Database:

PostgreSQL and MongoDB: PostgreSQL for relational data (e.g., tickets, users) and MongoDB for flexible schema data (e.g., chat logs).

Back-End Frameworks:

NestJS: For the majority of services due to its flexibility and easy integration with GraphQL.

Spring Framework: For the Payments service due to its robust transaction management.

Front-End Frameworks:

Admin App: Angular

Client App: Angular

PrimeNG: For the Angular-based client and admin app.

Communication:

Primarily REST APIs: For simpler, clear communication between services.

GraphQL: For data-heavy querying, especially combined with Angular Apollo for reactive front-end.

Message Broker (RabbitMQ): For decoupled communication and ensuring smooth operations (e.g., for handling chat messages and notifications).

High-Level Architecture

Client -> Validation Layer -> Business Logic Layer -> DAO Layer

Here's how we can map the services:

Auth Service:

Framework: NestJS

Database: PostgreSQL

Communication: REST API

JWT with refresh tokens and future Google Auth integration

Payments Service:

Framework: Spring Framework

Database: PostgreSQL

Communication: REST API

Supports subscriptions

Company Service:

Framework: NestJS

Database: PostgreSQL for company data, MongoDB for documents

Communication: GraphQL

Manager Service:

Framework: NestJS

Database: PostgreSQL

Communication: REST API

Tickets Service:

Framework: NestJS

Database: PostgreSQL for tickets, MongoDB for human documents

Communication: REST API

Admin App:

Framework: Angular + PrimeNG

Communication: GraphQL for reactive data querying

Verification Service:

Framework: NestJS

Database: PostgreSQL

Communication: REST API

Automated email verifications, manual document verifications

Support Service:

Framework: NestJS

Database: MongoDB for chat logs

Communication: REST API and RabbitMQ for live chat

Statistic Service:

Framework: NestJS

Database: PostgreSQL for structured data

Communication: GraphQL for interactive querying

Entities:

Client

Auth Service

Company Service

Manager Service

Tickets Service

Payments Service

Verification Service

Support Service

Admin Service

Statistic Service

Sequence:

The client authenticates via the Auth Service.

The client (manager) creates a company profile including document uploads via the Company Service.

The Verification Service verifies the documents.

The Authenticated client (manager) creates a ticket through the Manager Service which interacts with the Tickets Service.

A payment is processed for the ticket via the Payments Service.

The support service (for any issues) and statistic service are used to accumulate data for the Admin Service.

Client -> AuthService: Request Authentication

AuthService -> Client: Return Token

Manager (Client) -> CompanyService: Create Company Profile (+Documents)

CompanyService -> VerificationService: Verify Documents

VerificationService -> CompanyService: Verification Result

Manager (Client) -> ManagerService: Create Ticket

ManagerService -> TicketsService: Create Ticket Entry

TicketsService -> PaymentsService: Process Payment

PaymentsService -> TicketsService: Confirm Payment

TicketsService -> ManagerService: Ticket Creation Confirmation

ManagerService -> Manager (Client): Ticket Created

Client -> SupportService: Chat for Support

SupportService -> MongoDB: Save Chat Logs

Admin (Client) -> StatisticService: Request Statistics

StatisticService -> Admin (Client): Provide Statistics

Middleware for authentication and authorization validation.

DTOs and Pipes for input validation.

Service Layer for business rule validation.