them to improve their stock trading abilities and confidence.

Proposed Feature Set

- Buying/Selling Stocks
- Gamified Leaderboard and Topic-Specific Channels
- AI Insights and Chatbot
- AI News Sentiment Analysis
- Candlestick Charts View
- MarketWatch: to track favorited stocks
- Educational content
- Personalised Notification Systems

2. Non-functional requirements/Quality attributes of the system

<List down non-functional requirements of your system here. List security requirements in a separate section.>

1	The system should not utilize more than 4 GB of memory at any time during its execution.
2	System should be up 99% of the time. The system should not fail more than 2 times every 24 hours. In case of a failure, the system should restore to normal operations within 15 minutes of a failure.
3	The system should follow best UI practices as per Jakob Nielsen's 10 usability heuristics and Ben Schneiderman's 8 golden rules.
4	Deploy minor bug fixes or functional modifications within 48 hours, and major bug fixes within 72.
5	The system should be easy to maintain and update with modularity in code and components, and following clean code best principles to ensure a neat code base.
6	Stock market data should be incoming in real-time to simulate the real stock market effectively
7	Leaderboard should be updated everyday after market closes

3. Security Requirements

Sr#	Security Risks	Potential Losses	Controls
1	A01:2021 – Broken Access Control	Data loss. Trust loss. Business loss.	Deny by default protocol (unless deliberate access granted)
2	A02:2021 – Cryptographic Failures	Data loss. Trust loss. Business secrets loss. Business loss.	Ensure sensitive data is encrypted in database.

			Ensure data is encrypted when transferring over a transfer protocol.
3	A03:2021 – Injection	Data loss. Database corruption. Trust loss. Database functionality loss. Business loss.	Always “sanitize” or clean up what users type, so harmful stuff is never used directly. Use special “parameterized” queries that keep input and commands separate. Never build command strings by gluing together user input and code.
4	A07:2021 – Identification and Authentication Failures	User account penetration and sensitive information compromise. Business loss.	Implement multi-factor auth. Implement weak password checks. Don’t expose session identifier in URL.
5	A09:2021 – Security Logging and Monitoring Failures	Data loss. Business loss.	Ensure contextual logs are logged for all points of entry in the system.
6	ML02:2023 Data Poisoning Attack	Bad training of the AI model. Incorrect predictions.	Use verifiable and well reputed data. Ensure limited access to DB for AI model.
7	A06:2021-Vulnerable and Outdated Components	Hackable. Constant Bugs	Use updated frameworks and libraries

4. Project Risk Management

4.1 Potential Project Risks and Mitigation Strategies

< A risk is a probability that some adverse circumstances will occur that will have negative impact on your project. Assume that the following risks may possibly occur. Suggest how you can mitigate each of the risks, i.e., what would be your strategy to avoid or minimize the effects of risks. Refer to the slides about risk management for examples.>

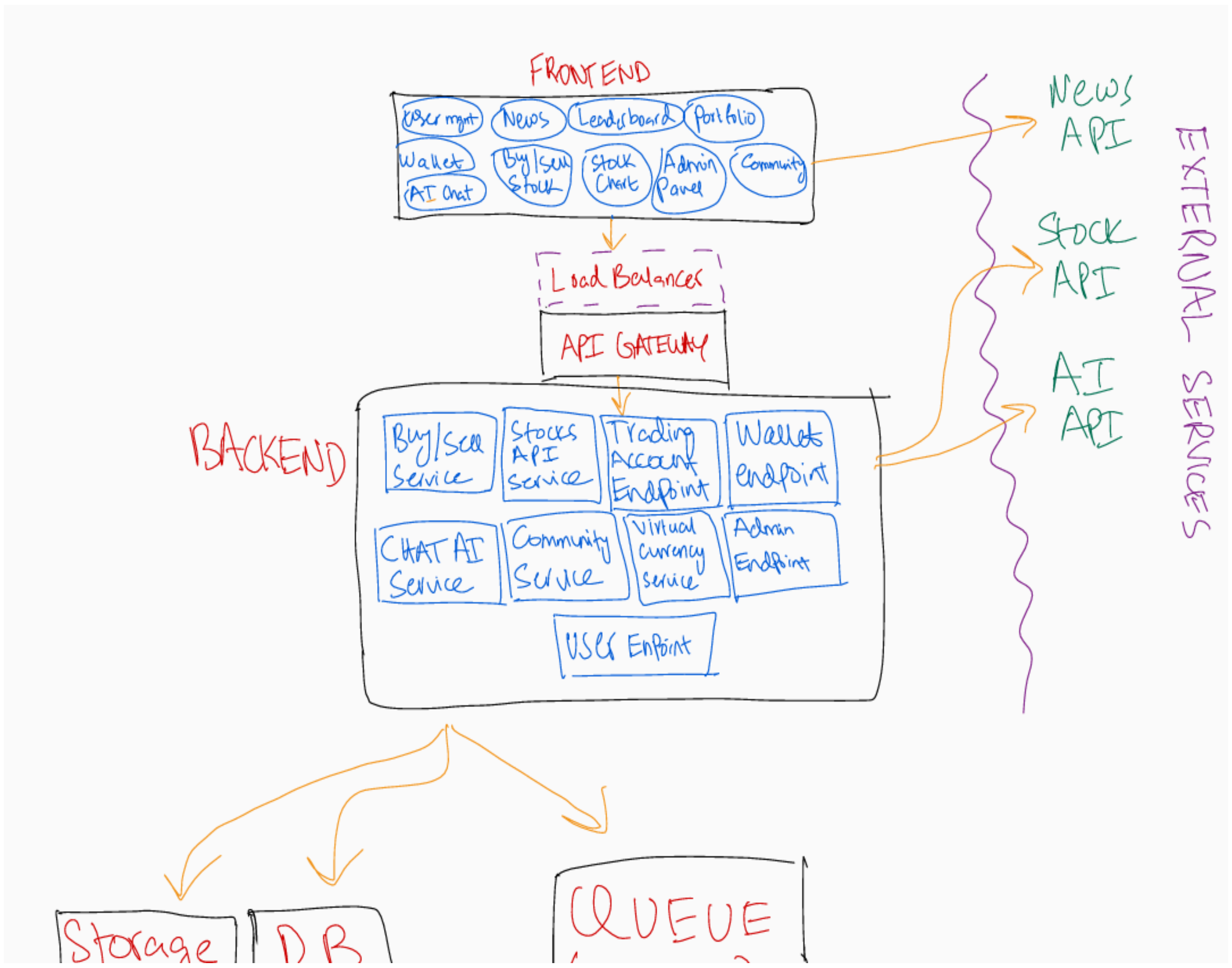
Sr.	Risk Description	Mitigation Strategy
1.	**Staff turnover** Experienced staff leaves the project before it is finished.	Pre-Risk Mitigation: - Ensure periodic team reviews of development and code-base, where front-end, backend, and dev-ops teams engage with each other and share learnings and new development updates with each other - Require lead staff to document their planning / ideas in easy to understand formats such as diagrams to ensure high-level understanding across the team Post-Risk Mitigation: - Identify where the team lacks in skill / resource knowledge and appoint the next most experienced staff member to lead the team by filling in their knowledge gaps and helping the team do the same - Leverage AI's domain knowledge (under supervision of next most experienced staff member) to support knowledge and experience gaps left behind
2.	**Requirements change** Changes in requirements that require major design rework are proposed.	- Identify what existing designs can be adapted / transferred over for the new and modified requirements (for re-using and saving time)

3.	**Underestimation** The size of the system is underestimated.	- Identify the system's core value offerings. Then match functional requirements and use cases to them to narrow down the most important features to develop and deliver. Filter out the use cases and features that don't align with the system's core value offerings (like quality of life, design-focused, or extra features)
4.	**Technology change** The underlying technology on which the system is built is superseded by a new technology.	- Assign one staff member who is familiar with the technology to start a phase-wise implementation of upgrading the underlying system to the new system. Ensure a version control system is used for development and testing to protect from system-wide malfunctions or breakages.
5.	**Code generation** The code generated by generative AI is inefficient.	- If using AI, then employ chain-of-thought prompting and other multi-shot prompting techniques to improve on the efficiency of the code. Identify also where in performance (time, space, resource consumption) is the inefficiency cropping up and prompt the AI model to solve for it specifically
6.	**Data** Required data is not available. The required data may be for training of ML Model or for some other purpose.	- Search data repositories like Kaggle, UC Irvine ML Repo, Open Data Registry on AWS, Google Dataset Search, and US Gov Open Data. - If data still not found, re-evaluate the utility of the data and model in question. - If required, design a data collection survey or study to collect primary data for your use-cases.
7.	**Stakeholder management** Customers fail to understand the impact of requirements change.	- Distill the workload and challenges associated with requirements change into easy to understand, layman language and present in a stakeholders meeting - Stakeholders are also concerned about time as well as money. Use the increased expenditure or loss of these resources to make a case against changing requirements. - Compromise on essentials from a product-value perspective only.

8.	**Off-the-shelf components and libraries** Software components/libraries that were planned to be used do not contain desired features or contain defects, i.e., they cannot be used as planned.	- If time and availability of skills and resources allows, build the missing functionality internally. Leverage generative AI heavily.
9.	**Scalability Issues** As the user base grows, the system might not handle the increased load effectively, leading to performance degradation.	- Use performance testing to simulate high load scenarios early in development. Implement horizontal scaling and load balancing using AWS Auto Scaling to dynamically adjust resources based on demand. - Monitor system performance using tools like AWS CloudWatch. If scalability issues arise, optimize the architecture (e.g., database indexing, microservices).
10	**Data Privacy Breaches** User data might be exposed due to vulnerabilities or mishandling of sensitive information.	**-** Implement strong encryption protocols for data storage and transfer. Use secure authentication mechanisms like OAuth2 or multi-factor authentication. - In case of a breach, quickly activate the incident response plan, notify affected users, and patch the security vulnerabilities.
11		

5. System Architecture

5.1 Architecture Diagram



5.2 Architecture Description

MVC Architecture

Frontend:

- **Candle-Stick/Stocks View:** The main component that displays information on all stocks in a candlestick view and other details interacts with the buy/sell service in the backend, initiating buy/sell orders, and communicates with the stocks API to retrieve stock information.
- **Portfolio View:** Displays users' purchased stocks and favourite stocks, interacting with the database through the backend using basic CRUD operations.
- **News Information:** Shows daily important news fetched through the news api This directly calls the api, but the content is cached in a Redis DB for faster fetch and fewer API calls
- **Leaderboard View:** Tracks and displays top users based on trading performance or achievements.A single user’s rank, score, and performance stats on a leaderboard.
- **Community View:** Forums to interact with other users on the app
- **Wallet:** Allows user to add virtual credits to their trading account/app account through the virtual currency service
- **Admin View:** A specialized user with elevated privileges to manage users, content, and system functions.
- **User Account Management View:** Represents a platform user with personal info, login credentials, subscription type, and activity tracking
- **Authentication View:** Login/Signup System interacts directly with the backend /DB to authenticate the user

Backend:

User Service: Manages user registration, authentication, and profiles. Handles account creation, login, password resets, and subscription upgrades, interacting directly with the database and authentication tokens.

Trading Service: Handles virtual trading operations such as executing buy/sell orders, updating balances, and calculating portfolio performance. Communicates with the Market Data Service and User Service.

Market Data Service: Fetches and updates real-time and historical stock data through external APIs. Caches prices and market info for efficient retrieval by trading-related services.

AI & Analytics Service: Provides market insights, trade recommendations, and sentiment analysis based on user behavior and news data. Integrates with the Chat Service and Market Data Service.

Leaderboard Service: Tracks achievements and leaderboards. Updates user rankings and rewards, interacting with the User and Notification Services.

Payment Service: Manages subscription billing, in-app currency purchases, and refunds. Connects securely with third-party payment gateways and updates user balances.

Notification Service: Sends alerts and reminders for trades, portfolio updates, achievements, and system messages. Uses queues and schedulers for reliable delivery.

Chat & Communication Service: Handles user and AI chat sessions, storing message history and conversation state. Integrates with the AI & Analytics Service for intelligent responses.

5.3 Justification of the Architecture

· Pros and cons of the architecture

Aspect	Pros	Cons
Scalability	Each microservice (e.g., Buy/Sell, Chat AI, Community) can scale independently on AWS ECS using auto-scaling groups.	Requires careful scaling policy management and load balancing tuning.
Reliability & Uptime	The use of AWS ALB (Application Load Balancer), Route 53, and ECS ensures high availability (99%+ uptime) and fast recovery in case of failure.	Slightly higher operational complexity with multiple distributed services.

Performance & Memory Usage	Each containerized service runs in a lightweight environment with resource limits set (under 4 GB memory total). Aurora Serverless and Redis caching minimize memory overhead.	Performance tuning required to balance memory use among multiple services.
Maintainability & Modularity	Clear separation of services (User, Trading, AI, etc.) allows easier updates, debugging, and feature rollout without system-wide downtime.	Coordination between services requires robust API documentation and version control.
Security	AWS Secrets Manager secures API keys, database credentials, and encryption keys. Deny-by-default policies and encrypted data flows (HTTPS, SSL) mitigate access and data loss risks.	Complex security configuration needed across multiple endpoints and services.
Data Management	Aurora Serverless handles auto-scaling databases with fault tolerance; S3 provides secure object storage for logs and media.	Cross-service data consistency must be managed carefully.
Integration	The API Gateway standardizes communication between the frontend and backend, while VPC isolation limits external exposure.	Additional latency introduced by API Gateway and load balancing layers.
Resilience	Queue-based design ensures messages and trades aren’t lost if a service goes down, restoring within 15 minutes as required.	Requires monitoring tools (e.g., CloudWatch) to ensure queues are drained properly.

· Implementation of non-functional requirements in system architecture

This **microservices architecture (MVP - Monolith Architecture)** deployed on **AWS** is appropriate for the Trading Simulator Platform because it achieves the key non-functional and security goals through modularity, isolation, and managed services.

1. Memory Efficiency (Under 4 GB)

- Each ECS container has defined memory limits, ensuring the total resource allocation stays below 4 GB.
- Aurora Serverless and S3 handle large data offloading from in-memory storage, keeping runtime memory usage minimal.
- Redis caching (for stock prices and news data) minimizes repetitive API calls, improving efficiency.

2. High Availability (99% Uptime, ≤2 Failures/Day)

- AWS **Route 53** provides DNS-level failover and routing between healthy instances.
- **ALB + ECS** ensures auto-healing and blue-green deployments.
- **API Gateway** and load balancers distribute traffic evenly, preventing bottlenecks.
- **Aurora Serverless** offers automatic failover and scaling, ensuring database continuity.
- Recovery time objective (RTO) under 15 minutes achieved using ECS restart policies and CloudWatch alarms for service failures.

3. Maintainability & Modularity

- Each service (e.g., Buy/Sell Service, Community Service, Chat AI Service) is developed, deployed, and updated independently.
- Microservice modularity aligns with **clean code principles**, reducing coupling and promoting single-responsibility design.

- Rolling updates and CI/CD pipelines (e.g., CodePipeline or GitHub Actions) ensure minimal downtime during deployments.

4. Security & OWASP Alignment

OWASP Category	Risk	Mitigation via Architecture
A01:2021 – Broken Access Control	Unauthorized access to data or admin actions.	“Deny by default” access policy implemented via API Gateway & IAM roles. JWT-based authentication on all endpoints ensures only verified access.
A02:2021 – Cryptographic Failures	Data leakage, secrets loss, or MITM attacks.	Secrets Manager stores credentials and API keys securely. All communications over HTTPS/TLS. Aurora encrypts data at rest and in transit.
A03:2021 – Injection	SQL/Command injection leading to data corruption.	Services use parameterized queries via ORM (e.g., SQLAlchemy, Prisma). API Gateway sanitizes input before routing. Input validation enforced on backend microservices.

5. Reliability & Fault Tolerance

- **SQS Queues** ensure trade requests, notifications, and updates are not lost during failures.

- **Storage (S3)** and **DB (Aurora)** backups enable fast disaster recovery.

- Container orchestration via ECS with health checks and CloudWatch monitoring ensures quick auto-recovery within the 15-minute SLA.

6. Tools and Technologies

1. Development Tools and Frameworks

Category	Tool / Technology	Version
Programming Language (Backend)	**Python,Java**	3.12
Web Framework	**FastAPI,Springboot**	0.111
Frontend Framework	**Nextjs**	18.3
State Management (Frontend)	**Redux Toolkit**	2.2
Form Handling	**React hook form**	latest
UI Animations	**Framer Motion/Lottie**	v11+/Latest
UI Components	**ShadCN**	Latest
CSS Framework	**Tailwind CSS**	3.4
Database ORM	**Spring data JPA**	2.0
AI/ML Integration	**OpenAI API**	Latest
Data Analysis	**Pandas / NumPy**	2.2 / 1.26

2. Databases and Storage

Component	Tool / Service	Version	Purpose

Relational Database	**Amazon Aurora Serverless (PostgreSQL)**	Compatible with PostgreSQL 15	Main data storage for users, trades, portfolios, and achievements.
Caching Layer	**Redis (Amazon ElastiCache)**	7.x	Caching market data, news, and session data for performance.
Object Storage	**Amazon S3**	Latest	Storing user profile images, educational videos, and logs.
Message Queue	**Amazon SQS**	Latest	Asynchronous message handling between services (trade orders, notifications).

3. Deployment & Cloud Infrastructure (AWS)

Service	Tool / Service	Version
Compute & Containers	**AWS ECS (Fargate)**	Latest
Networking & Load Balancing	**AWS VPC + ALB (Application Load Balancer)**	Latest
API Management	**AWS API Gateway**	Latest
DNS & Routing	**Amazon Route 53**	Latest
Secrets Management	**AWS Secrets Manager**	Latest
Monitoring & Logging	**Amazon CloudWatch**	Latest
Container Registry	**Amazon ECR (Elastic Container Registry)**	Latest

CI/CD Pipeline	**AWS CodePipeline + CodeBuild**	Latest

4. External APIs and Integrations

Integration	Tool / API	Version	Purpose
Market Data	**PSX stocks API**	Latest	Fetching live and historical stock data.
News Feed	**NewsAPI.org**	v2	Retrieving financial and market-related news.
AI Assistant	**OpenAI API Agent RAG**	Latest	Providing natural language chat, insights, and recommendations.

5. DevOps, Security & Collaboration Tools

Category	Tool / Technology	Version
Containerization	**Docker**	27.0
Reverse Proxy / Gateway	**NGINX/ AWS ALB**	1.26
Source Control	**GitHub**	Latest
Infrastructure as Code	**Terraform / AWS CloudFormation**	Latest
Code Quality	**Prettier + ESLint / Flake8**	Latest

7. Hardware Requirements

7.1 Development Machines

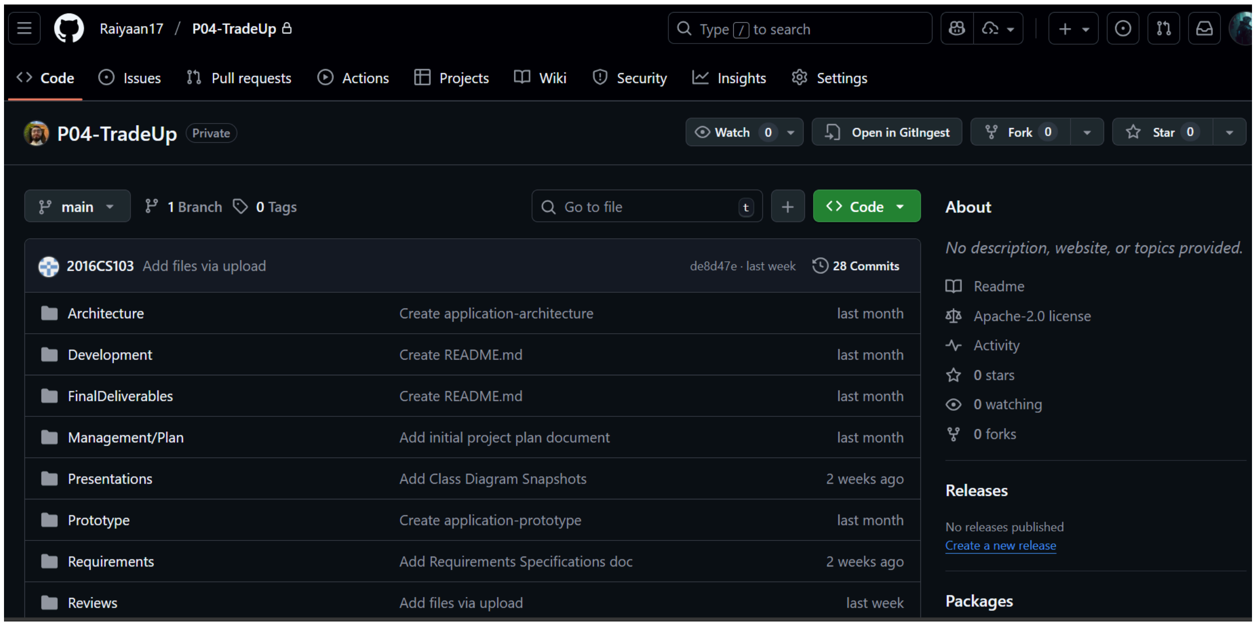
Component	Minimum Requirement	Recommended Specification
Processor (CPU)	Intel Core i5 (10th Gen) / AMD Ryzen 5 3600	Intel Core i7 (12th Gen) / AMD Ryzen 7 5800X
Memory (RAM)	8 GB	16 GB or higher
Storage	256 GB SSD	512 GB SSD or higher
Graphics (GPU)	Integrated GPU	Integrated GPU
Operating System	Windows 10 / macOS / Ubuntu 22.04	Cross-platform compatible
Internet Connection	Stable broadband (≥ 10 Mbps)	High-speed fiber (≥ 50 Mbps)

7.2 Deployment Servers

Component	Minimum Requirement	Recommended Specification
Processor (CPU)	4 vCPUs (e.g., Intel Xeon / AMD EPYC)	8 vCPUs or higher (e.g., Xeon Silver / EPYC 7002 series)
Memory (RAM)	8 GB	16–32 GB (for concurrent users, API load, and AI tasks)
Storage	256 GB SSD	512 GB NVMe SSD (for faster I/O and caching)
Graphics (GPU)	Optional (for ML inference only)	Optional (for ML inference only)
Operating System	Ubuntu Server 22.04 LTS / Debian 12	Ubuntu Server 22.04 LTS (preferred for stability and support)
Network Bandwidth	100 Mbps	1 Gbps dedicated or cloud auto-scaling network
Server Type	VPS / Cloud Instance (AWS EC2, GCP Compute Engine, Azure VM)	Managed container environment (Docker + Kubernetes / AWS ECS / GCP GKE)
Database Server	PostgreSQL / MySQL hosted instance	Managed DB service (AWS RDS / Cloud SQL)
AI/ML Integration	Cloud-based API (OpenAI / Azure Cognitive Services)	Scalable API integration with load balancing
Backup & Monitoring	Manual backup scripts	Automated backup + monitoring (Prometheus / Grafana / CloudWatch)

8. Development Environment Preparation

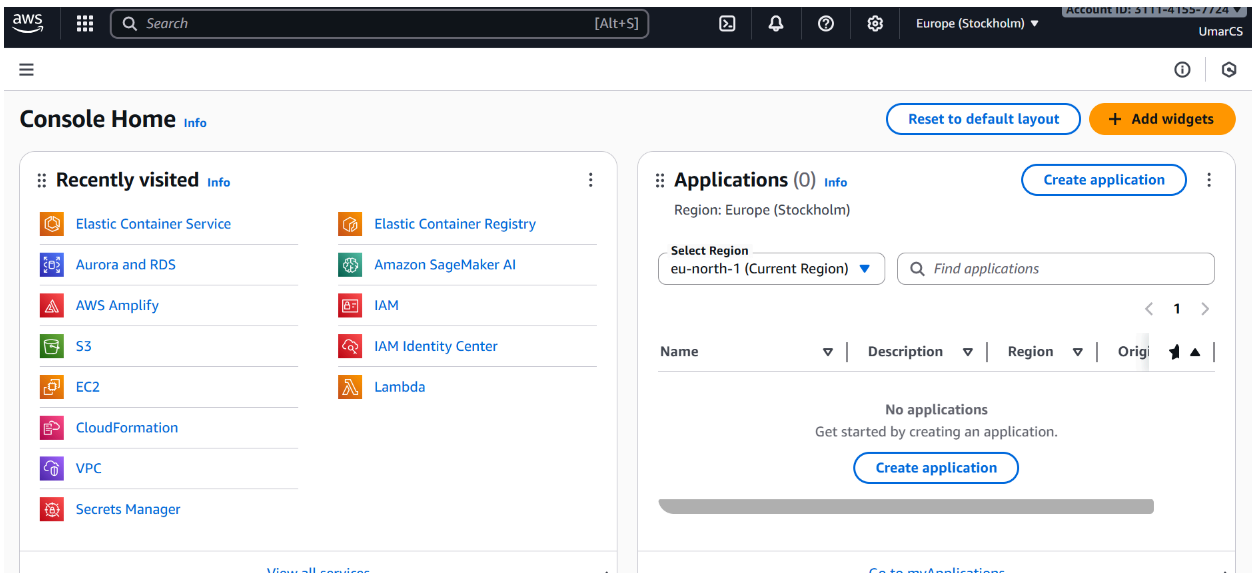
Github:



2016CS103 Add files via upload — de8d47e · last week — 28 Commits

Architecture	Create application-architecture	last month
Development	Create README.md	last month
FinalDeliverables	Create README.md	last month
Management/Plan	Add initial project plan document	last month
Presentations	Add Class Diagram Snapshots	2 weeks ago
Prototype	Create application-prototype	last month
Requirements	Add Requirements Specifications doc	2 weeks ago
Reviews	Add files via upload	last week

AWS:


Account ID: 3111-4155-7724

Europe (Stockholm)

UmarCS

Search [Alt+S]

Console Home Info

Reset to default layout | Add widgets

Recently visited Info

Elastic Container Service	Elastic Container Registry
Aurora and RDS	Amazon SageMaker AI
AWS Amplify	IAM
S3	IAM Identity Center
EC2	Lambda
CloudFormation	
VPC	
Secrets Manager	

Applications (0) Info

Create application

Region: Europe (Stockholm)

Select Region: eu-north-1 (Current Region)

Find applications

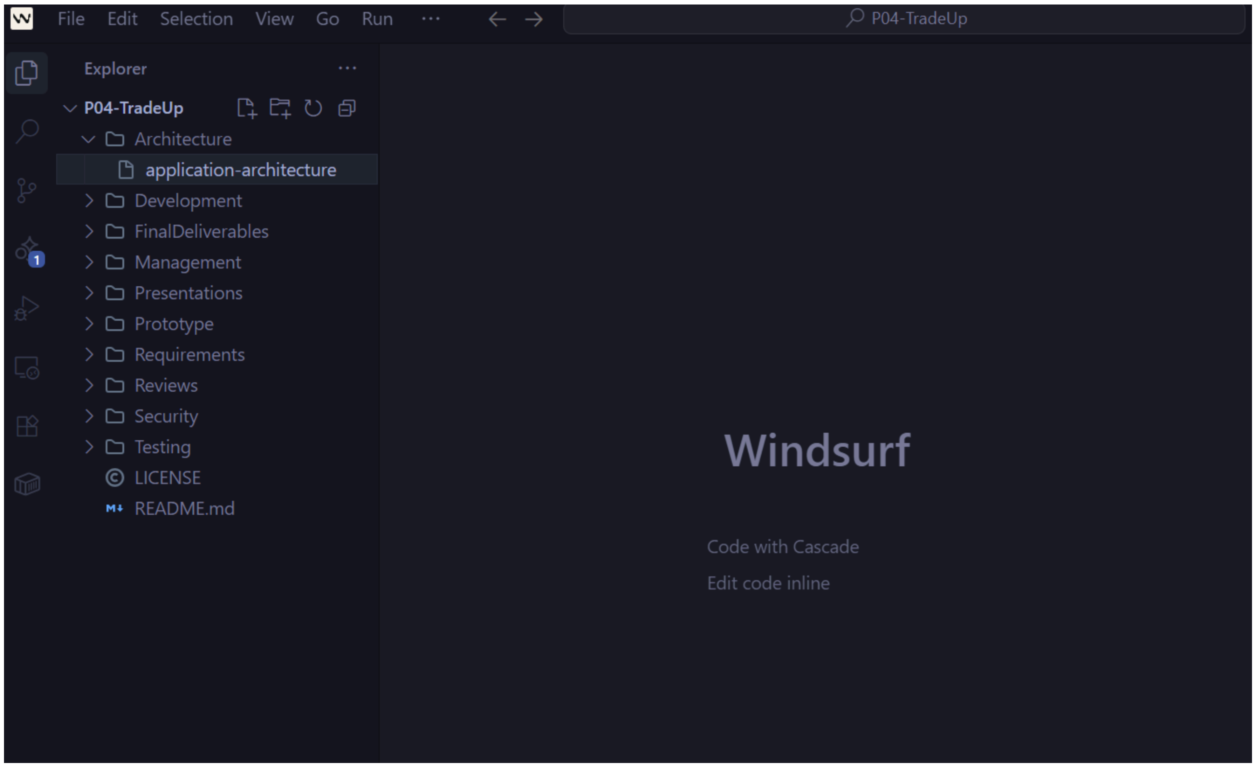
Name	Description	Region	Origi

No applications

Get started by creating an application.

Create application

IDE:


File Edit Selection View Go Run ··· P04-TradeUp

Explorer

```
P04-TradeUp
  Architecture
    application-architecture
  Development
  FinalDeliverables
  Management
  Presentations
  Prototype
  Requirements
  Reviews
  Security
  Testing
  LICENSE
  README.md
```

Windsurf

Code with Cascade

Edit code inline

9. Deployment Platform

Development: Vercel +Netlfiy
Production: AWS

10. Use of Generative AI

Used Ai to get an idea of what would be the appropriate hardware specifications for the deployment servers.

11. Who Did What?

Name of the Team Member	Tasks done
Umar Zubair	Did/Contributed to sections 5, 6,8,9
Rayyan Khan	Contributed to sections 6 and 7
Raiyaan Junaid	Contributed to section 4 and drew the architecture diagram
Muhammad Ahmad	Contributed to section 1,2,3,4
Muhammad Shahmir	Contributed to sections 5.7 and 5.8

12. Review checklist

Before submission of this deliverable, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

Section Title	**Reviewer Name(s)**
Development Environment	Muhammad Rayyan Khan
System Architecture	Muhammad Rayyan Khan
Tools & Technologies (section 6), Dev Environment Prep (section 8)	Mohammed Raiyaan Junaid Hamid
Reviewed section 4,7	M Umar Zubair
Section 4,5	M Ahmad
Section 5 and reviewed section 6	Muhammad Shahmir