coursewagon / **SYSTEM_ARCHITECTURE.md**

Uttam Mahata Implement code changes to enhance functionality and improve performan... ✓

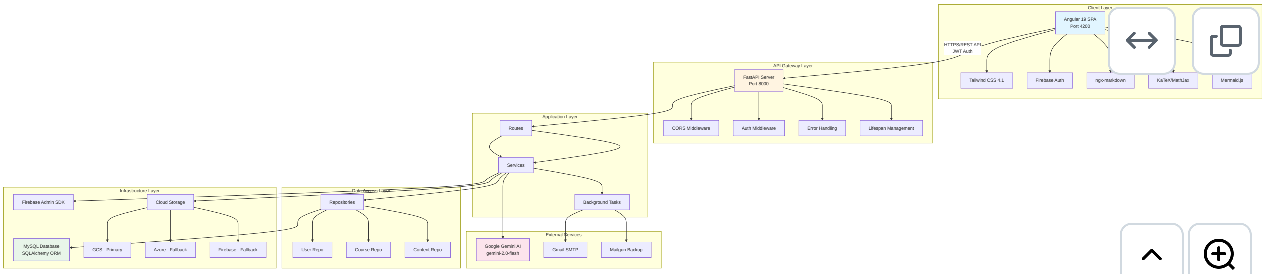
3ed3cdd · 9 minutes ago

```
1842 lines (1469 loc) · 48.2 KB
```

CourseWagon System Architecture Documentation

Table of Contents

1. High-Level System Architecture



Technology Stack Overview

Frontend:

- Framework: Angular 19
- Styling: Tailwind CSS 4.1
- Authentication: Firebase Auth
- Markdown: ngx-markdown with Prism.js
- Math: KaTeX, MathJax
- Diagrams: Mermaid.js
- Icons: Font Awesome

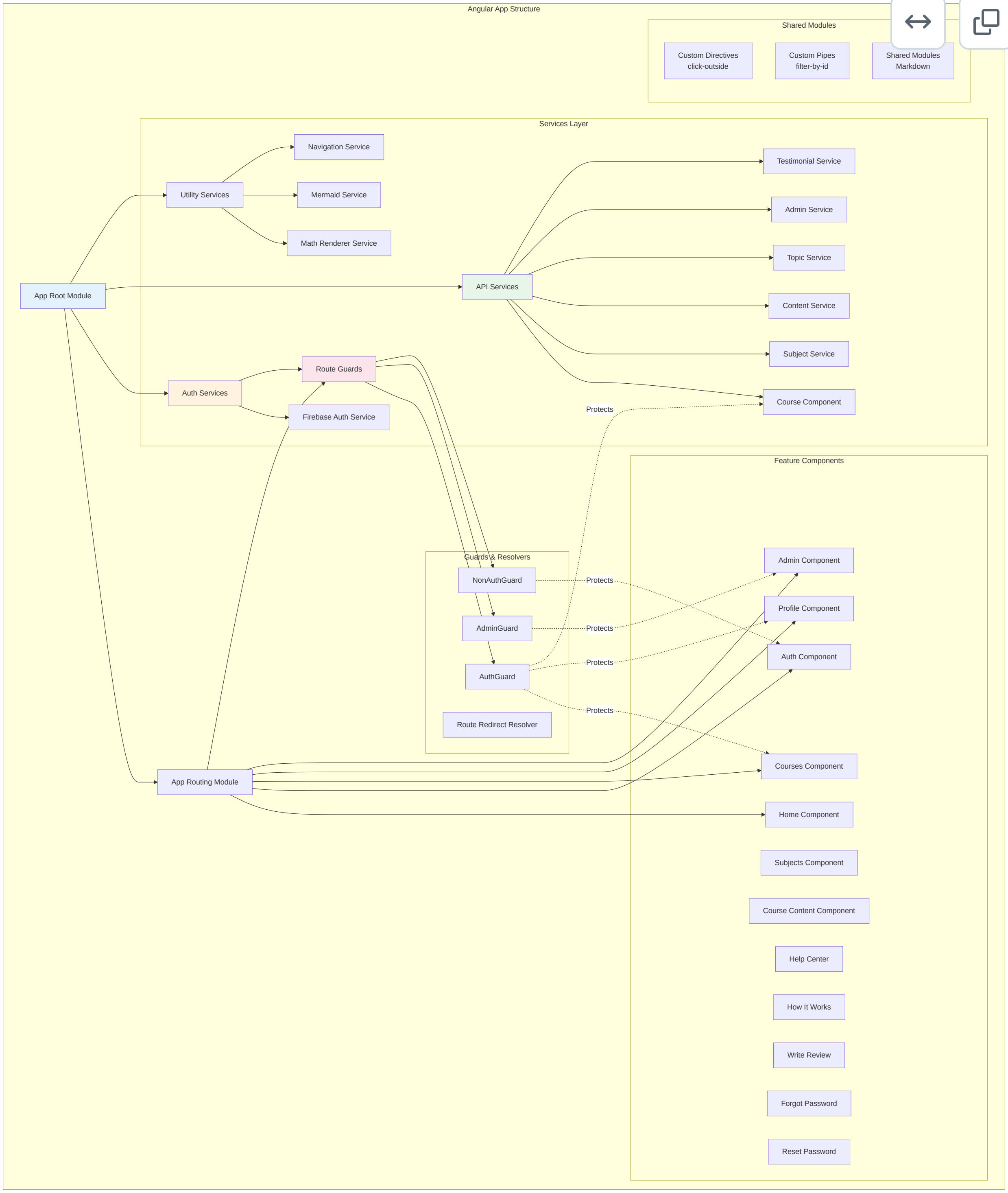
Backend:

- Framework: FastAPI (Python)
- Database: MySQL with SQLAlchemy ORM
- AI: Google Gemini AI (gemini-2.0-flash)
- Storage: Azure Blob Storage, Firebase Storage, Google Cloud Storage
- Auth: JWT tokens with Firebase Admin SDK
- Email: Gmail SMTP, Mailgun (backup)

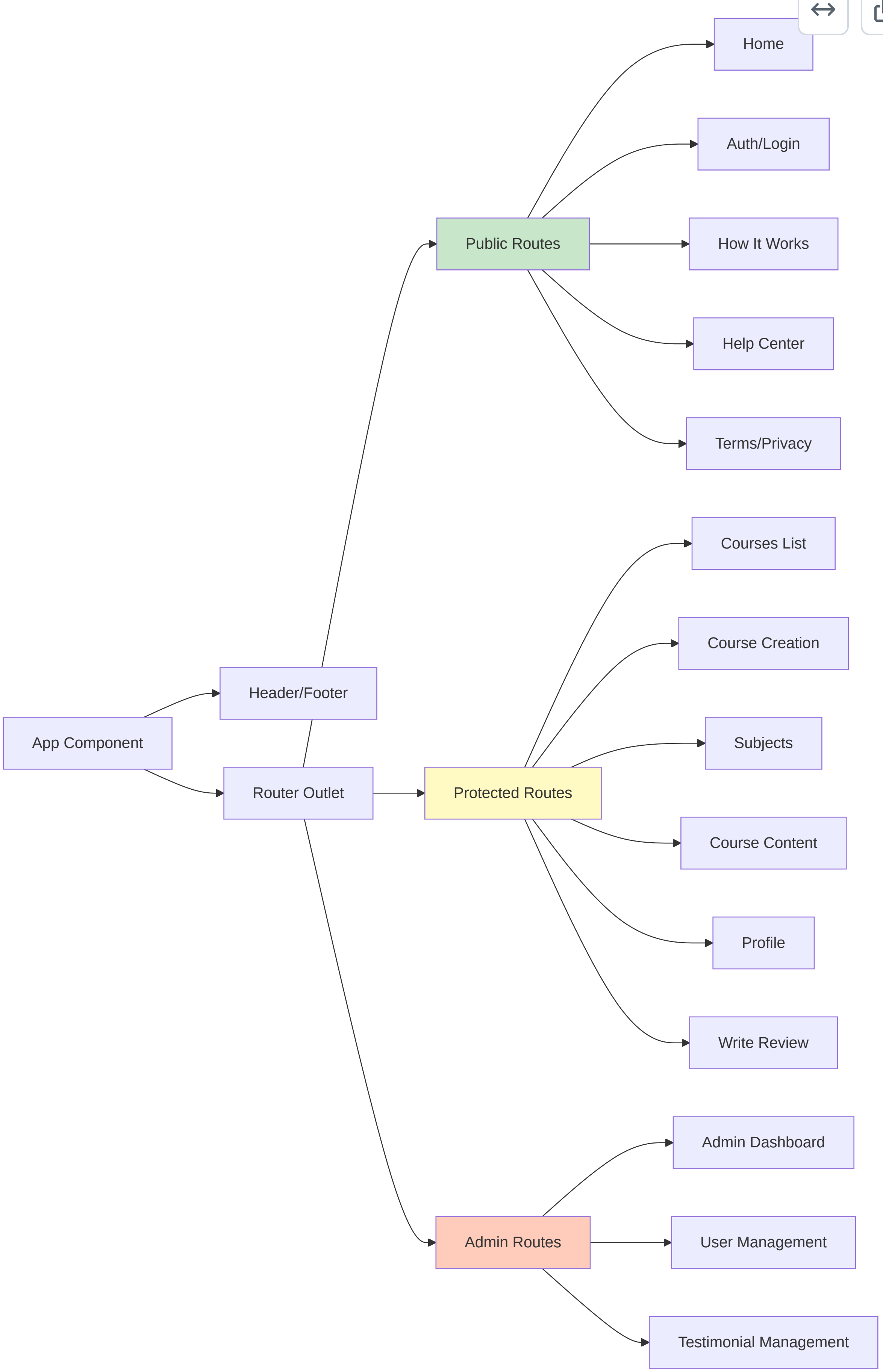
Deployment:

- Frontend: Firebase Hosting (www.coursewagon.live)
- Backend: Google Cloud Run / Azure Container Apps
- CI/CD: GitHub Actions

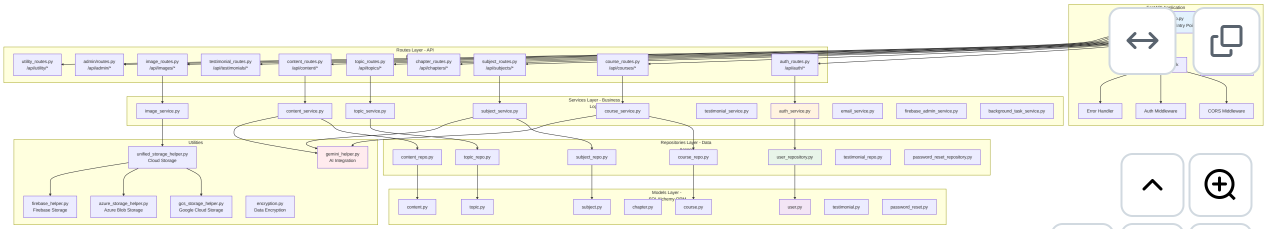
2. Frontend Architecture (Angular 19)


Angular App Structure
Shared Modules
Custom Directives click-outside
Custom Pipes filter-by-id
Shared Modules Markdown
Services Layer
Navigation Service
Utility Services
Mermaid Service
Math Renderer Service
App Root Module
API Services
Testimonial Service
Admin Service
Topic Service
Content Service
Subject Service
Course Component
Route Guards
Auth Services
Firebase Auth Service
Protects
Feature Components
Guards & Resolvers
NonAuthGuard
AdminGuard
AuthGuard
Route Redirect Resolver
Protects
Protects
Protects
Protects
Admin Component
Profile Component
Auth Component
Courses Component
Home Component
Subjects Component
Course Content Component
Help Center
How It Works
Write Review
Forgot Password
Reset Password
App Routing Module

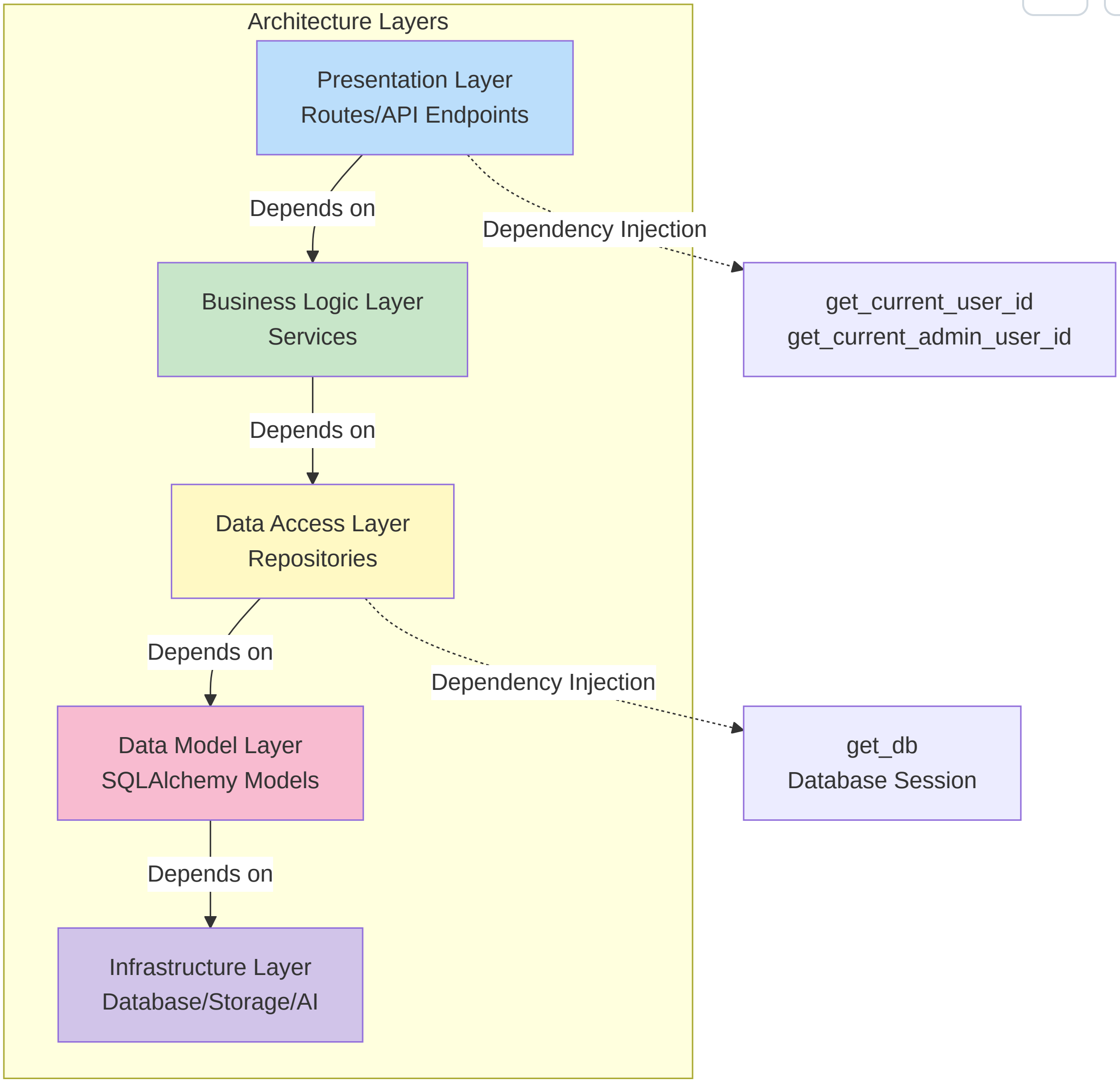
Frontend Component Hierarchy


Home
Auth/Login
Public Routes
How It Works
Help Center
Terms/Privacy
Courses List
Course Creation
Header/Footer
App Component
Subjects
Router Outlet
Protected Routes
Course Content
Profile
Write Review
Admin Dashboard
Admin Routes
User Management
Testimonial Management

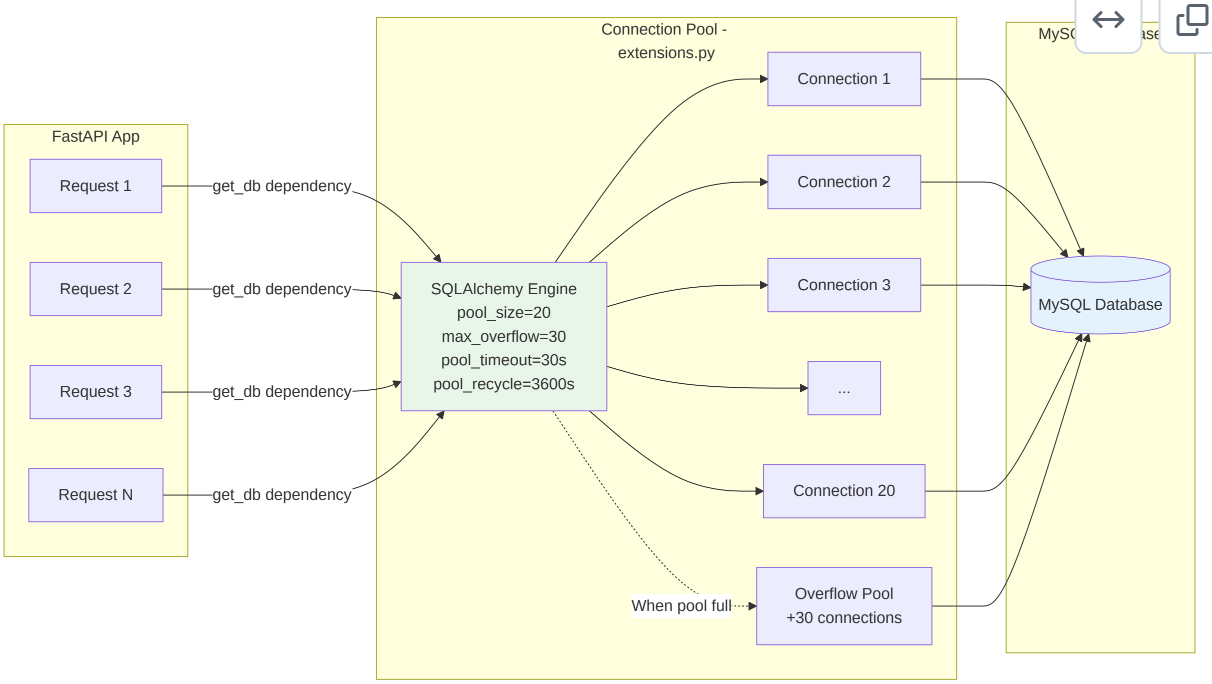
3. Backend Architecture (FastAPI + Python)



Layered Architecture Pattern

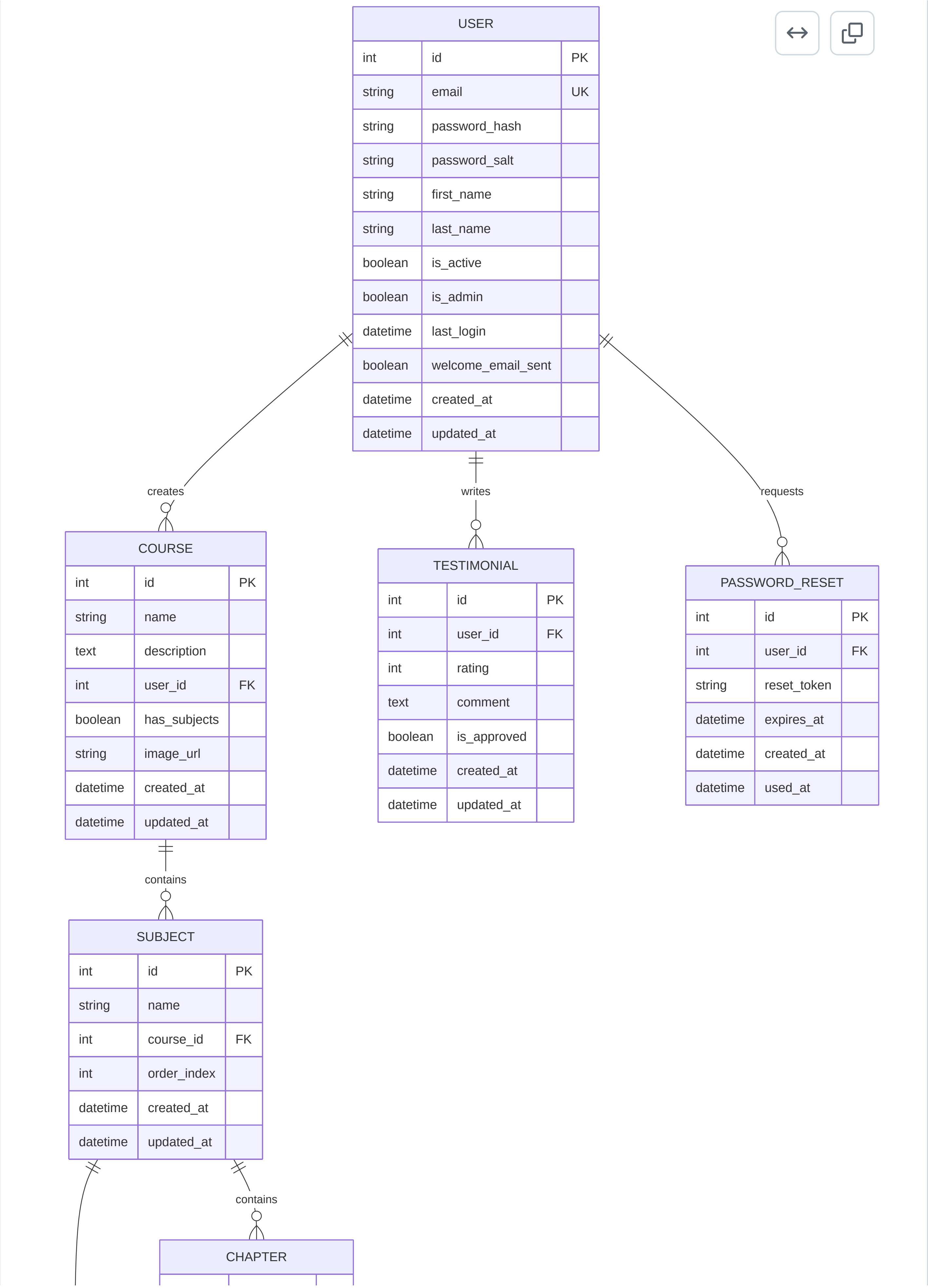


Database Connection Pooling



4. Database Architecture (MySQL + SQLAlchemy)

Entity Relationship Diagram



USER		
int	id	PK
string	email	UK
string	password_hash	
string	password_salt	
string	first_name	
string	last_name	
boolean	is_active	
boolean	is_admin	
datetime	last_login	
boolean	welcome_email_sent	
datetime	created_at	
datetime	updated_at	

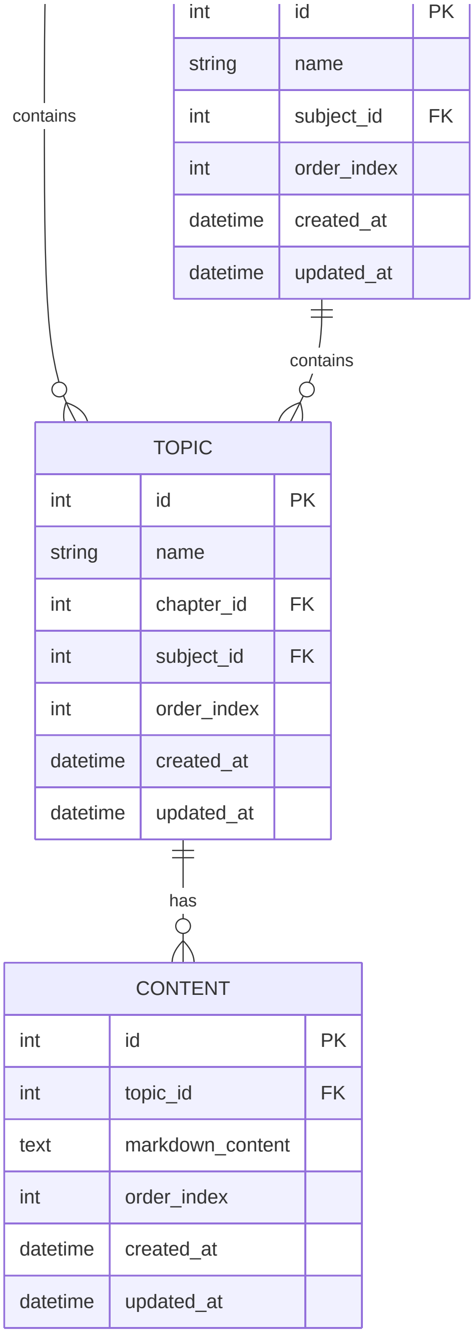
COURSE		
int	id	PK
string	name	
text	description	
int	user_id	FK
boolean	has_subjects	
string	image_url	
datetime	created_at	
datetime	updated_at	

TESTIMONIAL		
int	id	PK
int	user_id	FK
int	rating	
text	comment	
boolean	is_approved	
datetime	created_at	
datetime	updated_at	

PASSWORD_RESET		
int	id	PK
int	user_id	FK
string	reset_token	
datetime	expires_at	
datetime	created_at	
datetime	used_at	

SUBJECT		
int	id	PK
string	name	
int	course_id	FK
int	order_index	
datetime	created_at	
datetime	updated_at	

CHAPTER



int	id	PK
string	name	
int	subject_id	FK
int	order_index	
datetime	created_at	
datetime	updated_at	

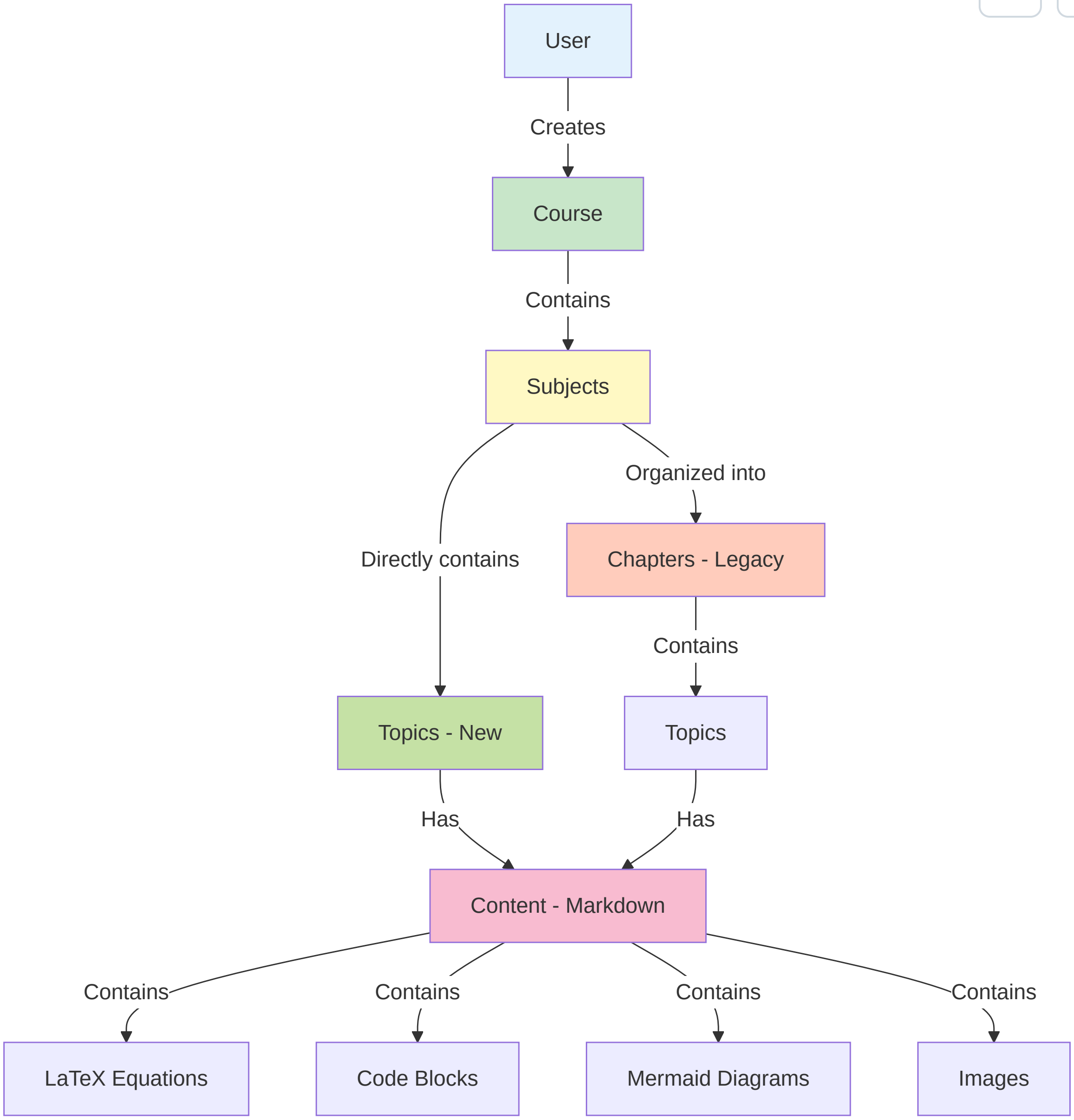
TOPIC

int	id	PK
string	name	
int	chapter_id	FK
int	subject_id	FK
int	order_index	
datetime	created_at	
datetime	updated_at	

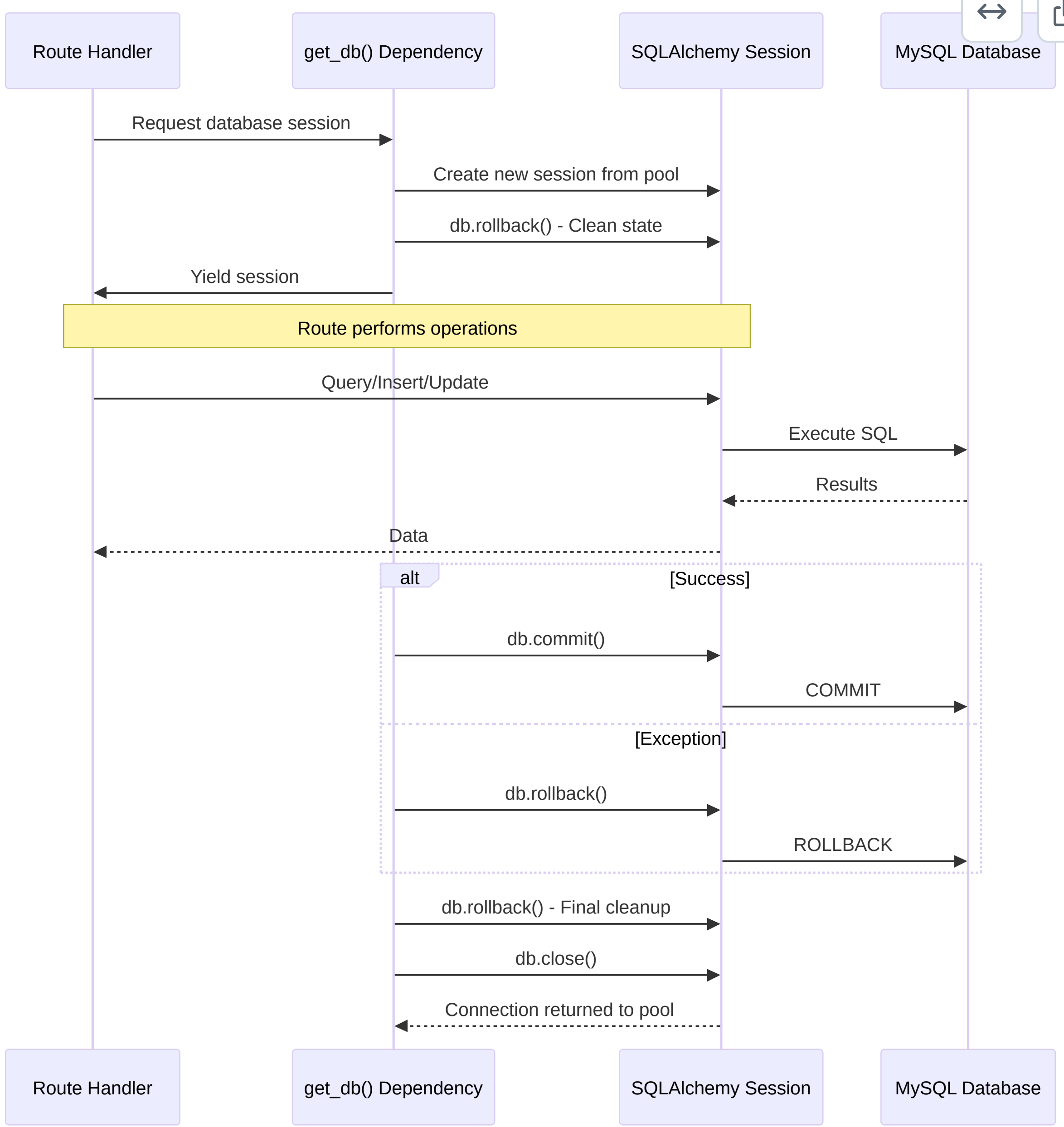
CONTENT

int	id	PK
int	topic_id	FK
text	markdown_content	
int	order_index	
datetime	created_at	
datetime	updated_at	

Data Hierarchy Flow

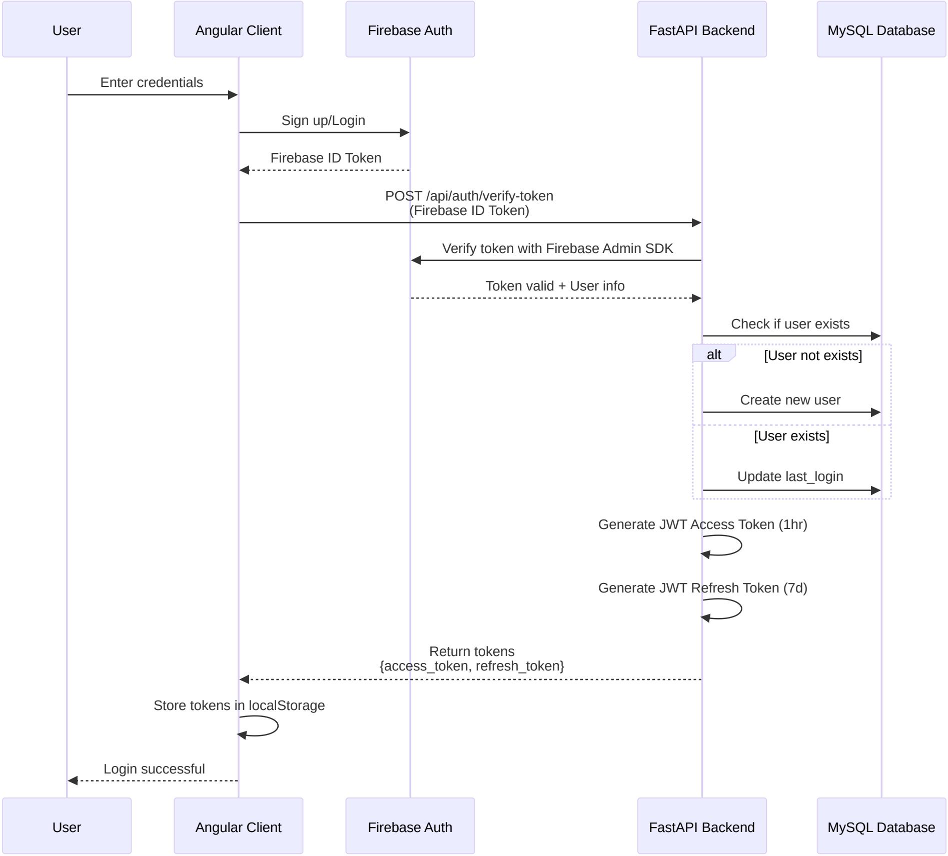


Database Session Management

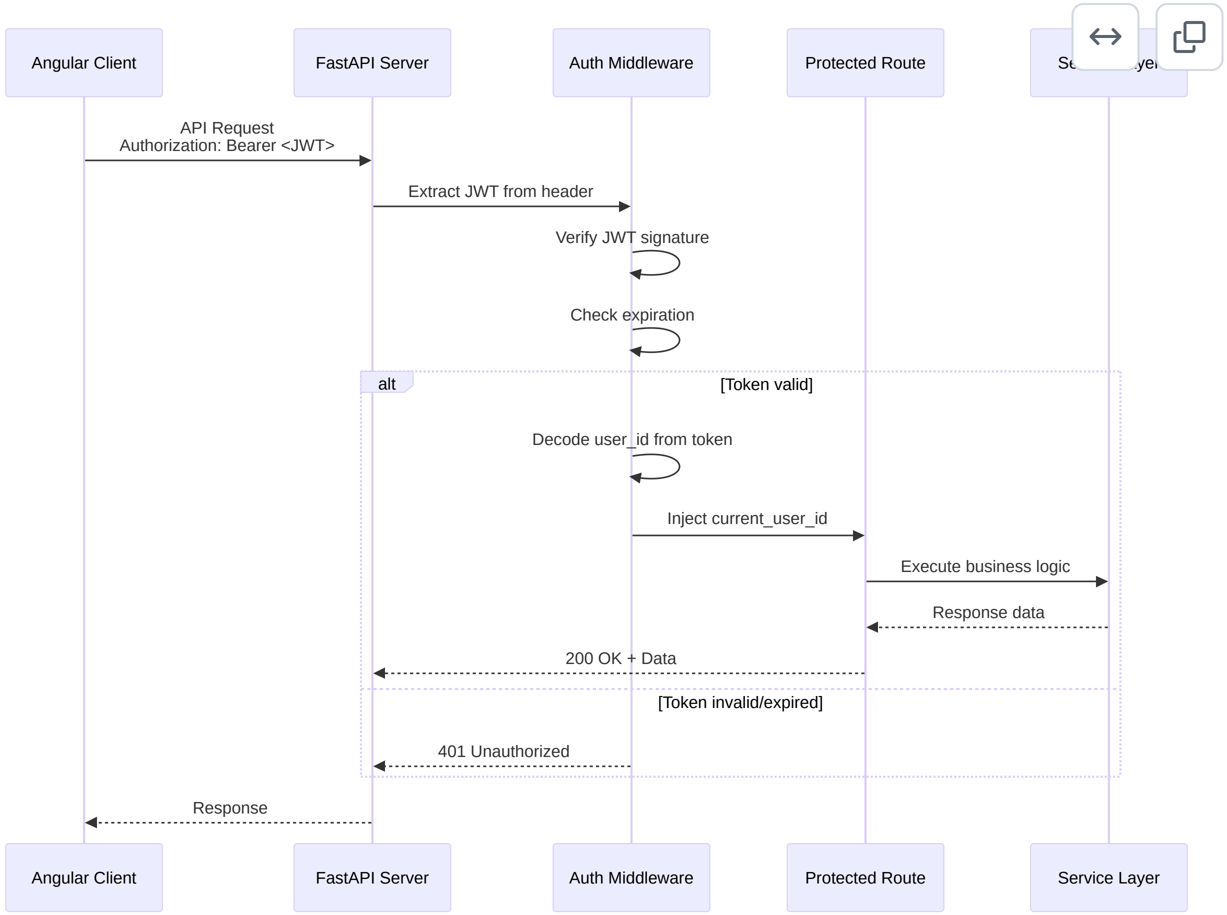

Route Handler
get_db() Dependency
SQLAlchemy Session
MySQL Database
Request database session
Create new session from pool
db.rollback() - Clean state
Yield session
Route performs operations
Query/Insert/Update
Execute SQL
Results
Data
alt
[Success]
db.commit()
COMMIT
[Exception]
db.rollback()
ROLLBACK
db.rollback() - Final cleanup
db.close()
Connection returned to pool
Route Handler
get_db() Dependency
SQLAlchemy Session
MySQL Database

5. Authentication & Authorization Flow

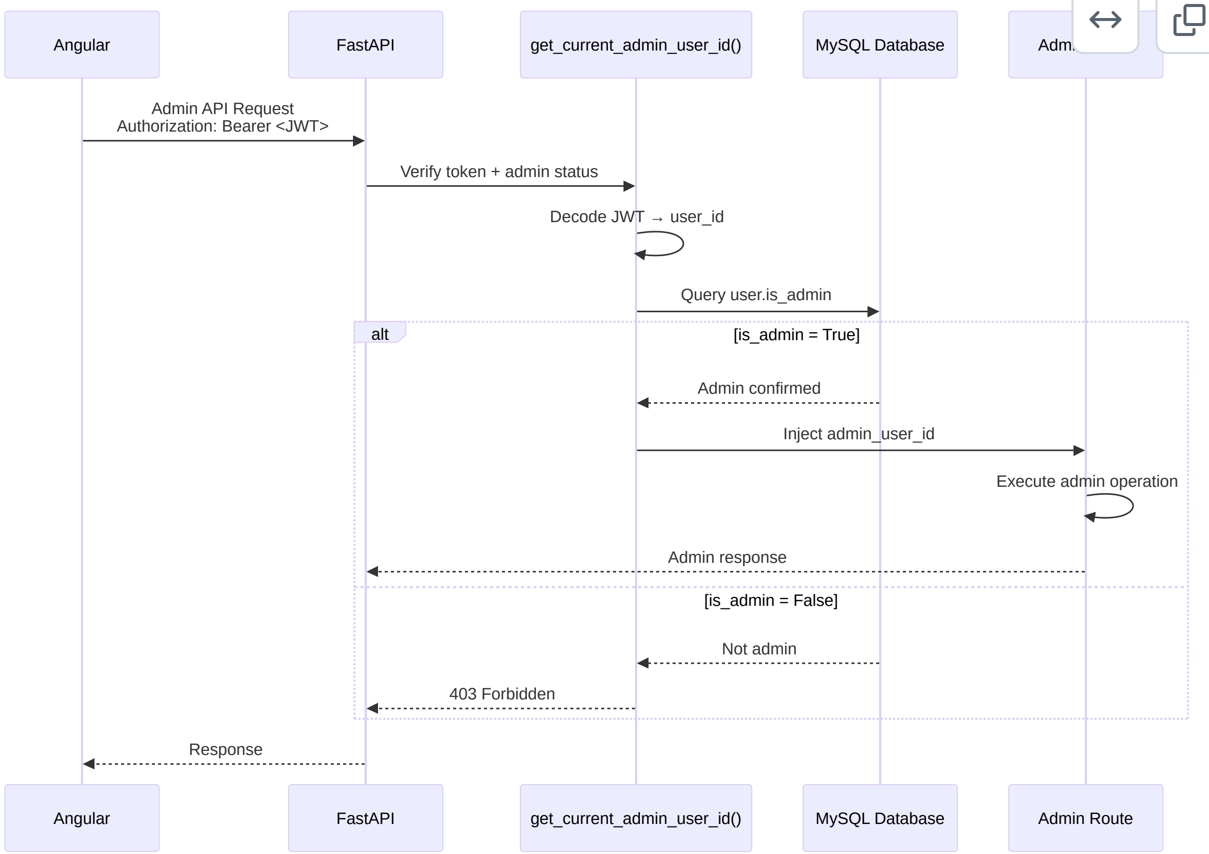
User Registration & Login



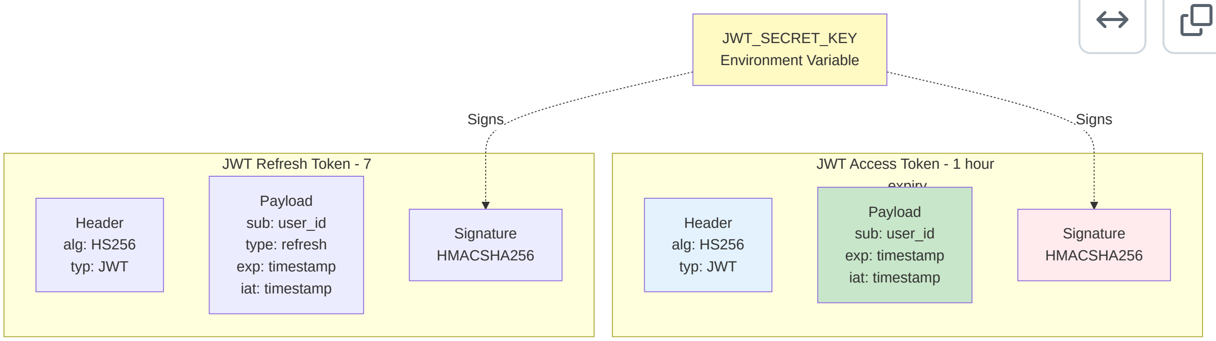
Authenticated API Request



Admin Authorization

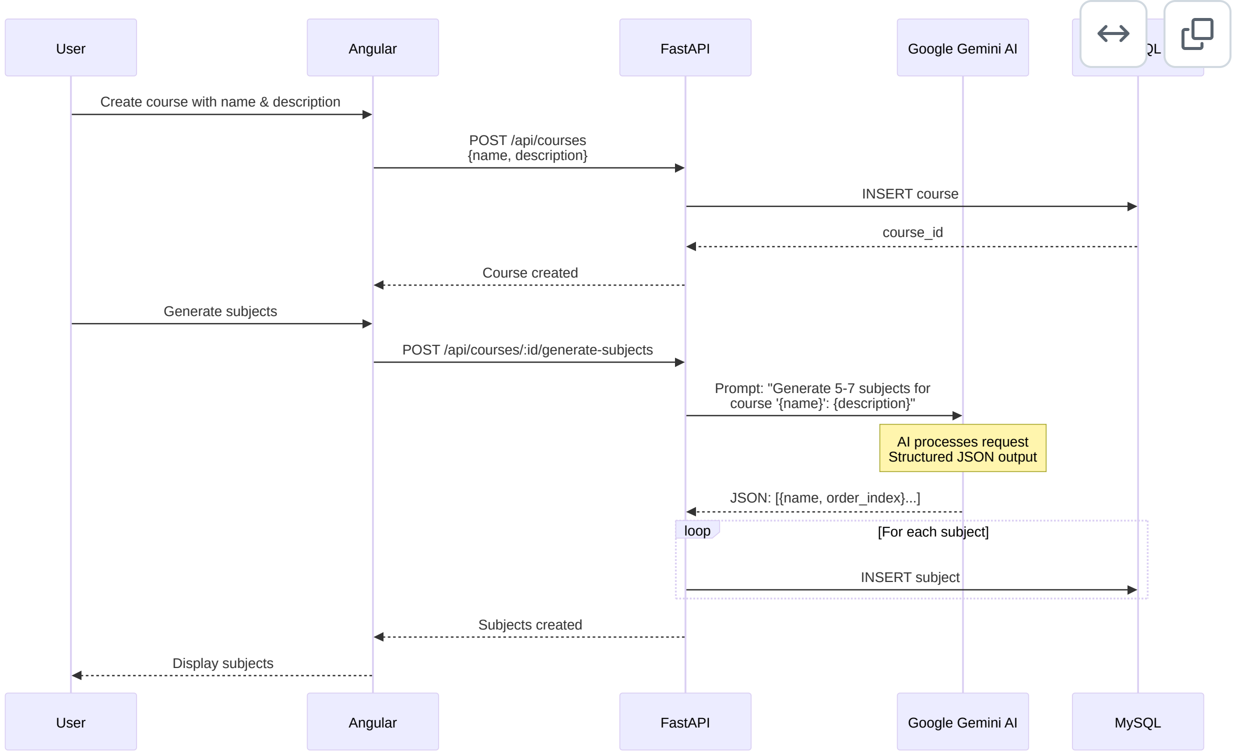


JWT Token Structure

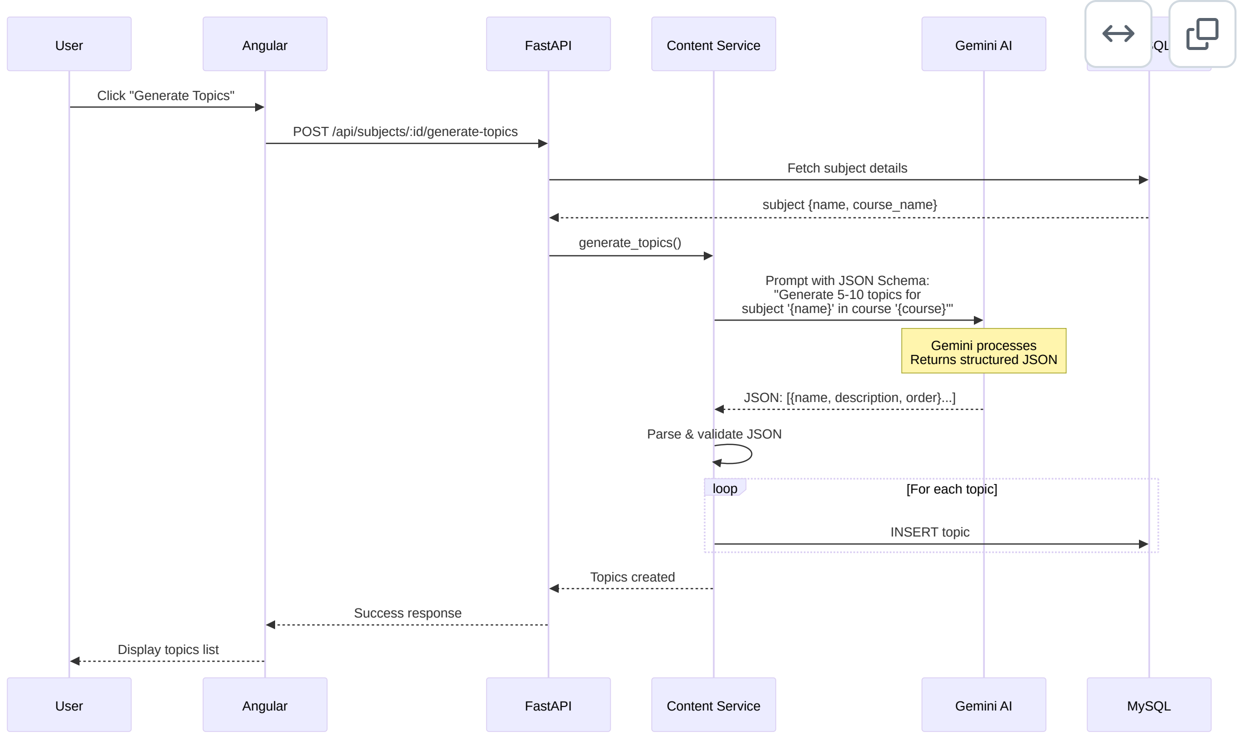


6. AI Content Generation Flow

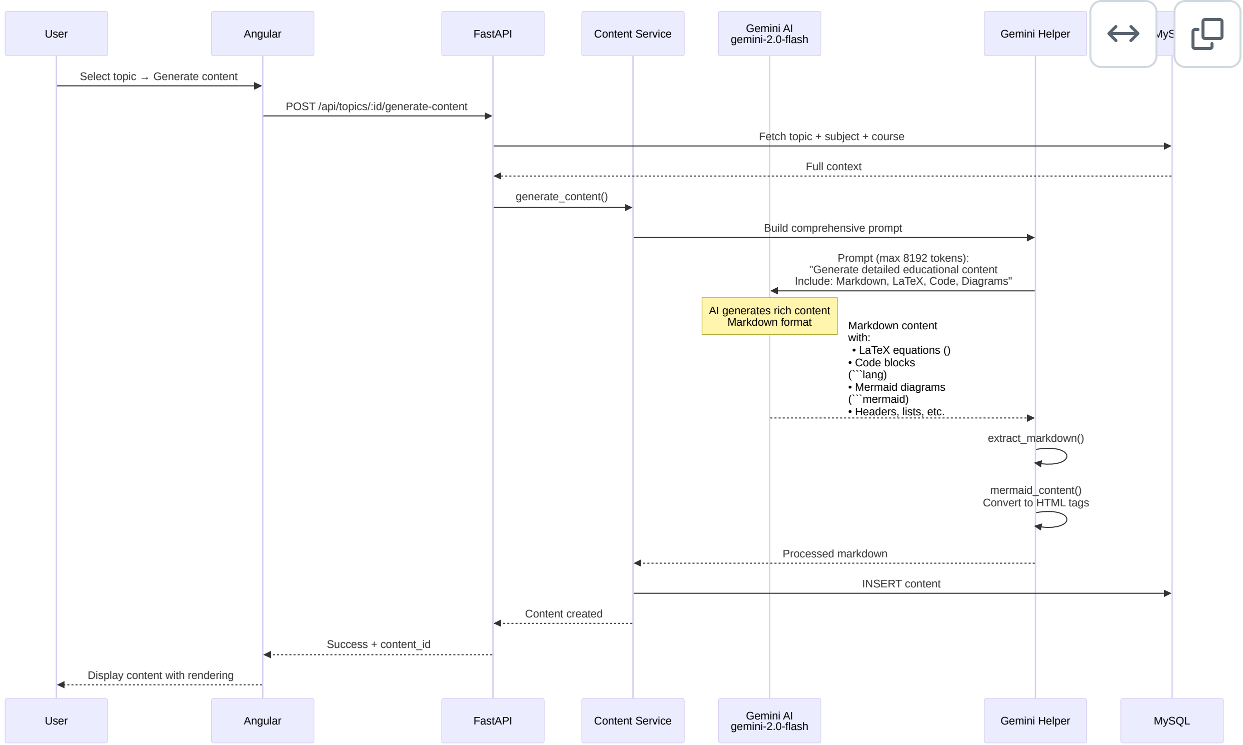
Course Subject Generation



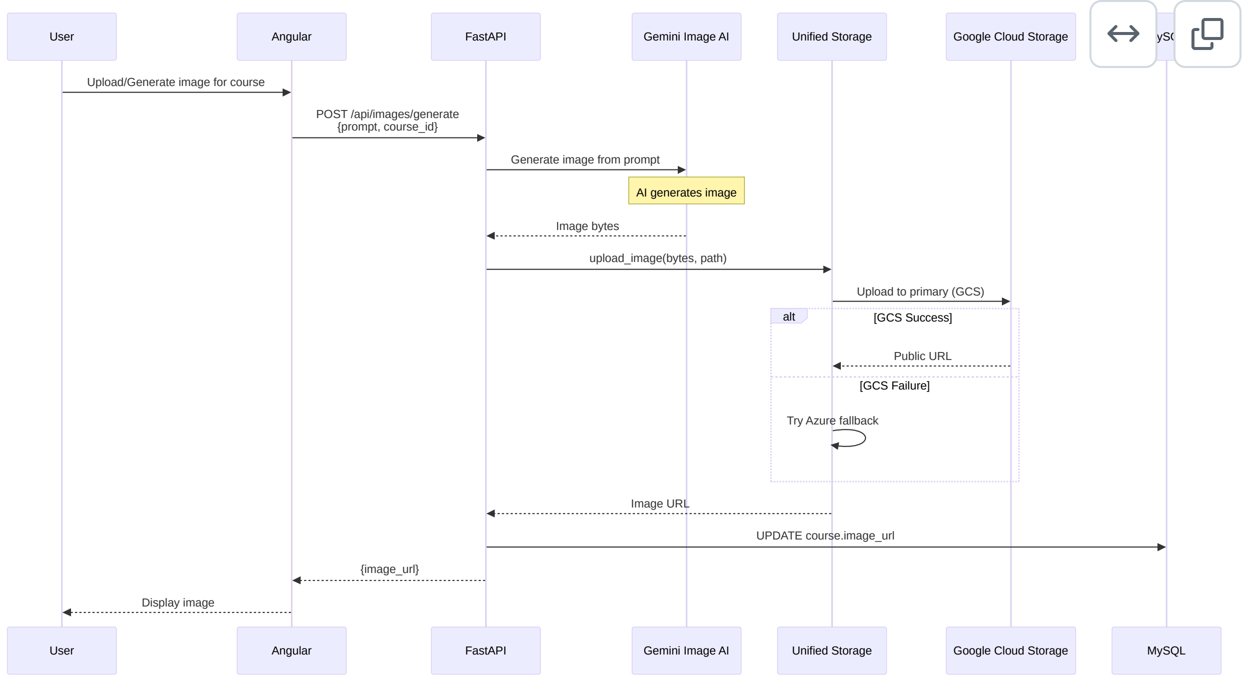
Topic Generation for Subject



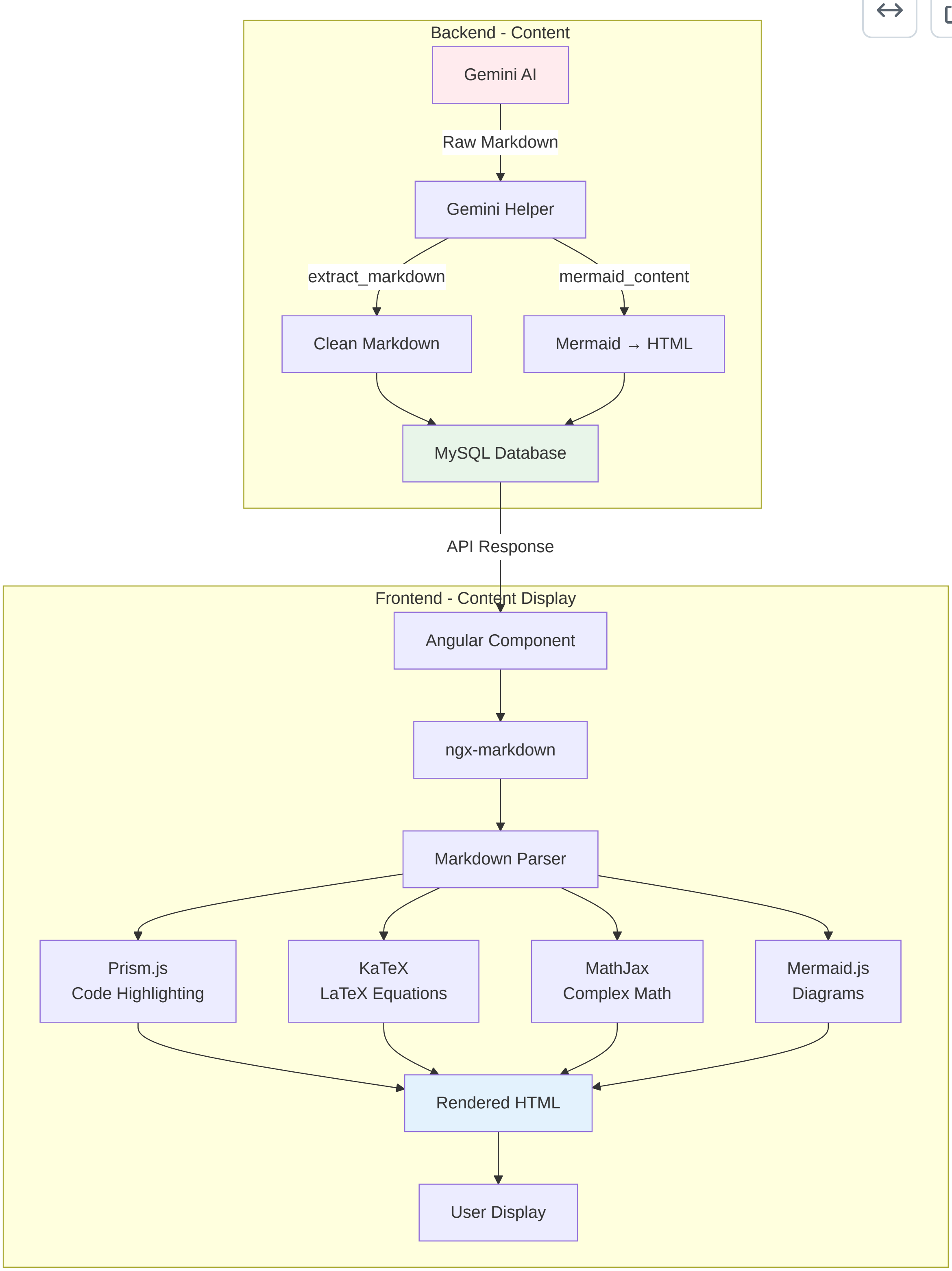
Detailed Content Generation



AI Image Generation (Optional)

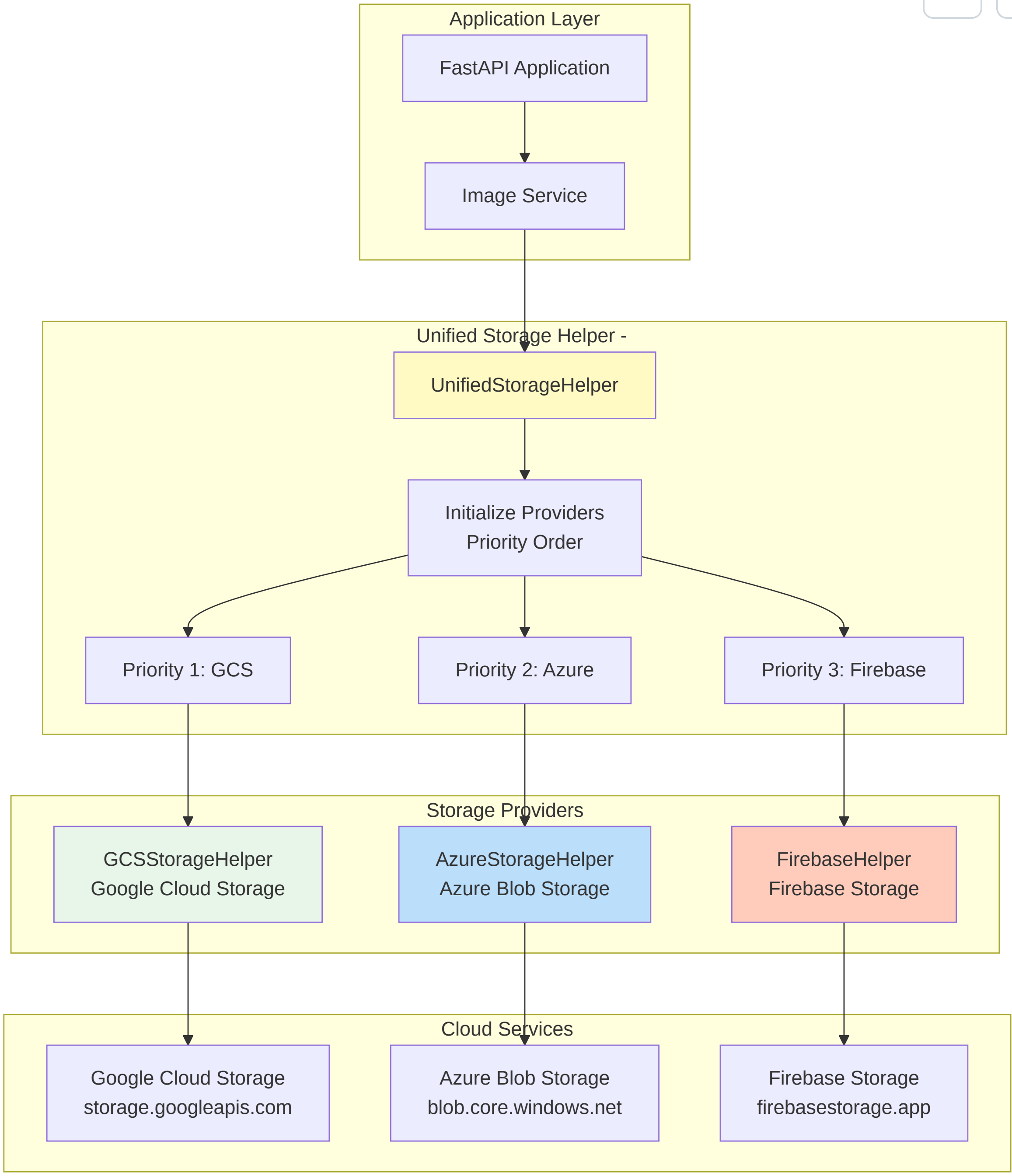


Content Rendering Pipeline

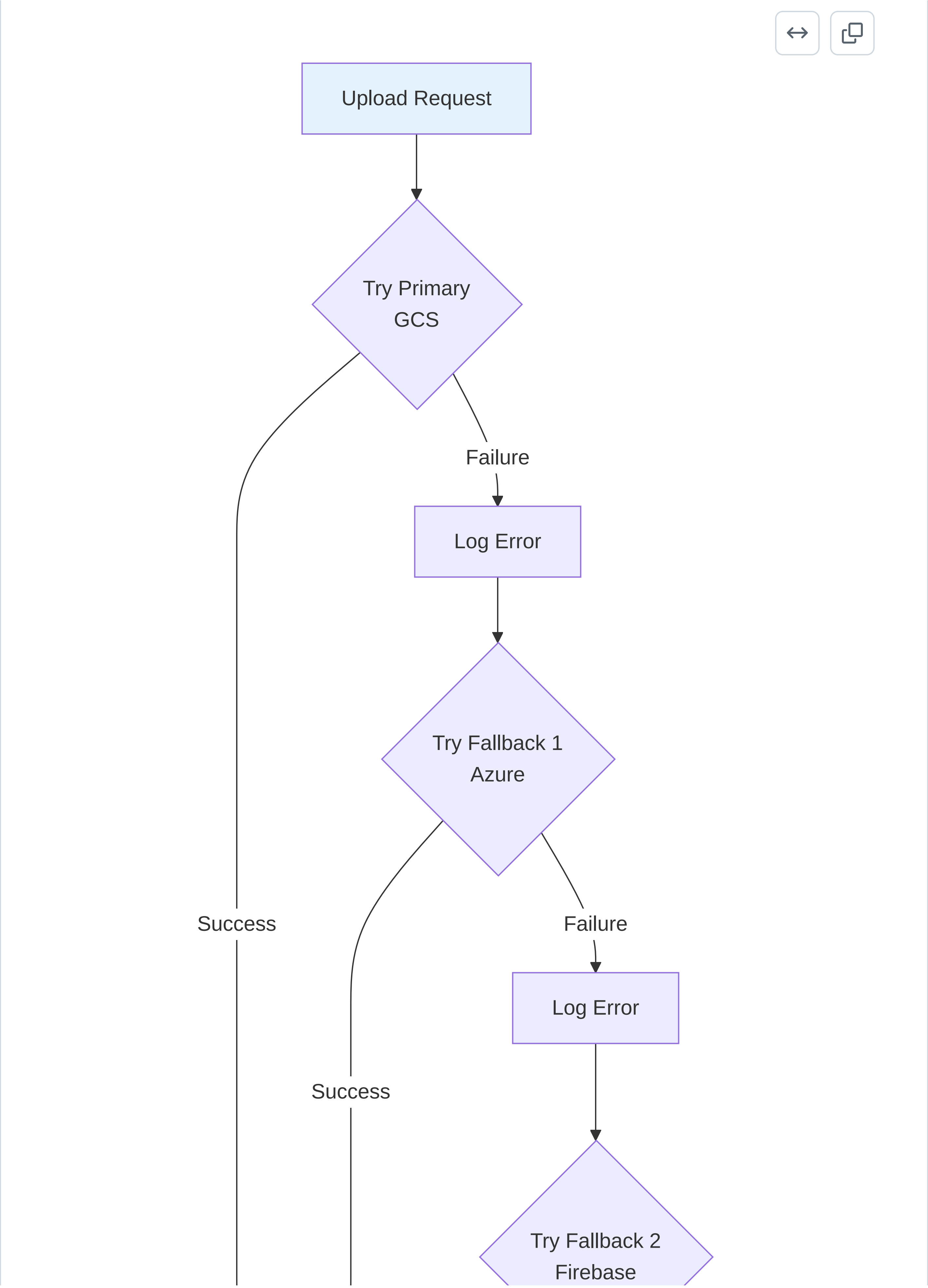

Backend - Content
Gemini AI
Raw Markdown
Gemini Helper
extract_markdown
mermaid_content
Clean Markdown
Mermaid → HTML
MySQL Database
API Response
Frontend - Content Display
Angular Component
ngx-markdown
Markdown Parser
Prism.js
Code Highlighting
KaTeX
LaTeX Equations
MathJax
Complex Math
Mermaid.js
Diagrams
Rendered HTML
User Display

7. Multi-Cloud Storage Architecture

Unified Storage Abstraction


Application Layer
FastAPI Application
Image Service
Unified Storage Helper -
UnifiedStorageHelper
Initialize Providers
Priority Order
Priority 1: GCS
Priority 2: Azure
Priority 3: Firebase
Storage Providers
GCSStorageHelper
Google Cloud Storage
AzureStorageHelper
Azure Blob Storage
FirebaseHelper
Firebase Storage
Cloud Services
Google Cloud Storage
storage.googleapis.com
Azure Blob Storage
blob.core.windows.net
Firebase Storage
firebasestorage.app

Storage Failover Flow


Upload Request
Try Primary GCS
Failure
Success
Log Error
Try Fallback 1 Azure
Failure
Success
Log Error
Try Fallback 2 Firebase

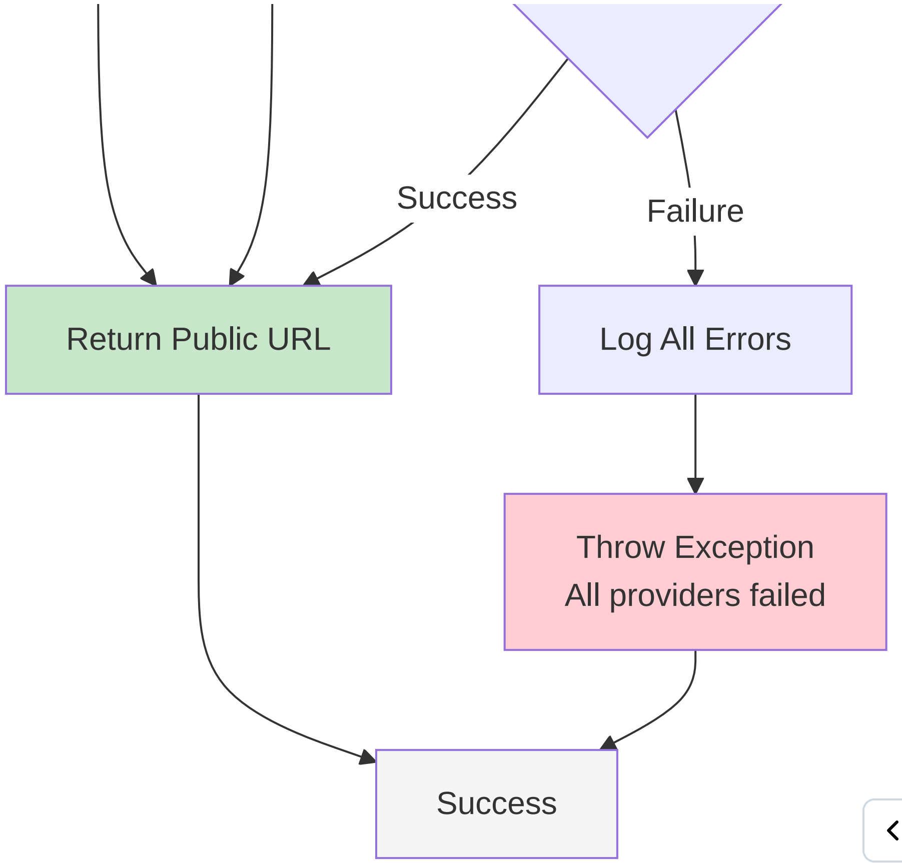
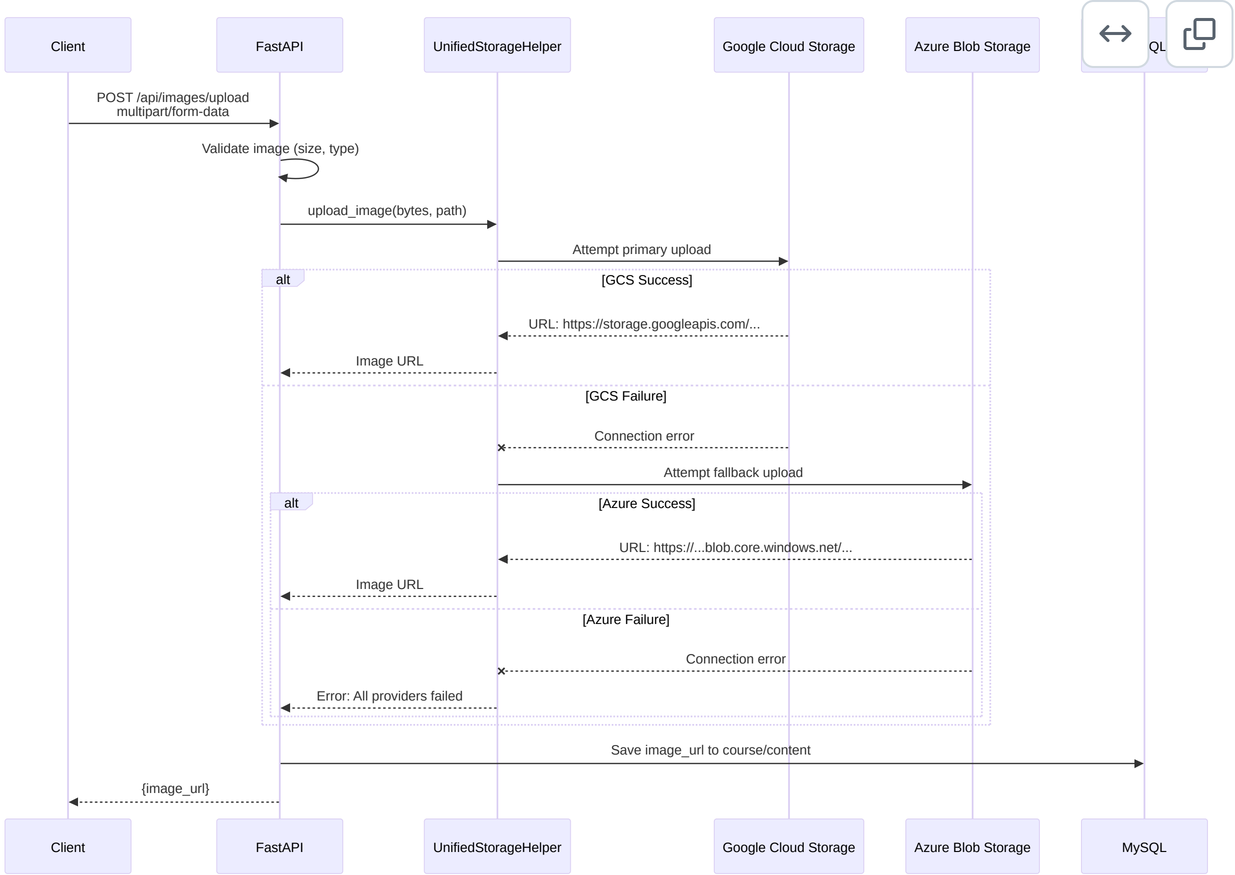
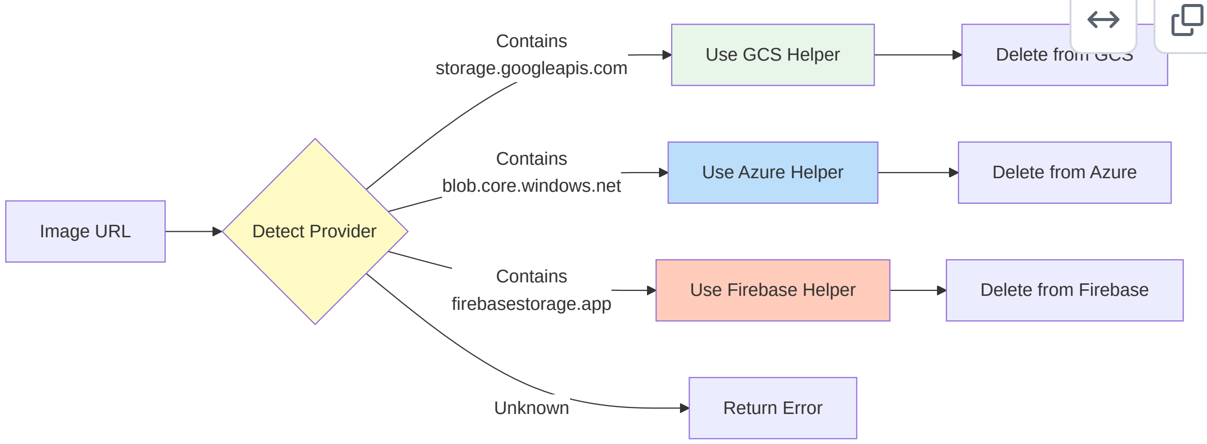


Image Upload Sequence

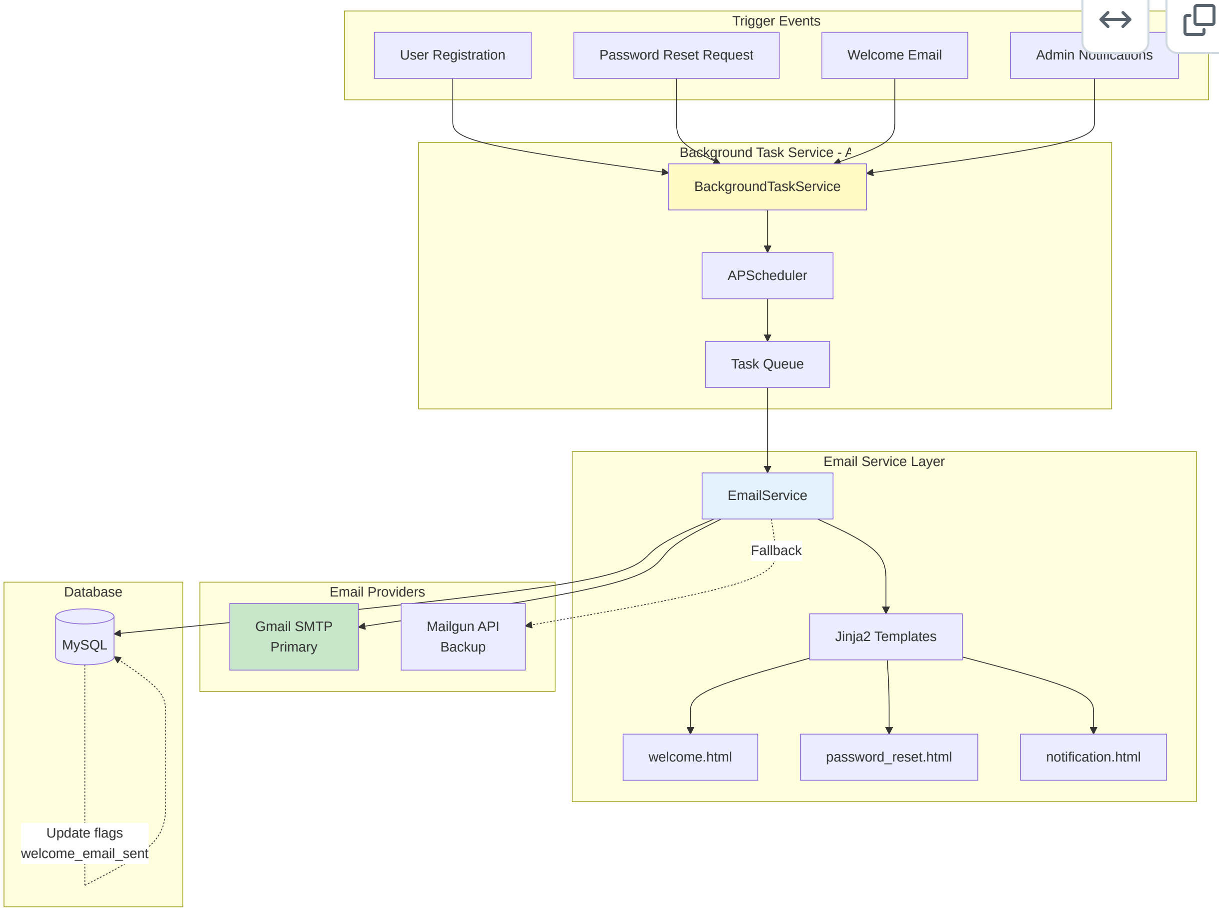


Storage Provider Detection

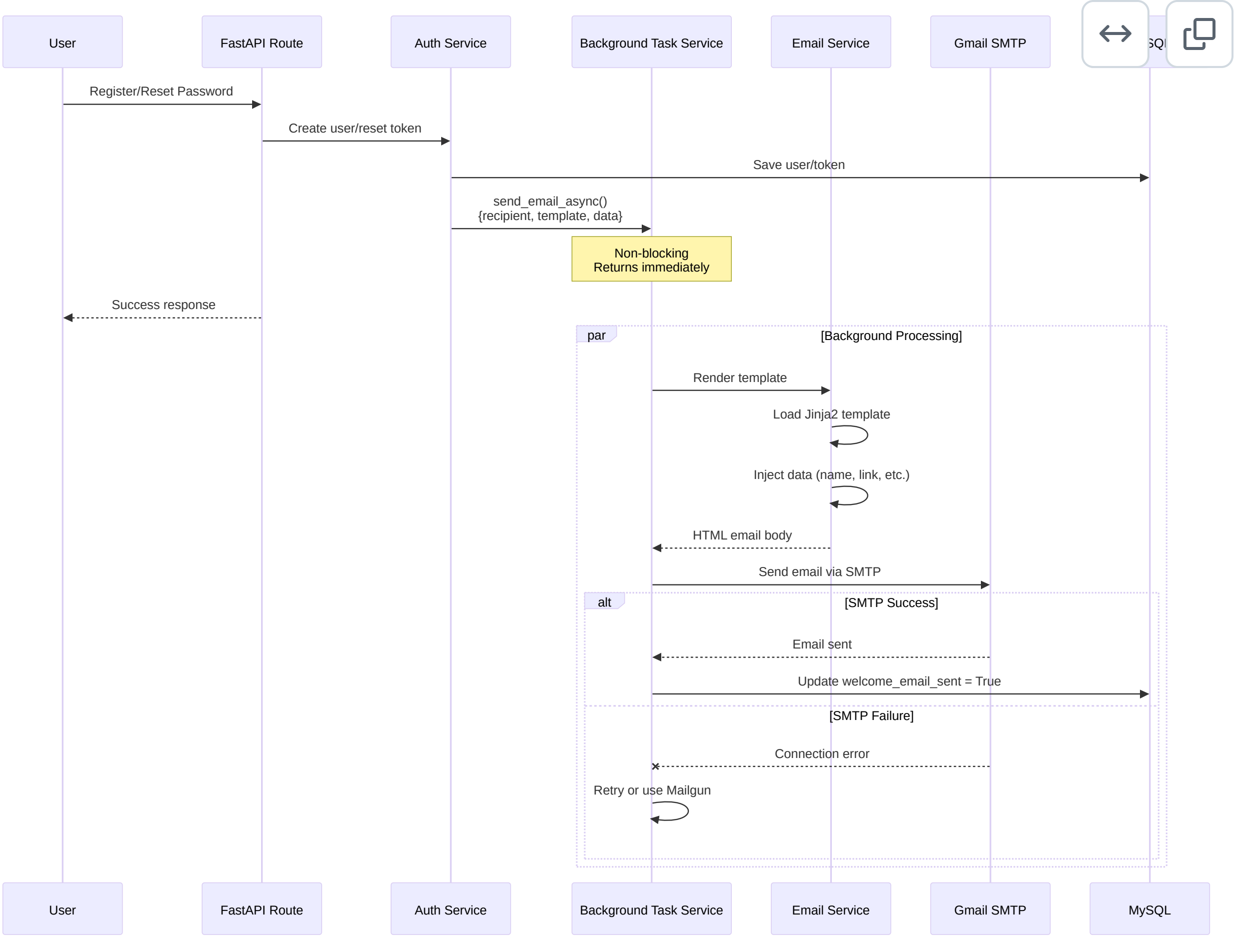


8. Email Notification System

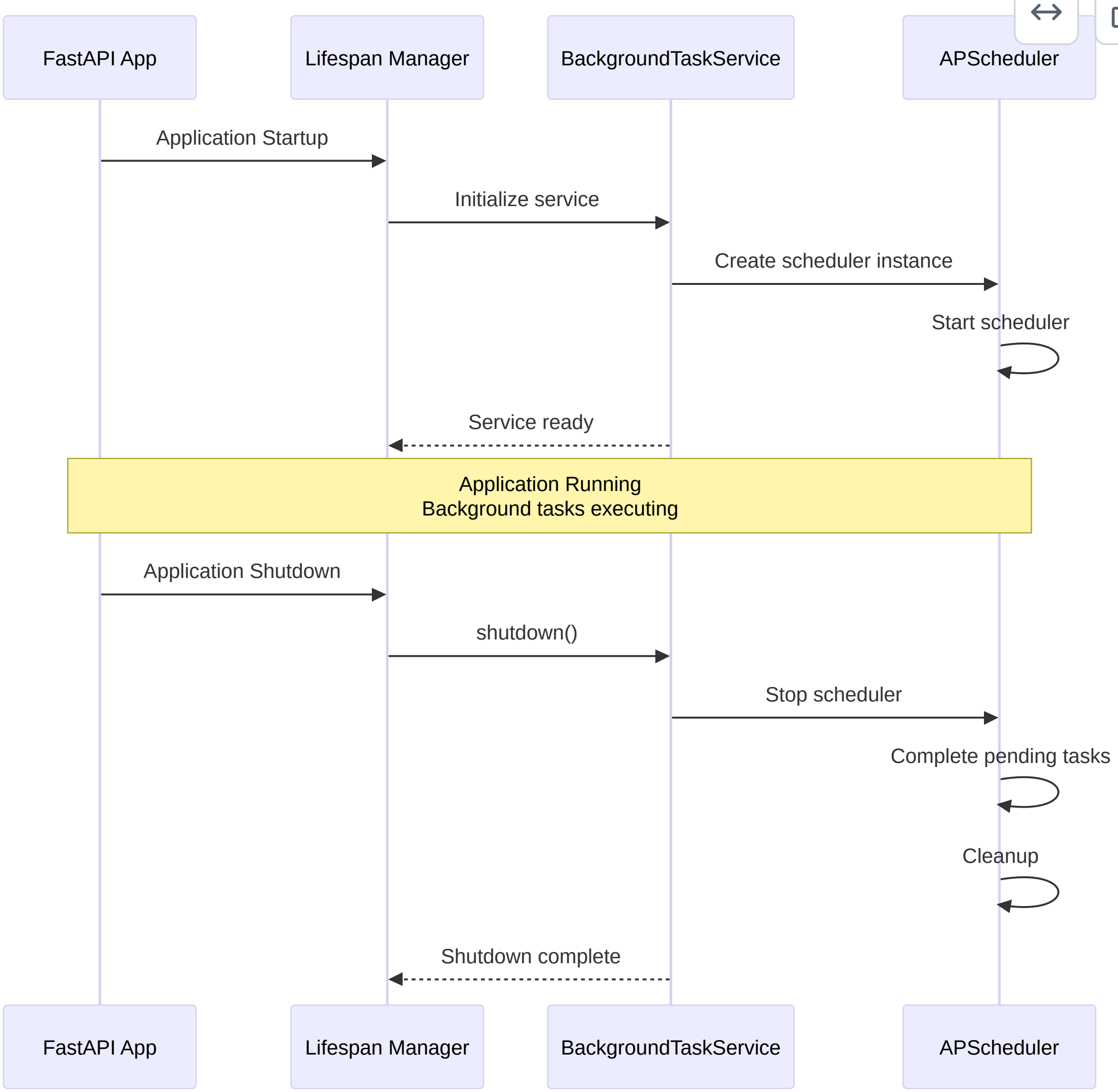
Email System Architecture



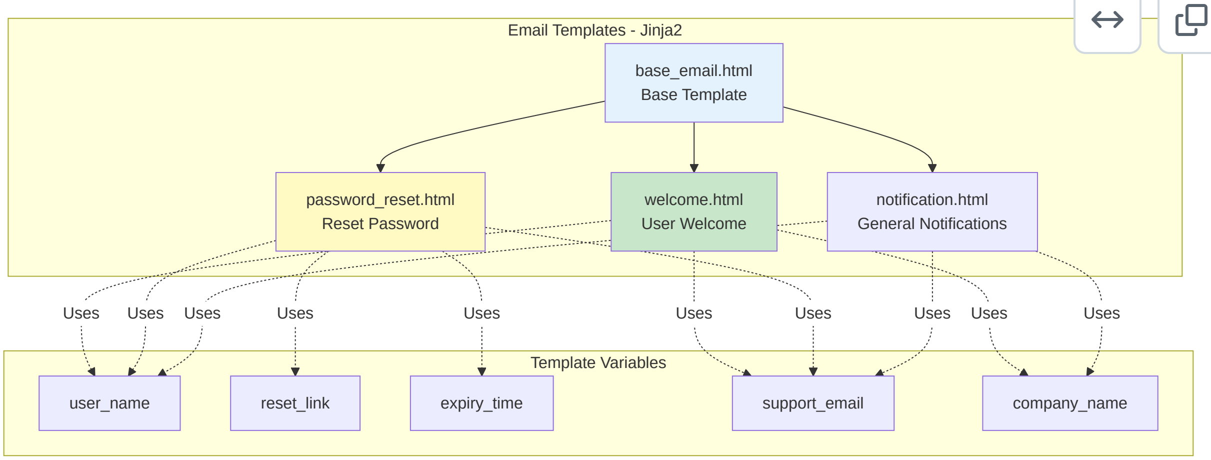
Email Sending Sequence



APScheduler Lifecycle

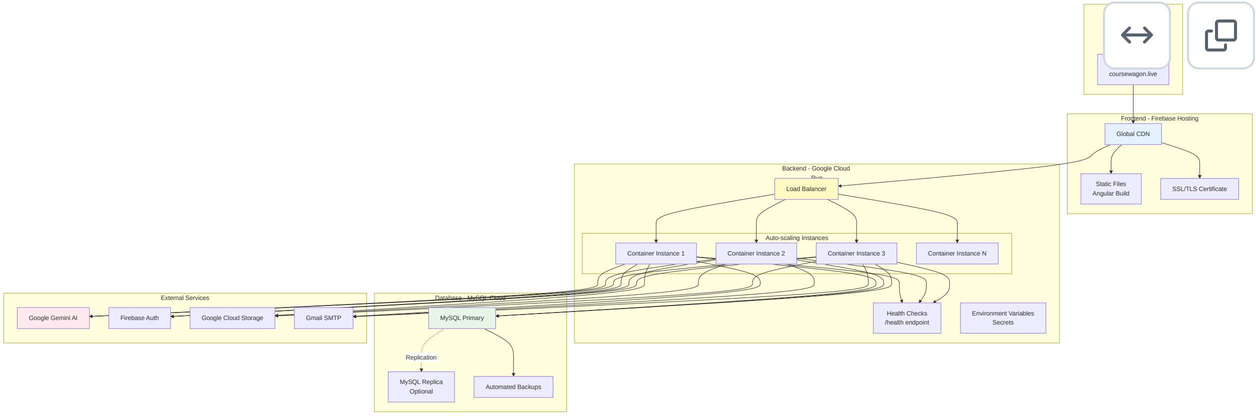


Email Templates Structure

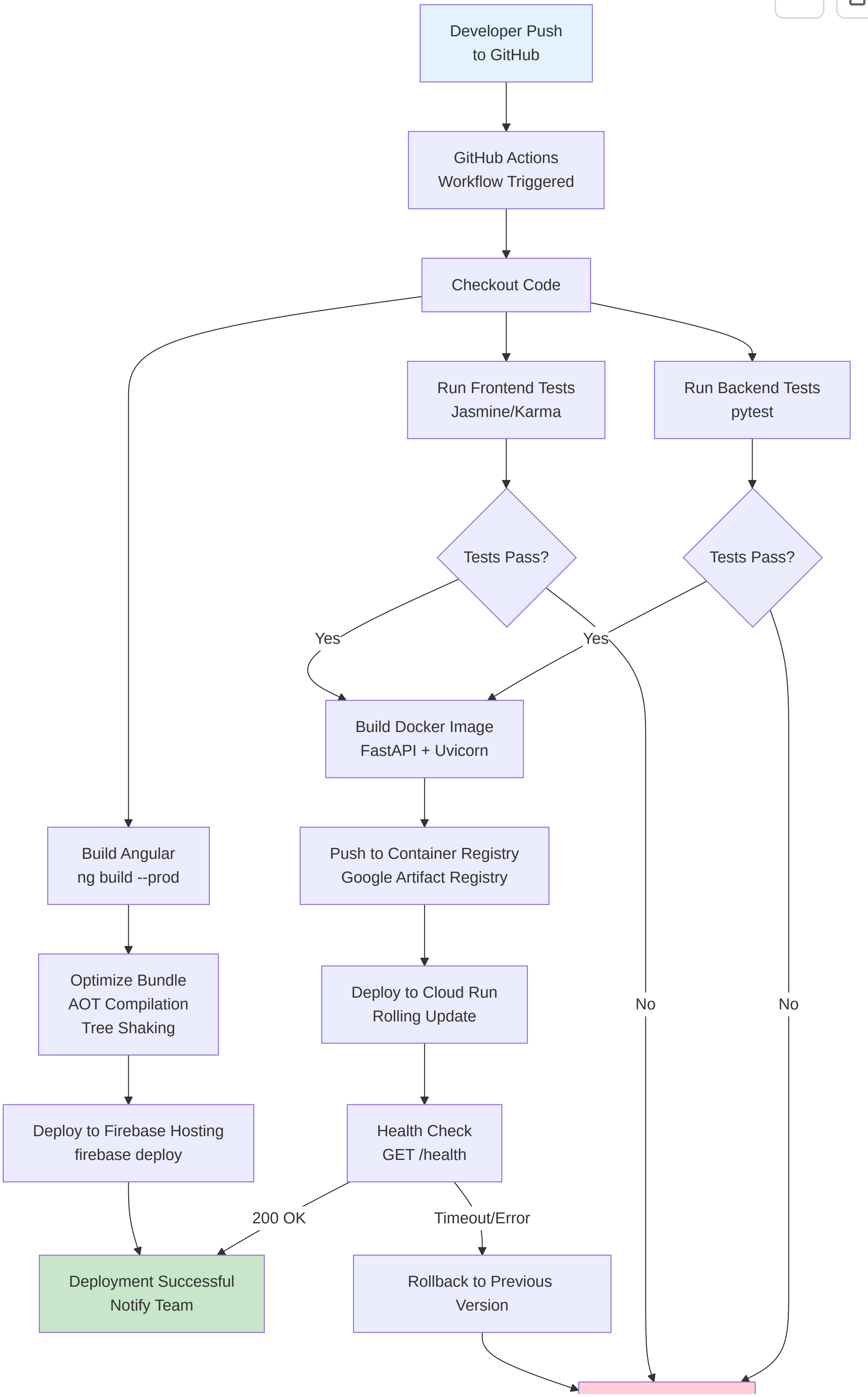


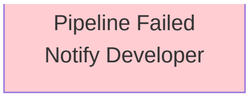
9. Deployment Architecture

Production Infrastructure

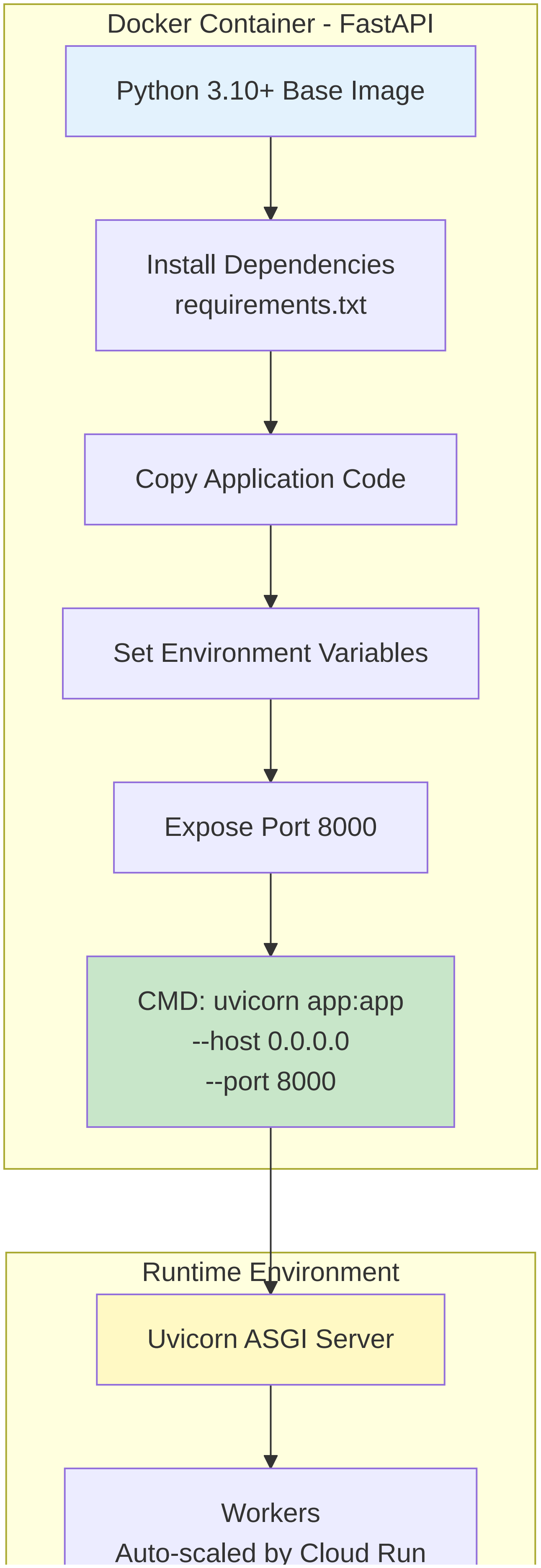


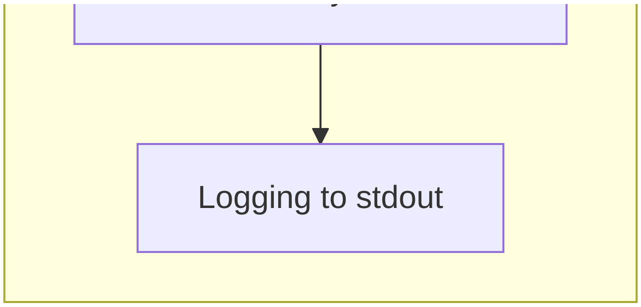
CI/CD Pipeline


Developer Push
to GitHub
GitHub Actions
Workflow Triggered
Checkout Code
Run Frontend Tests
Jasmine/Karma
Run Backend Tests
pytest
Tests Pass?
Tests Pass?
Yes
Yes
Build Docker Image
FastAPI + Uvicorn
Build Angular
ng build --prod
Push to Container Registry
Google Artifact Registry
Optimize Bundle
AOT Compilation
Tree Shaking
Deploy to Cloud Run
Rolling Update
No
No
Deploy to Firebase Hosting
firebase deploy
Health Check
GET /health
200 OK
Timeout/Error
Deployment Successful
Notify Team
Rollback to Previous
Version

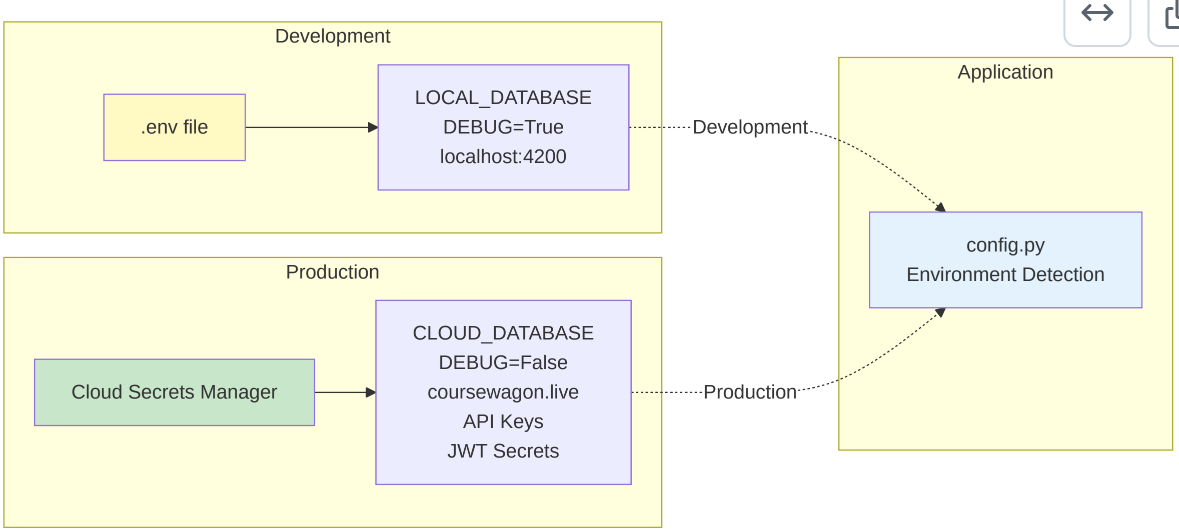


Container Architecture


Docker Container - FastAPI
Python 3.10+ Base Image
Install Dependencies
requirements.txt
Copy Application Code
Set Environment Variables
Expose Port 8000
CMD: uvicorn app:app
--host 0.0.0.0
--port 8000
Runtime Environment
Uvicorn ASGI Server
Workers
Auto-scaled by Cloud Run

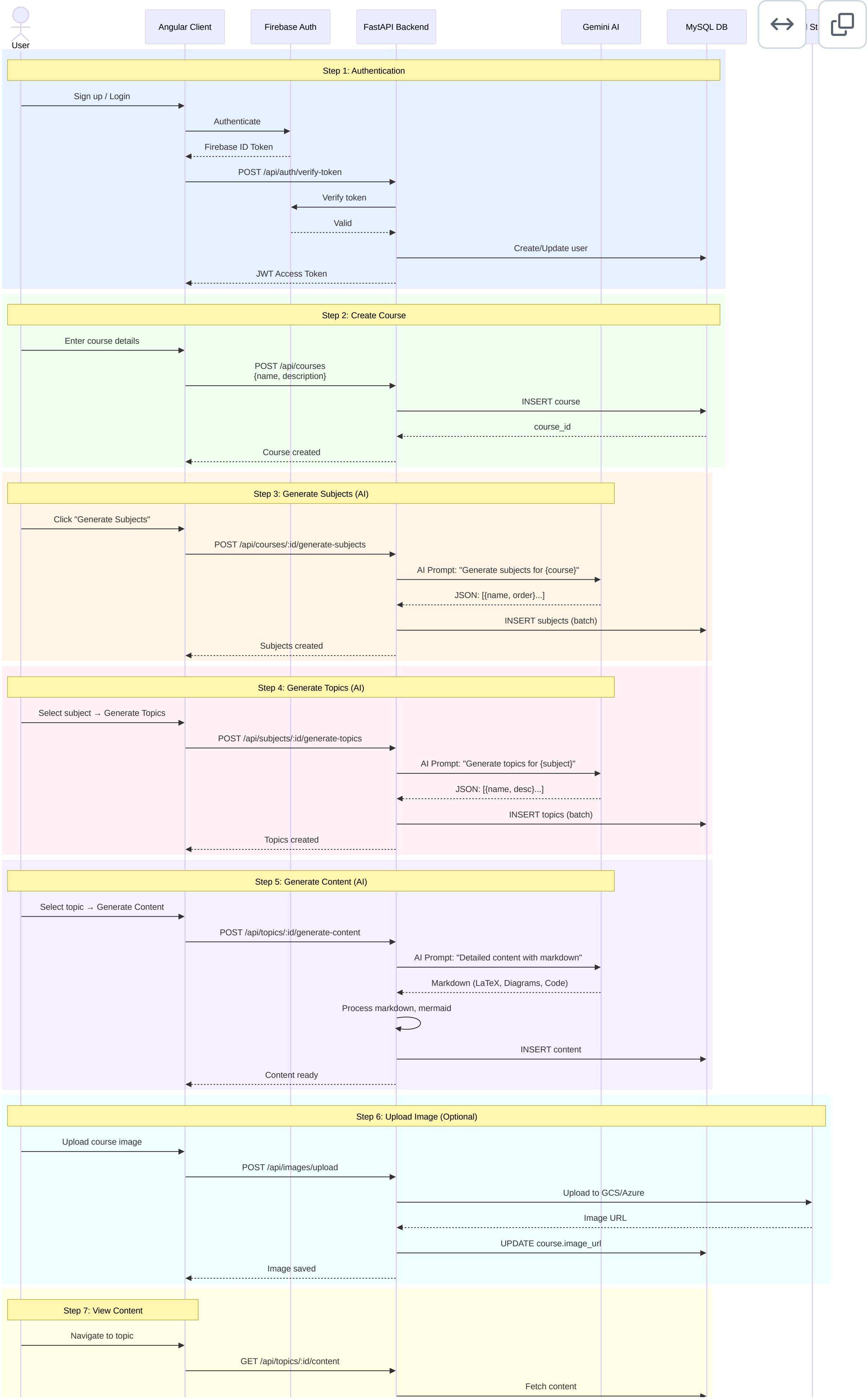


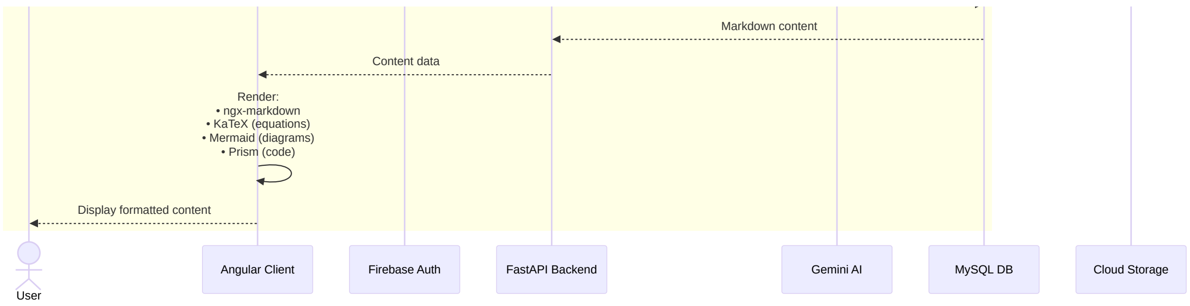
Environment Configuration



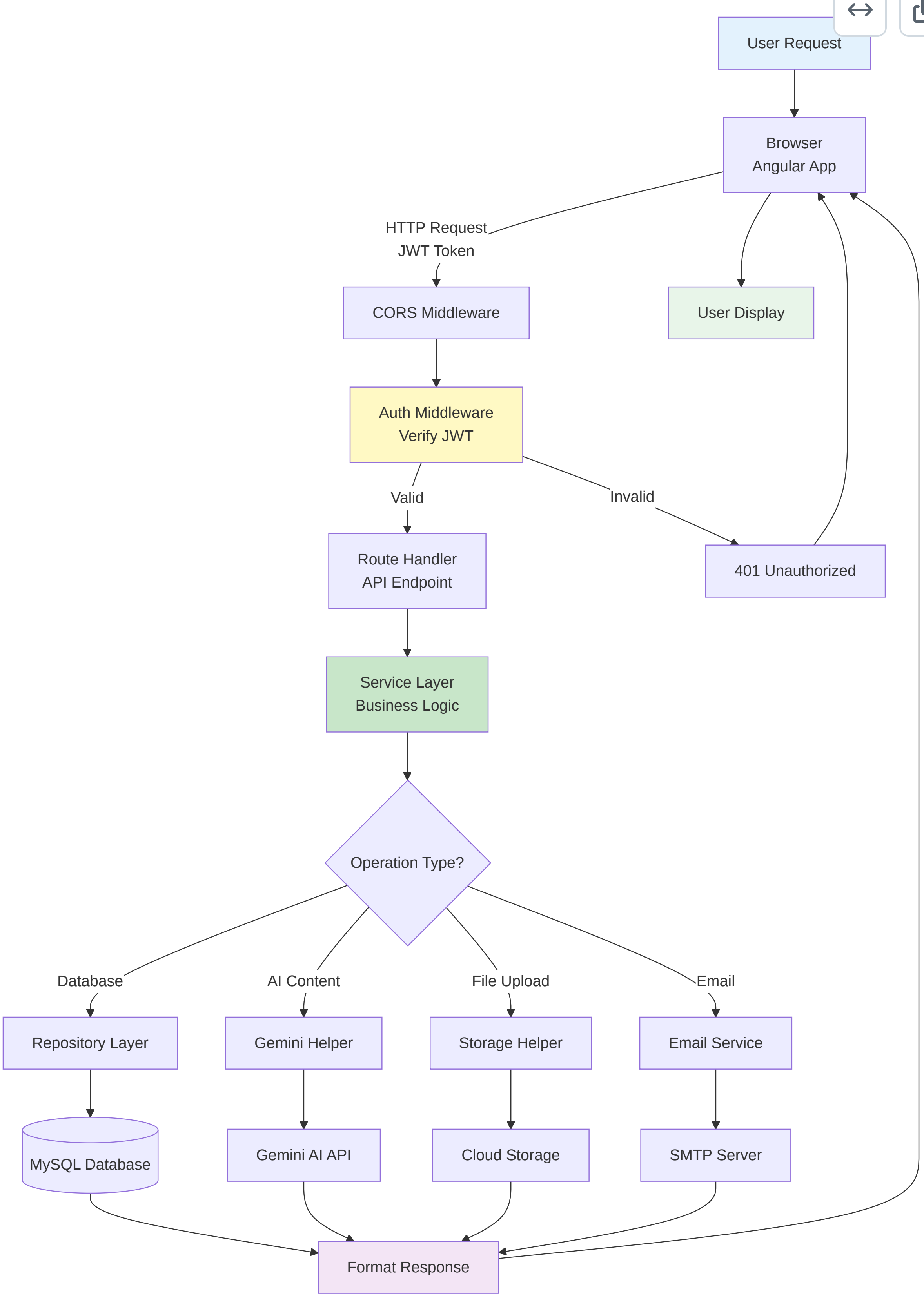
10. Complete Data Flow (End-to-End)

User Journey: Create and View Course Content


User
Angular Client
Firebase Auth
FastAPI Backend
Gemini AI
MySQL DB
l St
Step 1: Authentication
Sign up / Login
Authenticate
Firebase ID Token
POST /api/auth/verify-token
Verify token
Valid
Create/Update user
JWT Access Token
Step 2: Create Course
Enter course details
POST /api/courses
{name, description}
INSERT course
course_id
Course created
Step 3: Generate Subjects (AI)
Click "Generate Subjects"
POST /api/courses/:id/generate-subjects
AI Prompt: "Generate subjects for {course}"
JSON: [{name, order}...]
INSERT subjects (batch)
Subjects created
Step 4: Generate Topics (AI)
Select subject → Generate Topics
POST /api/subjects/:id/generate-topics
AI Prompt: "Generate topics for {subject}"
JSON: [{name, desc}...]
INSERT topics (batch)
Topics created
Step 5: Generate Content (AI)
Select topic → Generate Content
POST /api/topics/:id/generate-content
AI Prompt: "Detailed content with markdown"
Markdown (LaTeX, Diagrams, Code)
Process markdown, mermaid
INSERT content
Content ready
Step 6: Upload Image (Optional)
Upload course image
POST /api/images/upload
Upload to GCS/Azure
Image URL
UPDATE course.image_url
Image saved
Step 7: View Content
Navigate to topic
GET /api/topics/:id/content
Fetch content

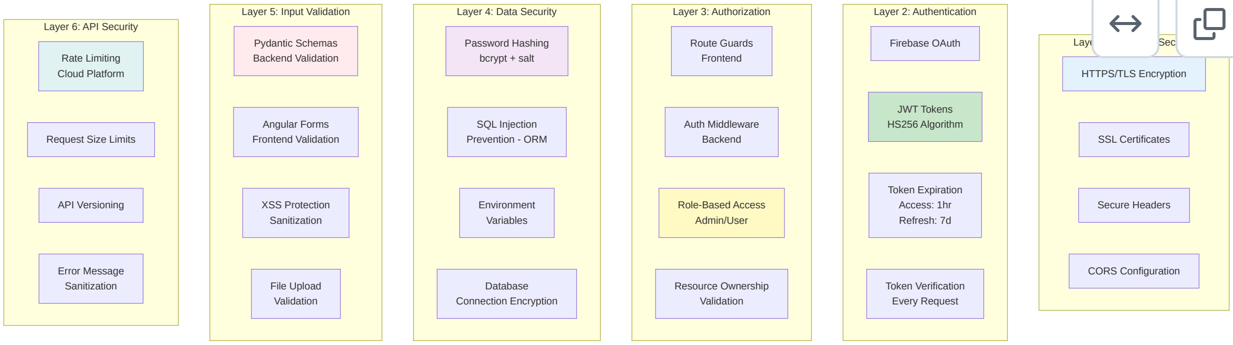


Request Flow Through System Layers

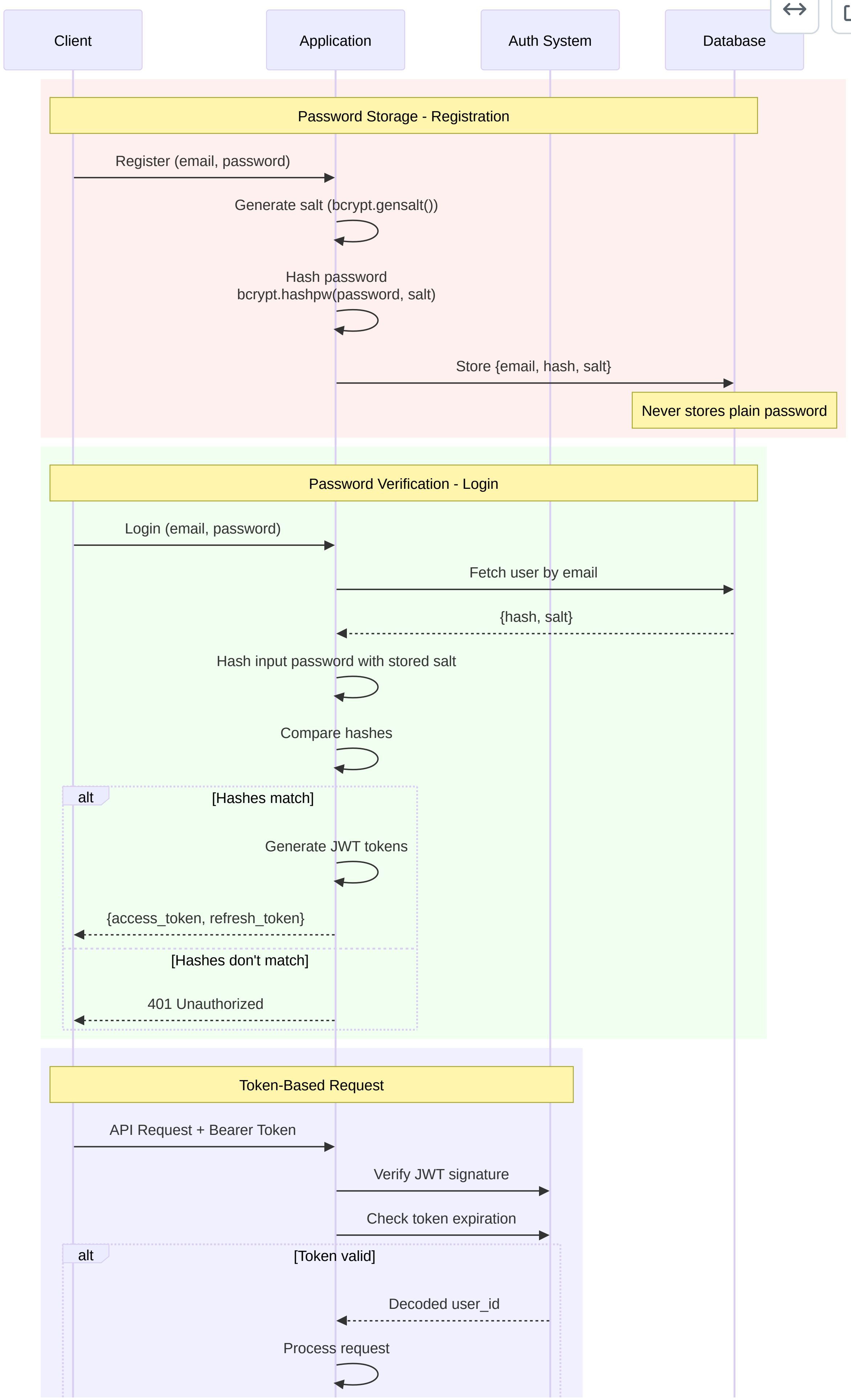

User Request
Browser
Angular App
HTTP Request
JWT Token
CORS Middleware
User Display
Auth Middleware
Verify JWT
Valid
Invalid
Route Handler
API Endpoint
401 Unauthorized
Service Layer
Business Logic
Operation Type?
Database
AI Content
File Upload
Email
Repository Layer
Gemini Helper
Storage Helper
Email Service
MySQL Database
Gemini AI API
Cloud Storage
SMTP Server
Format Response

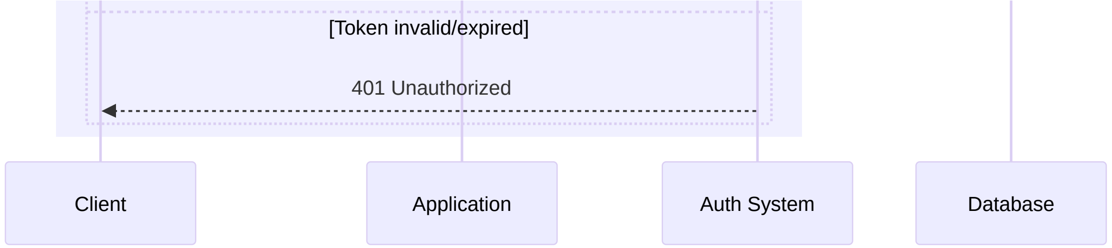
11. Security Architecture

Security Layers



Authentication Security Flow


Client
Application
Auth System
Database
Password Storage - Registration
Register (email, password)
Generate salt (bcrypt.gensalt())
Hash password
bcrypt.hashpw(password, salt)
Store {email, hash, salt}
Never stores plain password
Password Verification - Login
Login (email, password)
Fetch user by email
{hash, salt}
Hash input password with stored salt
Compare hashes
alt
[Hashes match]
Generate JWT tokens
{access_token, refresh_token}
[Hashes don't match]
401 Unauthorized
Token-Based Request
API Request + Bearer Token
Verify JWT signature
Check token expiration
alt
[Token valid]
Decoded user_id
Process request



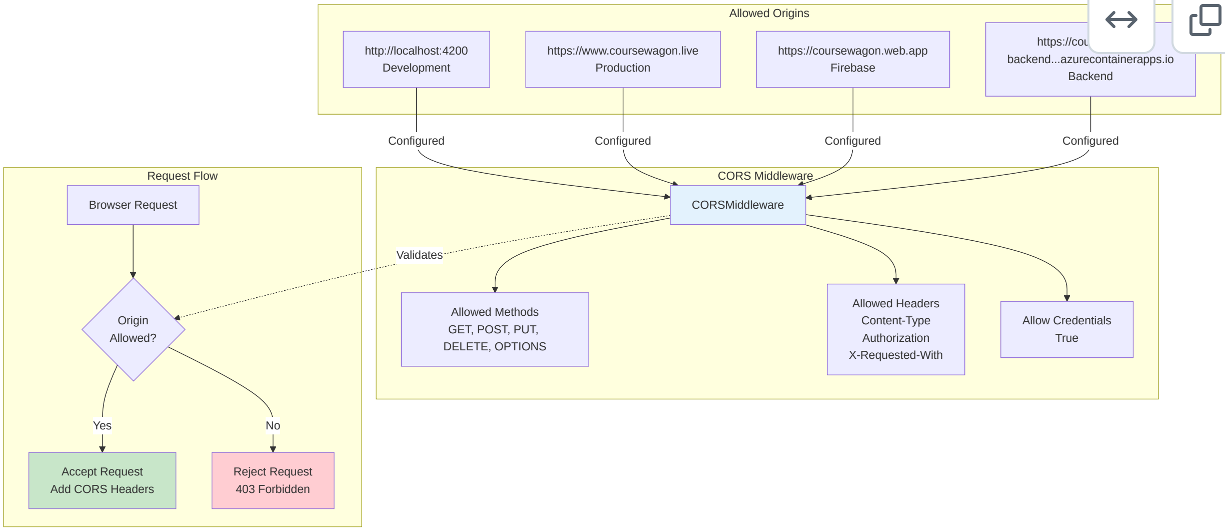
SQL Injection Prevention

Unable to render rich display

```
Parse error on line 3:
...PUT[User Input<br/>"'; DROP TABLE users;...
----------------------^
Expecting 'SQE', 'DOUBLECIRCLEEND', 'PE', '-)', 'STADIUMEND',
'SUBROUTINEEND', 'PIPE', 'CYLINDEREND', 'DIAMOND_STOP', 'TAGEND',
'TRAPEND', 'INVTRAPEND', 'UNICODE_TEXT', 'TEXT', 'TAGSTART', got 'STR'

For more information, see https://docs.github.com/get-started/writing-on-github/working-with-advanced-formatting/creating-diagrams#creating-mermaid-diagrams
```

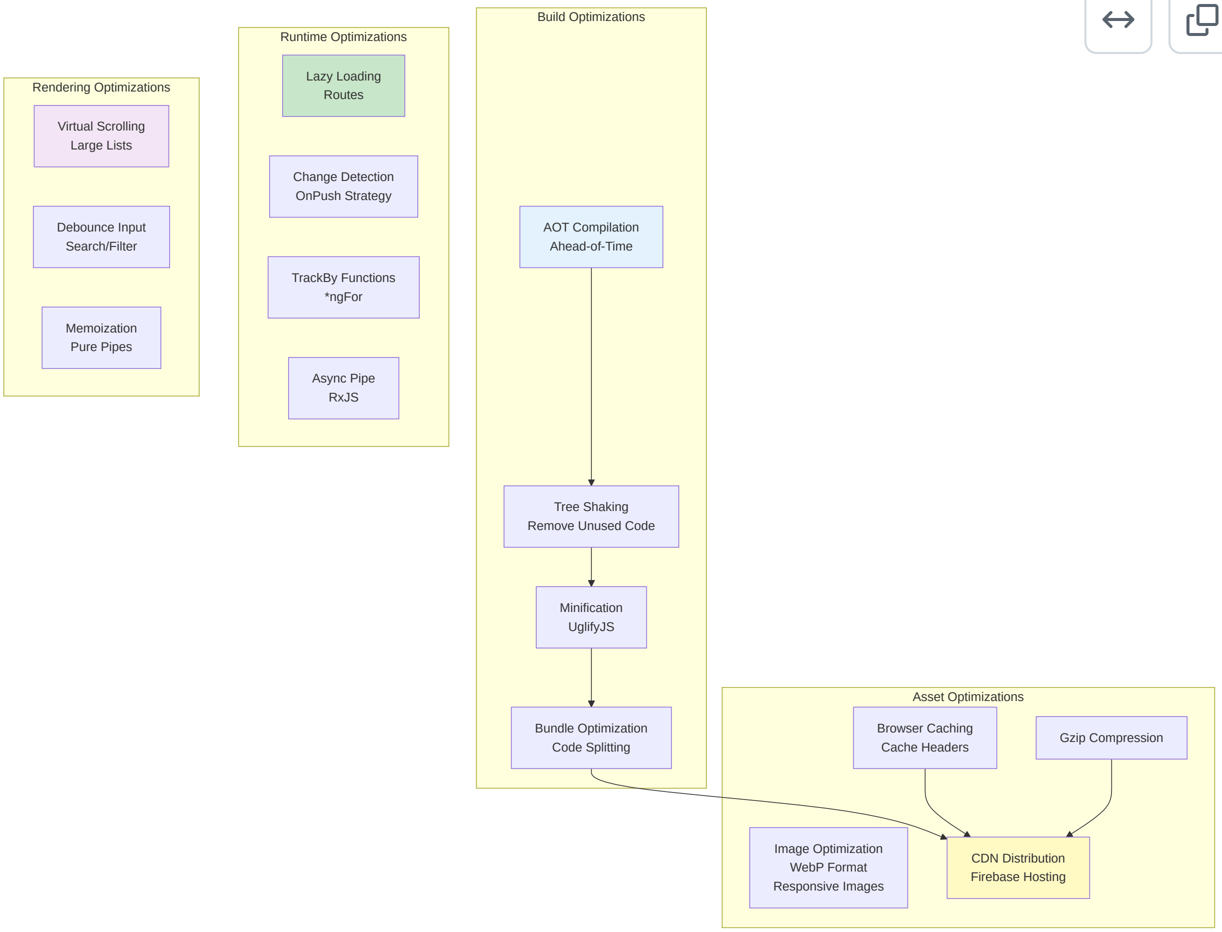
```
graph LR
    subgraph "User Input"
        INPUT[User Input<br/>"'; DROP TABLE users; --"]
    end

    subgraph "Application Layer"
        VALIDATION[Input Validation<br/>Pydantic Schema]
    end

    subgraph "ORM Layer - SQLAlchemy"
        PARAM[Parameterized Queries]
        ESCAPE[Automatic Escaping]
        SAFE[Safe Query Building]
    end

    subgraph "Database"
        DB[(MySQL)]
        SAFE_QUERY["SELECT * FROM users<br/>WHERE email = ?<br/>Parameter:
    end

    INPUT --> VALIDATION
    VALIDATION -->|Validated| PARAM
    PARAM --> ESCAPE
    ESCAPE --> SAFE
    SAFE --> DB
    DB --> SAFE_QUERY

    style VALIDATION fill:#c8e6c9
    style PARAM fill:#e3f2fd
    style SAFE_QUERY fill:#fff9c4
```

CORS Configuration

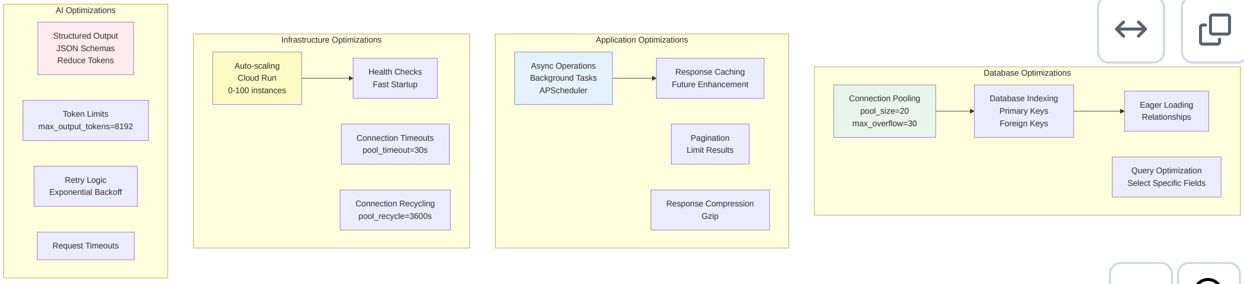


12. Performance Optimizations

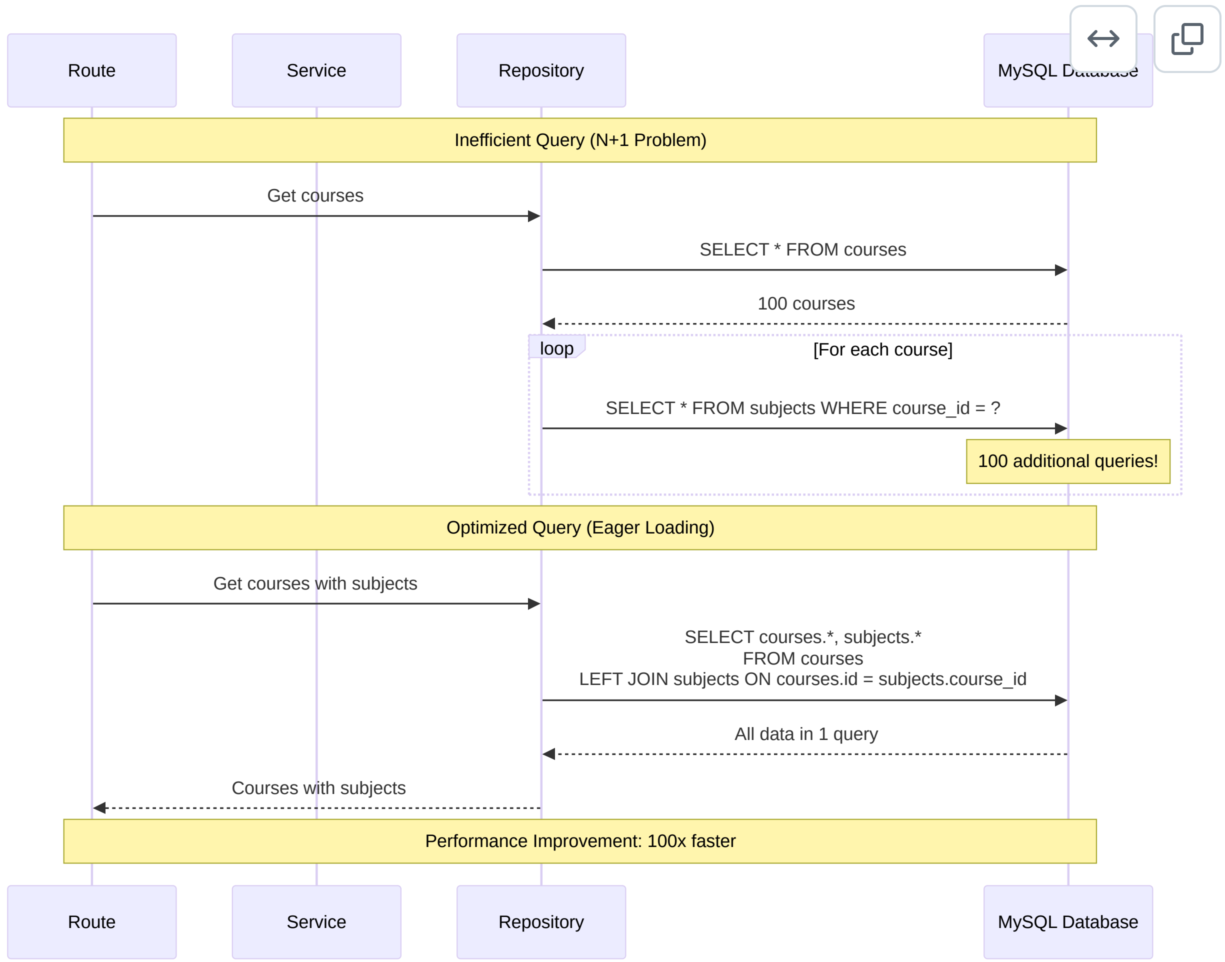
Frontend Optimizations



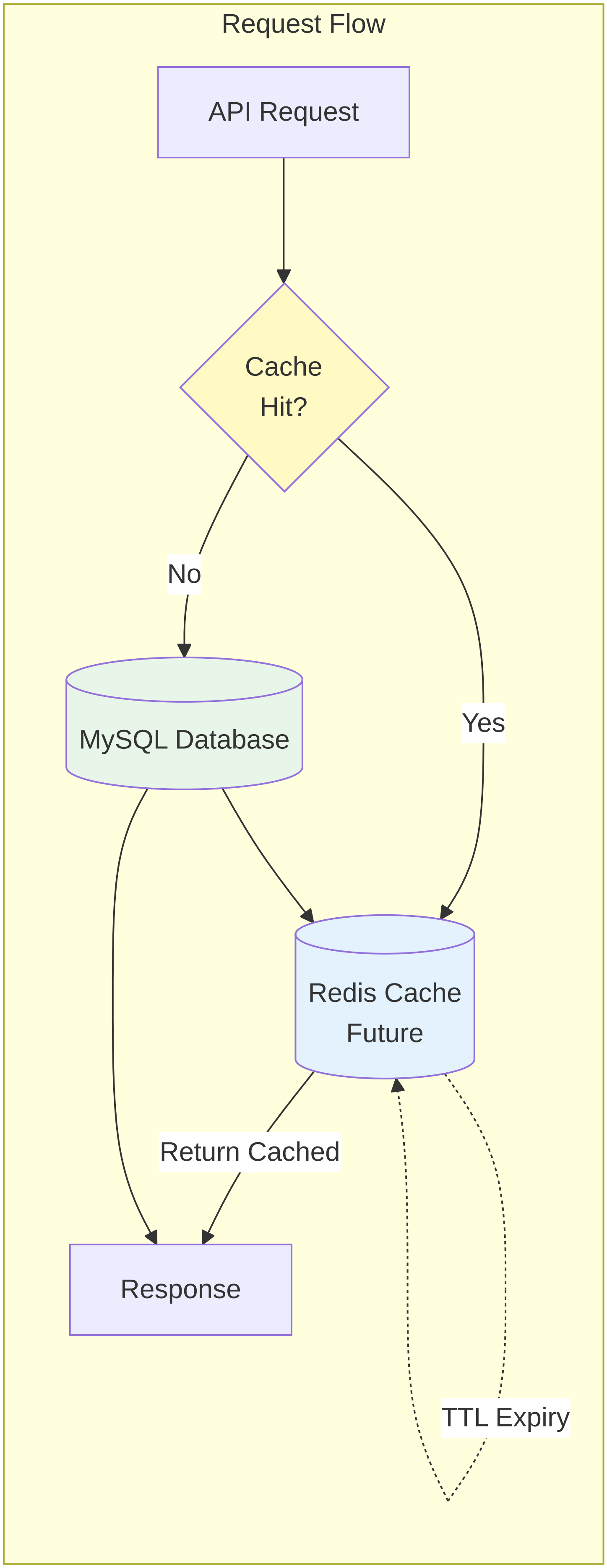
Backend Optimizations



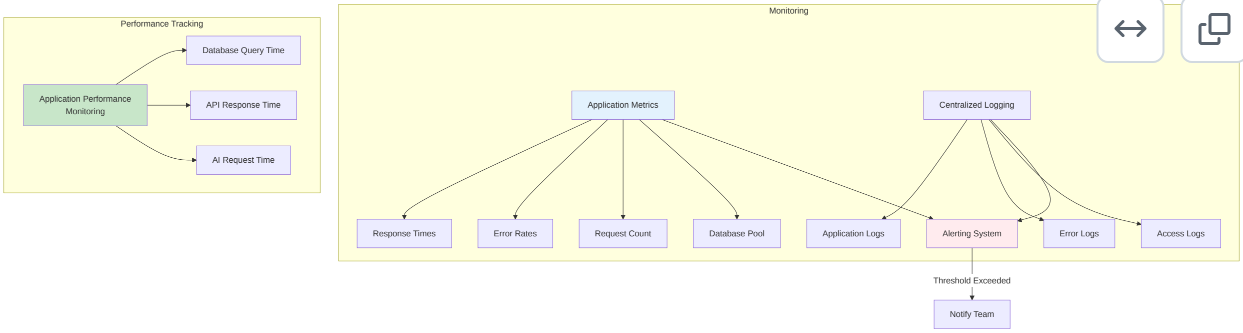
Database Query Optimization


Route
Service
Repository
MySQL Database
Inefficient Query (N+1 Problem)
Get courses
SELECT * FROM courses
100 courses
loop
[For each course]
SELECT * FROM subjects WHERE course_id = ?
100 additional queries!
Optimized Query (Eager Loading)
Get courses with subjects
SELECT courses.*, subjects.*
FROM courses
LEFT JOIN subjects ON courses.id = subjects.course_id
All data in 1 query
Courses with subjects
Performance Improvement: 100x faster
Route
Service
Repository
MySQL Database

Caching Strategy (Future)


Request Flow
API Request
Cache Hit?
No
Yes
MySQL Database
Redis Cache Future
Return Cached
Response
TTL Expiry

Load Testing & Monitoring



Summary

This document provides a comprehensive overview of the CourseWagon system architecture, covering:

1. **High-level system design** with clear separation of concerns
2. **Frontend architecture** using Angular 19 with modern patterns
3. **Backend architecture** following layered design principles
4. **Database design** with proper relationships and indexing
5. **Authentication & authorization** using Firebase and JWT
6. **AI integration** with Google Gemini for content generation
7. **Multi-cloud storage** with automatic failover
8. **Email system** with background task processing
9. **Deployment** on cloud infrastructure with CI/CD
10. **Security** measures at multiple layers
11. **Performance optimizations** for scalability

All diagrams use Mermaid syntax and can be:

- Rendered on GitHub

- Exported to images for presentations
- Used in documentation tools
- Integrated with Markdown-based systems

Document Version: 1.0 **Last Updated:** October 8, 2025 **Project:** CourseWagon - AI-Powered Educational Platform **Live URL:** https://www.coursewagon.live