GIC Team 9

Meet the Team!

Drustan Yeo
NUS Y3 CS + Math
(Backend)

Ong Jing Xuan
NTU Y4 REP CS
(Frontend)

Asher Laiu
SMU Y3 Econs + IS
(Backend)

Heng Junxiang
NUS Y3 Comp Eng
(Frontend)

Ong Jung Yi
SUTD Y3 CS
(Backend)

Table of contents

1

Tech Stack

Tech Stack

Next.js

React.js
TailwindCSS
Material UI

Spring Boot

Spring Security
Spring Cloud Gateway
Server Sent Event
Lombok

MySQL

Spring Data JPA
Adminer
Docker

Next.js

1. Declarative UI
2. Component-based
3. Clean Styling
4. Consistency

Spring Boot

1. Spring Ecosystem
2. Embedded Servers
3. Microservices Support

2

Live Demo

3

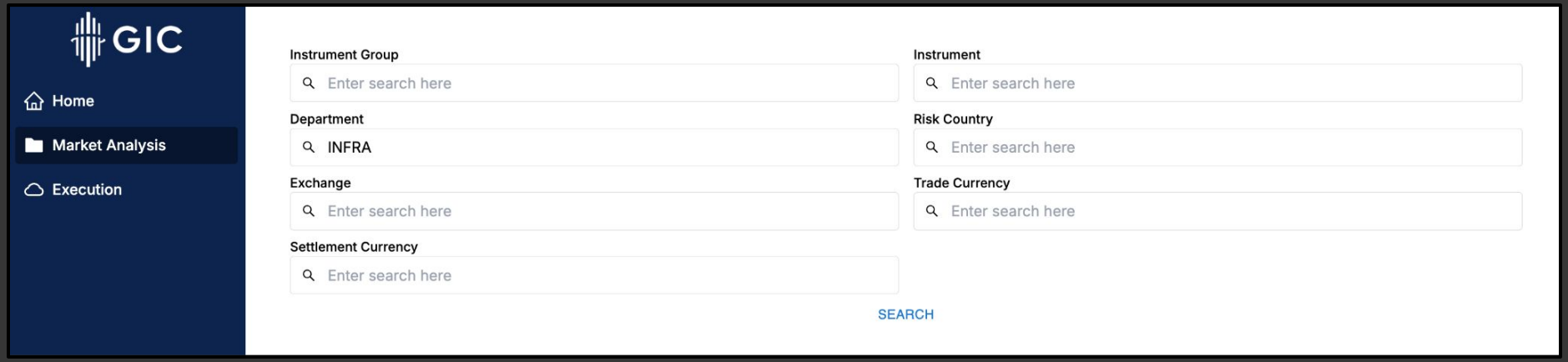
User Stories

User Story 1 & 2

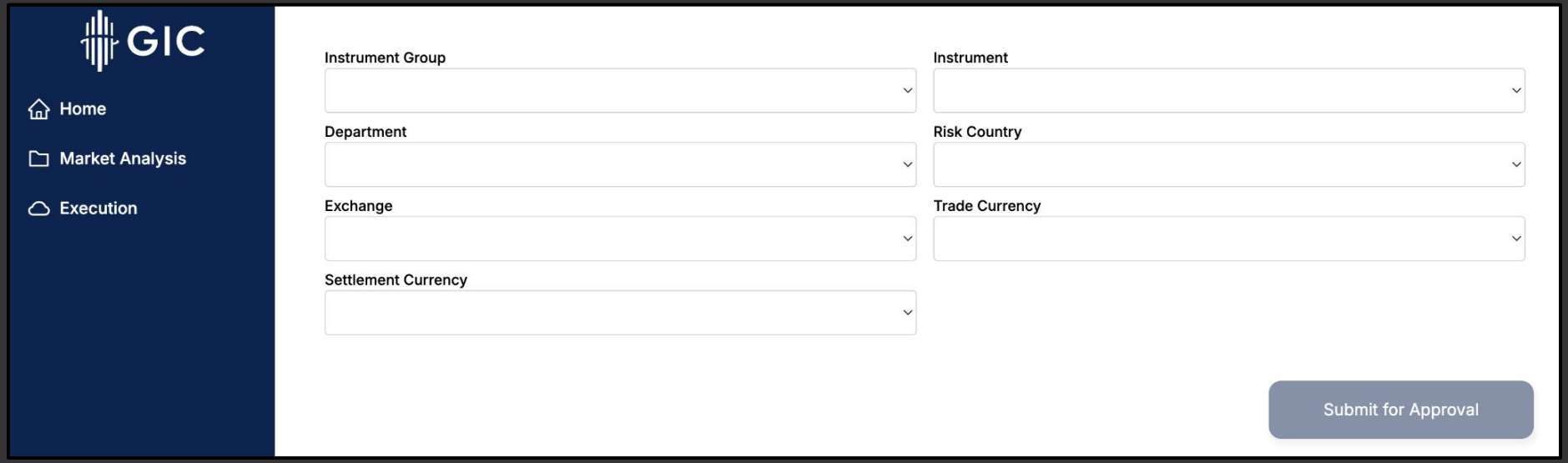
Analysis: Trader searches for verified instruments and sends request for new instruments that are not present

- System returns exact matches within 3s
- Department validation through authentication
- Request Form

UI Component



UI Component

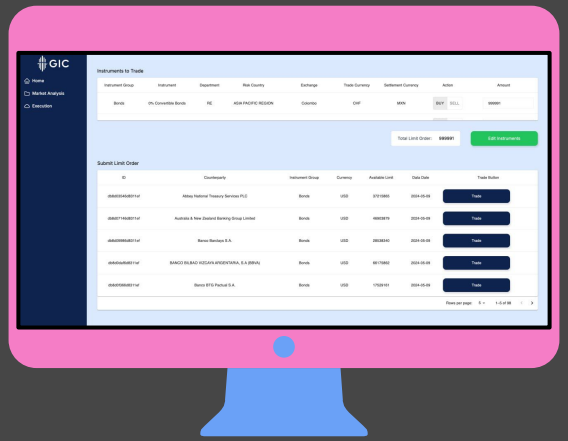


Efficient Querying of DB

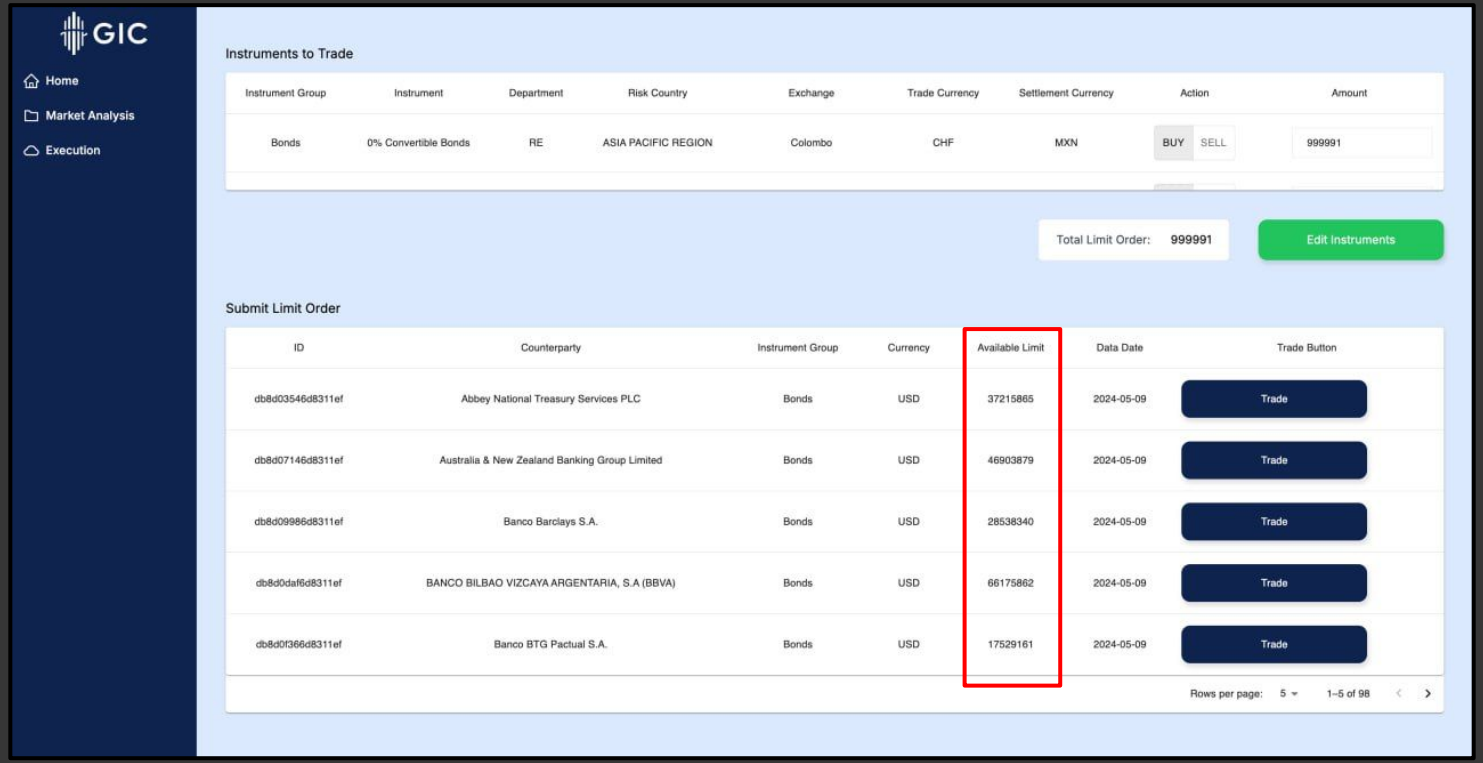
- Use only one API call for multiple search fields
- Reduces API calls to DB

User Story 3 & 4

Execution: Trader uses a reliable Real-time Limits Dashboard for Executing Trades Concurrently



UI Component

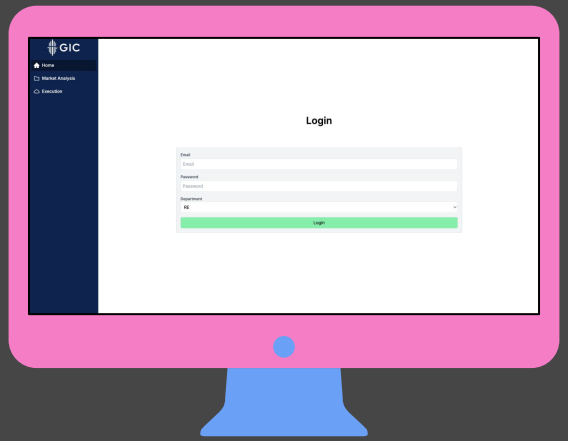


Integration With DB

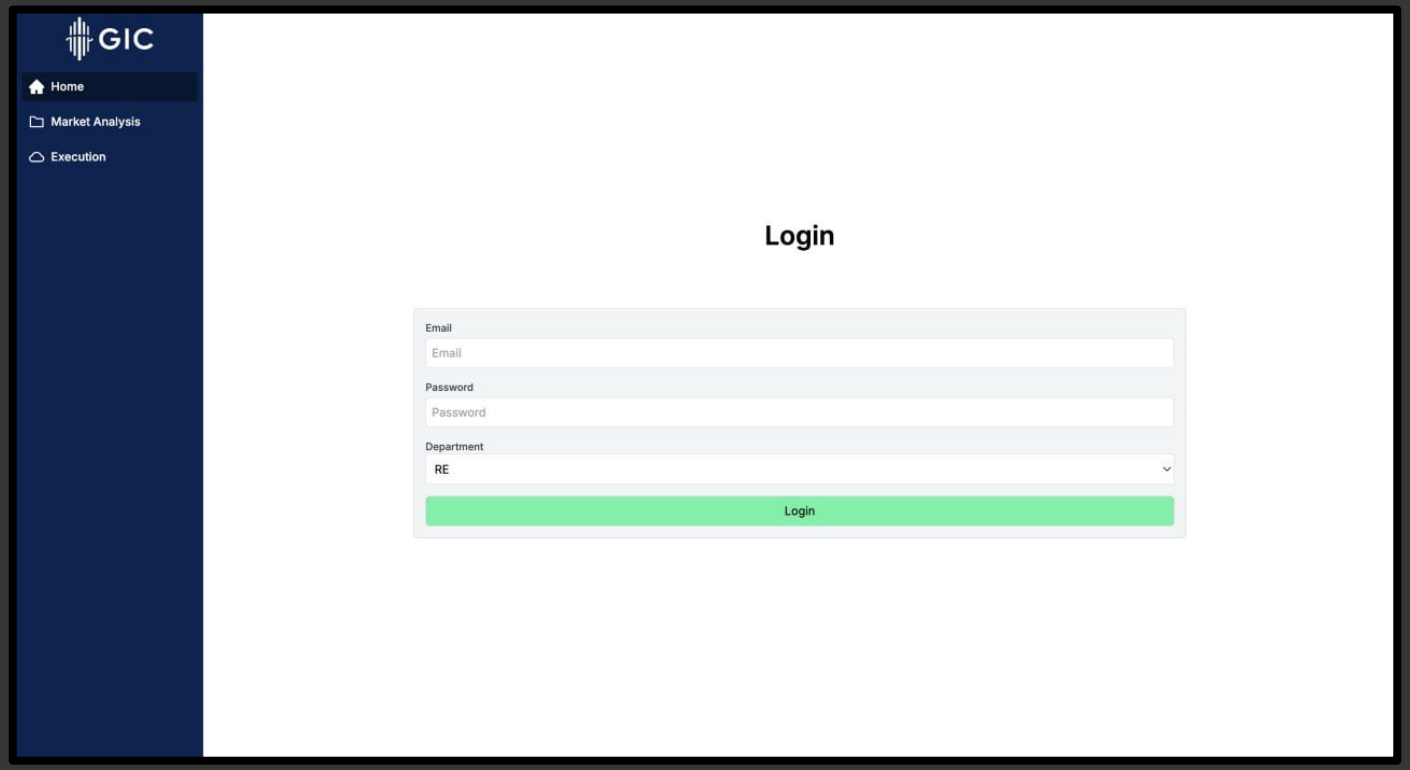
- SSE To Receive Live Updates
- Orders Are Queued
- Backend Check To Prevent Overuse

User Story 5

Integrated Platform: All-in-one webapp for Traders



UI Component

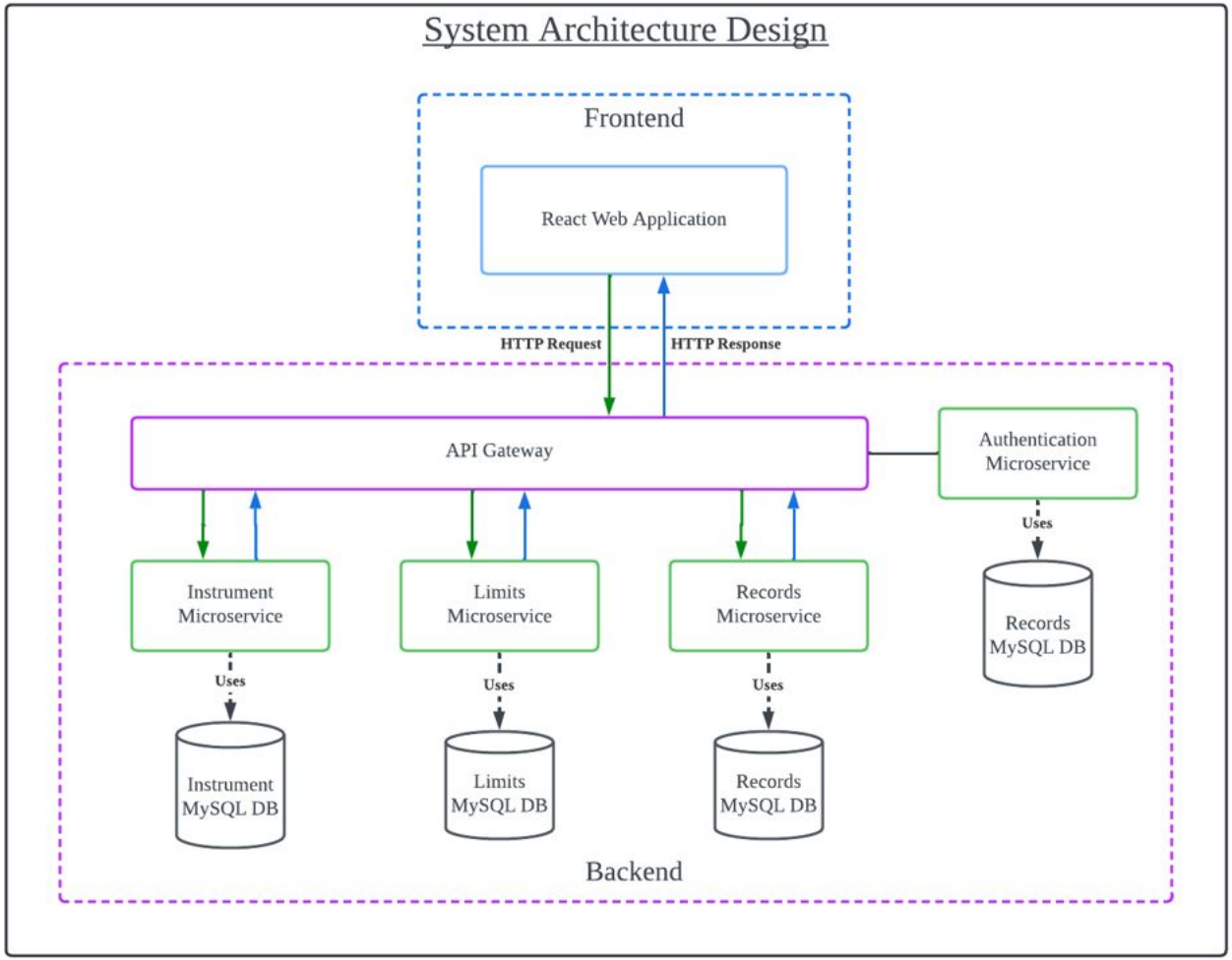


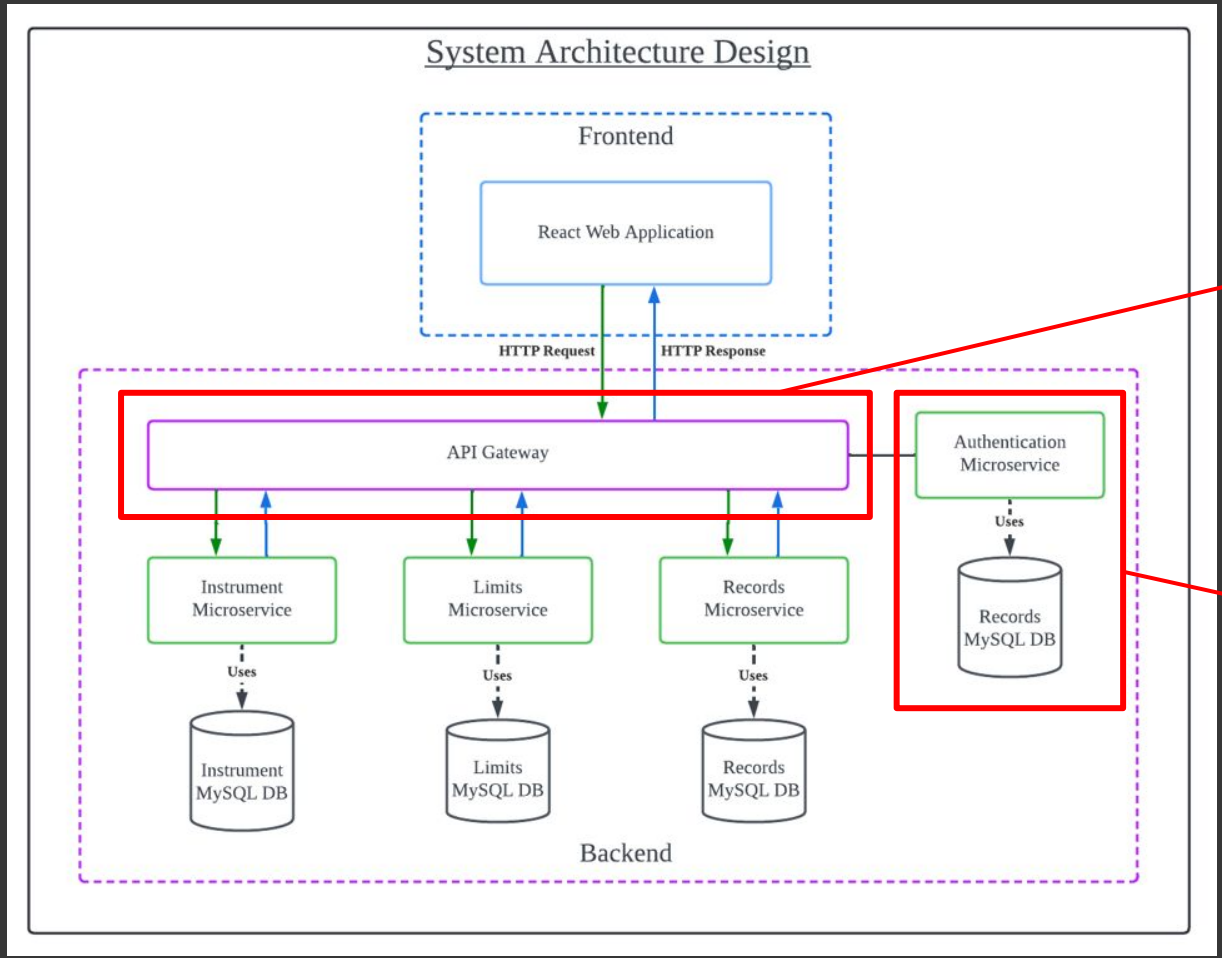
4

Architecture & Design

What is Microservices Architecture?

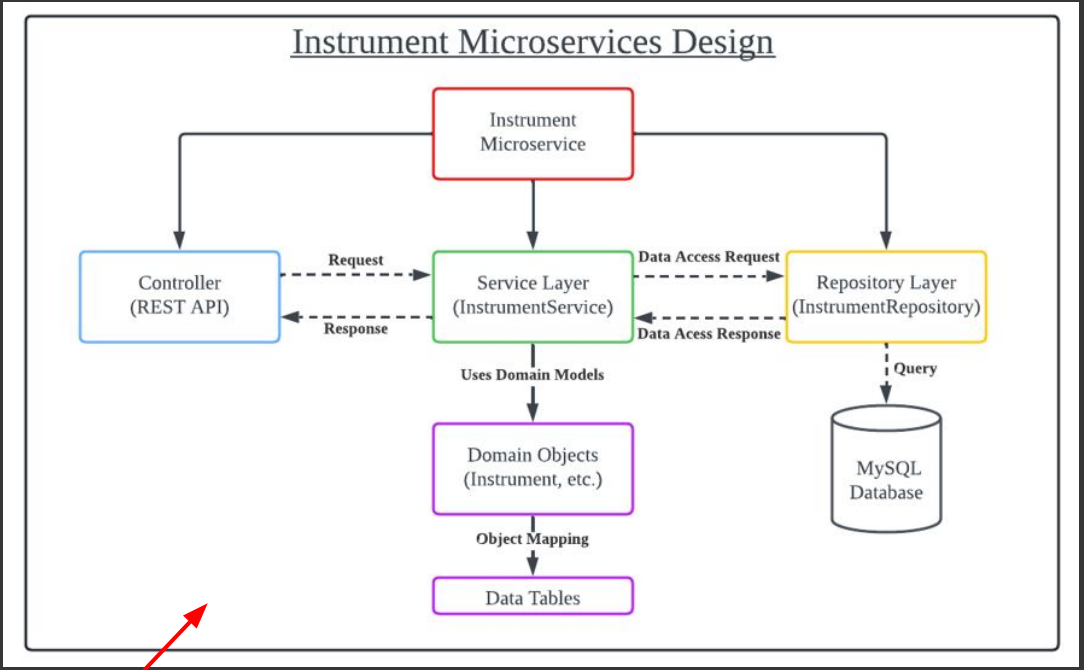
- Microservices architecture breaks down an application into **independent, modular services** that each handle a specific functionality.
- Each service is **developed, deployed, and scaled independently**, while communicating with others through APIs.

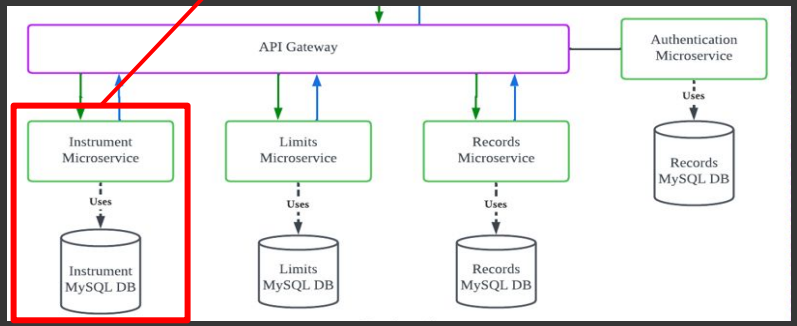

System Architecture Design
Frontend
React Web Application
HTTP Request
HTTP Response
API Gateway
Authentication Microservice
Uses
Records MySQL DB
Instrument Microservice
Limits Microservice
Records Microservice
Uses
Uses
Uses
Instrument MySQL DB
Limits MySQL DB
Records MySQL DB
Backend



API Gateway: Centralized routing point, handling authentication and traffic distribution to individual services.

Authentication Microservice: Secures the system using JWT, applied once for all services through the API gateway.

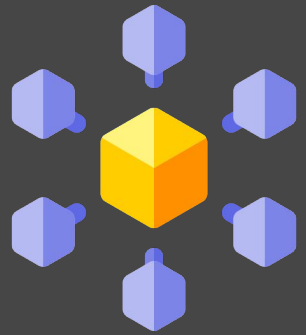




Instrument Microservice: Handles instrument search and queries.

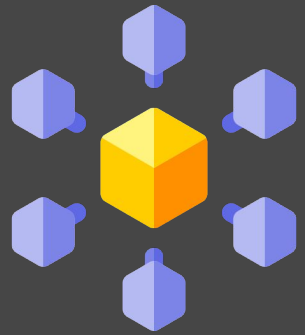
Limits Microservice: Manages limit tracking and trade execution.

Records Microservice: Logs trade approvals and user actions.



Pros

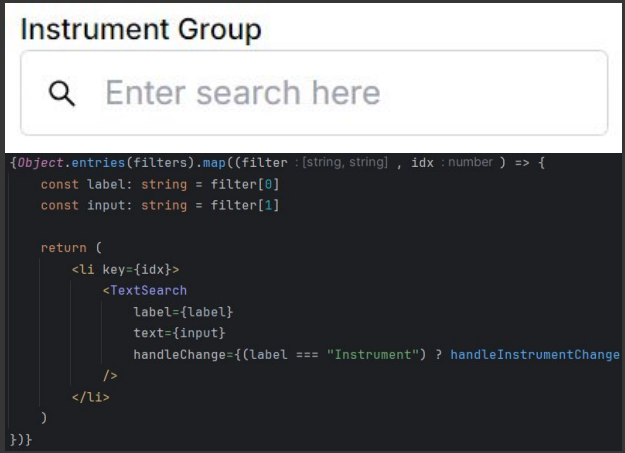
- **Scalability**: Each service can be scaled independently based on usage demand.
- **Flexibility**: We can deploy and update services without affecting the entire system.
- **Resilience**: If one service fails, others continue to function, reducing downtime.



Cons

- **Complexity**: Managing multiple services introduces complexity in terms of deployment, monitoring, and communication between services.
- **Overhead**: Requires additional infrastructure and careful management of service-to-service communication.

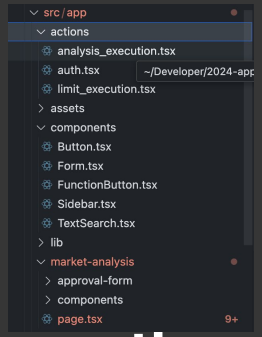
Frontend Code Design



Component-Based Architecture

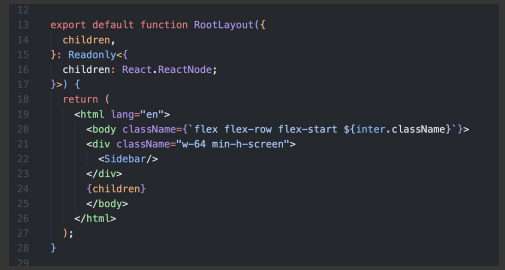
Reusable, self-contained UI components that manage their own logic
System is modular and maintainable

Frontend Code Design



Separation of Concerns

- Divides UI components into two types:
  - Container: Handles data fetching, logic, and state management.
  - Presentation: Receives data via props and focuses solely on UI display.
- Keeps code clean and consistent



```
export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">
      <body className={`flex flex-row flex-start ${inter.className}`}>
      <div className="w-64 min-h-screen">
        <Sidebar/>
      </div>
      {children}
      </body>
    </html>
  );
}
```

Code Reuse and Consistent Styling

- Reusable components
- Nextjs layout tools
- Tailwindcss configuration
- Material design guidelines

THANK YOU!