Web Programming Final Project Documentation

- Muhammad Rasyad Lubis (5054231 010)
- Faiz Muhammad Kautsar (5054231 013)
- Shalahuddin Ahmad Cahyoga (5054231 014)
- Abdan Hafidz (5054231 021)

Quzuu - Interactive Online Examination Platform

Demonstration Video
Paper File : COMING SOON
Slides/PPT
Front-End Link : quzuu.vercel.app
Backend API Gateway : lifedebugger-quzuu-api-dev.hf.space
Postman Documentation

📋 Executive Summary

Quzuu is an innovative interactive online examination platform designed to revolutionize the way educational assessments are conducted. The platform addresses the growing need for diverse, engaging, and technologically advanced examination systems in modern education. By implementing a comprehensive microservice architecture and incorporating cutting-edge features like Block Code Puzzles, Quzuu provides an unparalleled examination experience that goes beyond traditional multiple-choice questions.

🎯 Project Overview

Problem Statement

Traditional online examination platforms often lack variety in question types and fail to engage students effectively, particularly in technical subjects like programming. Most existing platforms are limited to basic question formats and don't provide interactive elements that can properly assess practical skills.

Solution

Quzuu introduces a comprehensive examination platform featuring:

- Eight distinct question types including innovative Block Code Puzzles
- Interactive drag-and-drop programming interfaces
- Integration with automated code evaluation systems
- Modern microservice architecture for scalability
- Advanced authentication and security features

Key Innovation: Block Code Puzzle

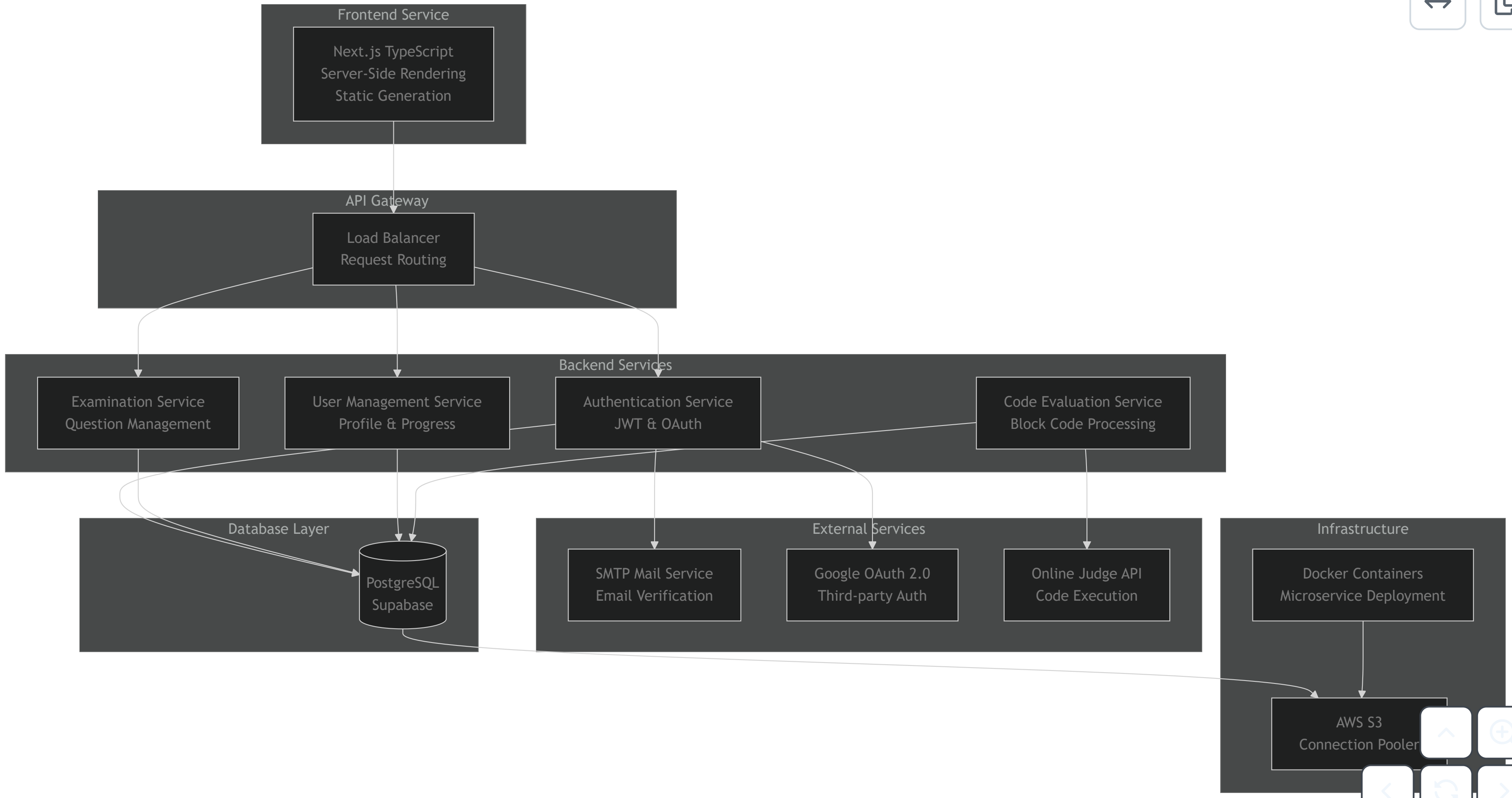
The standout feature of Quzuu is the **Block Code Puzzle** system, which allows students to:

- Drag and drop code blocks to complete programs

- Fill in missing syntax through interactive typing
- Visualize program structure through block-based interfaces
- Receive real-time feedback on code construction

🏗️ System Architecture

Microservice Architecture Overview



Technology Stack

Frontend Technologies:

- **Next.js 14** with TypeScript for type safety
- **React 18** with hooks and context API

- **Tailwind CSS** for responsive design
- **Axios** for HTTP client communications

Backend Technologies:

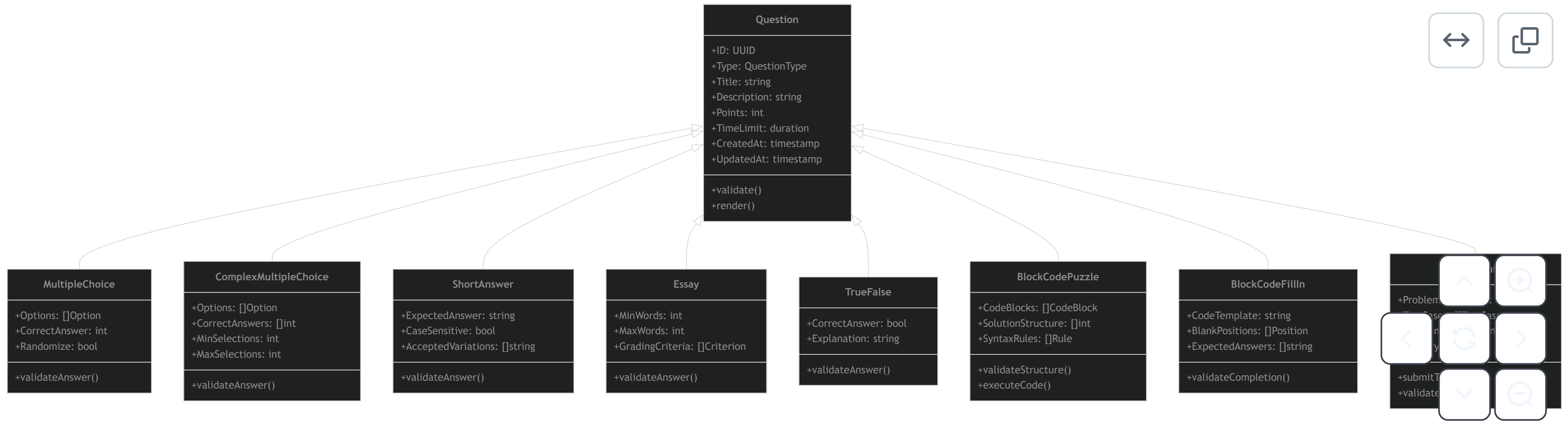
- **Go 1.21** with Gin framework for high performance
- **GORM** as Object-Relational Mapping tool
- **JWT-Go** for token-based authentication
- **Google OAuth 2.0** for external authentication

Database & Storage:

- **PostgreSQL 15** as primary database
- **Supabase** for database hosting and management
- **AWS S3** for connection pooling and file storage

📊 Detailed Feature Specifications

1. Question Type System



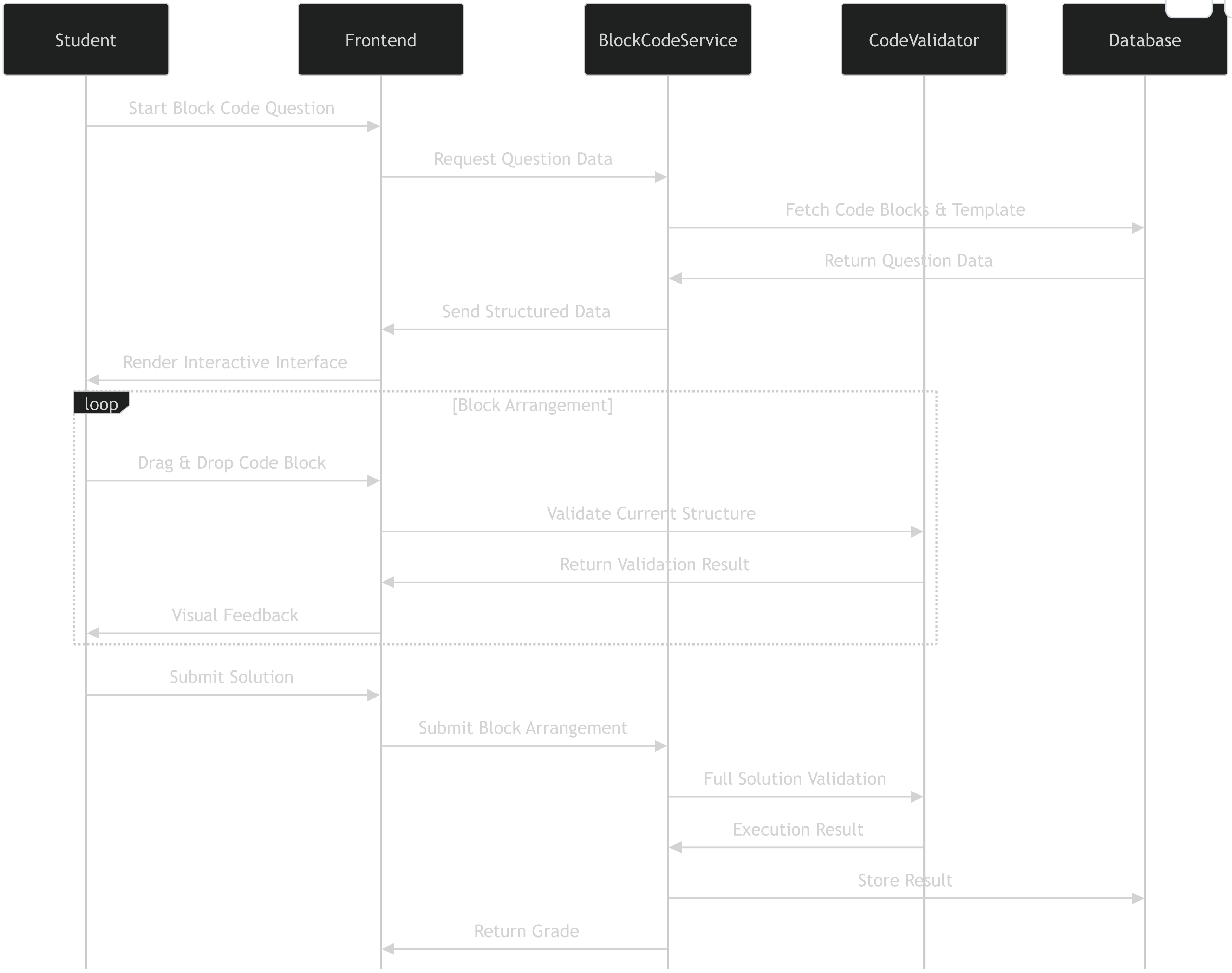
2. Block Code Puzzle Implementation

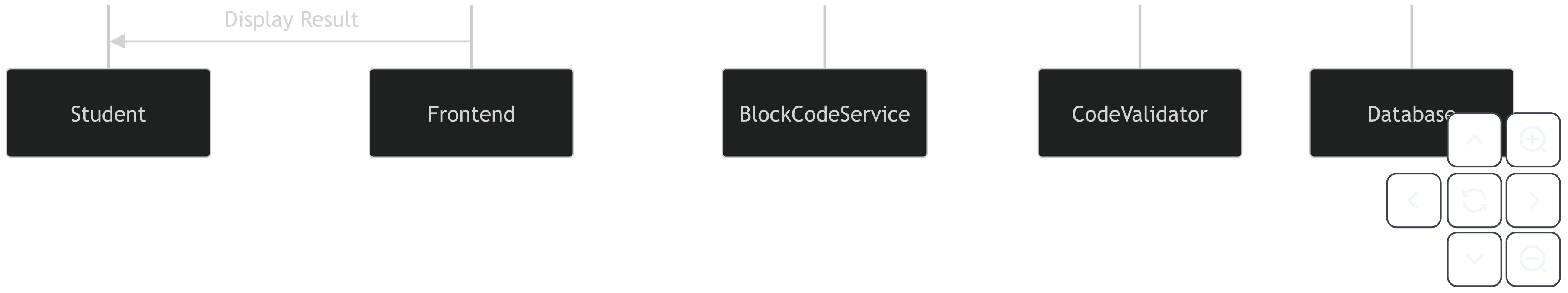
The Block Code Puzzle system represents the most innovative aspect of Quzuu:

Technical Implementation:

- **Drag & Drop Engine**: Built using native React useState for click, drag and drop features
- **Code Block Rendering**: SVG-based visual representation of code blocks
- **Syntax Validation**: Real-time syntax checking as blocks are arranged
- **Execution Engine**: Server-side code compilation and execution for validation

User Interaction Flow:

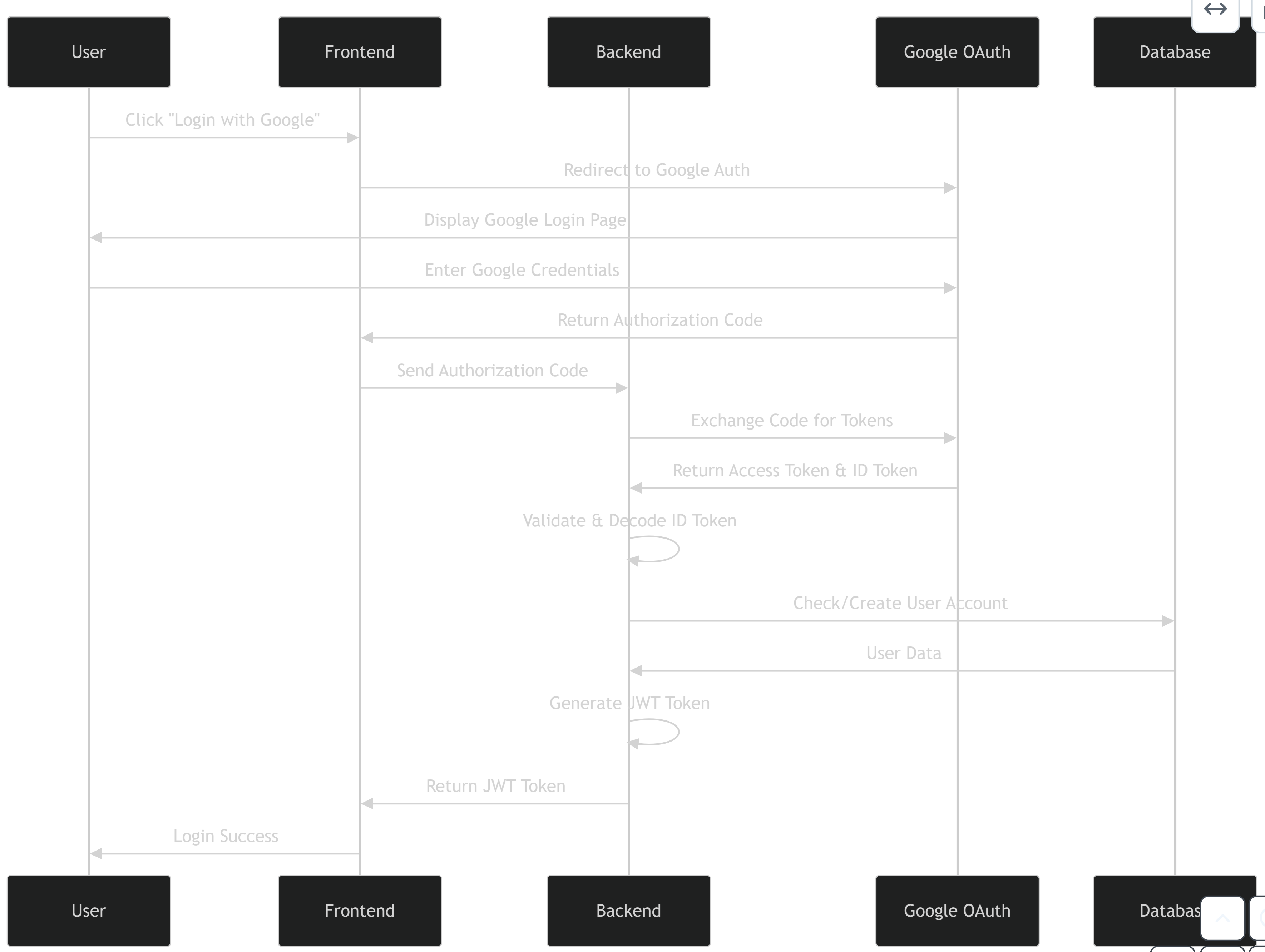

Student
Frontend
BlockCodeService
CodeValidator
Database
Start Block Code Question
Request Question Data
Fetch Code Blocks & Template
Return Question Data
Send Structured Data
Render Interactive Interface
loop
[Block Arrangement]
Drag & Drop Code Block
Validate Current Structure
Return Validation Result
Visual Feedback
Submit Solution
Submit Block Arrangement
Full Solution Validation
Execution Result
Store Result
Return Grade



🔐 Authentication & Security System

Google OAuth 2.0 Integration

Quzuu implements Google OAuth 2.0 as the primary external authentication method, providing users with a seamless login experience using their Google accounts.


User
Frontend
Backend
Google OAuth
Database
Click "Login with Google"
Redirect to Google Auth
Display Google Login Page
Enter Google Credentials
Return Authorization Code
Send Authorization Code
Exchange Code for Tokens
Return Access Token & ID Token
Validate & Decode ID Token
Check/Create User Account
User Data
Generate JWT Token
Return JWT Token
Login Success
User
Frontend
Backend
Google OAuth
Database

Google OAuth Implementation Details

Frontend Implementation (Next.js):

```javascript
// Google OAuth Configuration
const googleAuth = {
  clientId: process.env.NEXT_PUBLIC_GOOGLE_CLIENT_ID,
  redirectUri: process.env.NEXT_PUBLIC_GOOGLE_REDIRECT_URI,
  scope: 'openid email profile'
};

// OAuth Login Handler
const handleGoogleLogin = async () => {
  const authUrl = `https://accounts.google.com/oauth/authorize?` +
    `client_id=${googleAuth.clientId}&` +
    `redirect_uri=${googleAuth.redirectUri}&` +
    `response_type=code&` +
    `scope=${googleAuth.scope}`;

  window.location.href = authUrl;
};
```

Backend Implementation (Go):

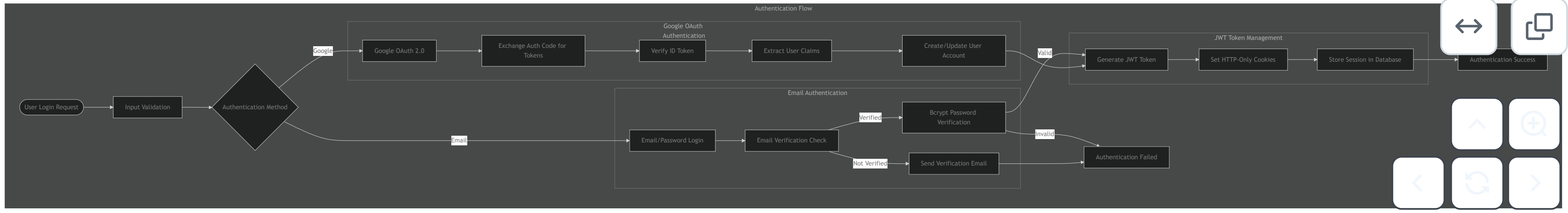
```go
package services

import (
        "context"
        "errors"

        "github.com/google/uuid"
```

```go
	"godp.abdanhafidz.com/models"
	"godp.abdanhafidz.com/repositories"
	"google.golang.org/api/idtoken"
)

type GoogleAuthService struct {
	Service[models.ExternalAuth, models.AuthenticatedUser]
}

func (s *GoogleAuthService) Authenticate(isAgree bool) {
	GoogleAuth := repositories.GetExternalAccountByOauthId(s.Constructor.OauthID)
	payload, errGoogleAuth := idtoken.Validate(context.Background(), s.Constructor.OauthID, "")
	s.Error = errGoogleAuth
	if errGoogleAuth != nil {
		s.Exception.Unauthorized = true
		s.Exception.Message = "Oauth Provider Failed Login (Google Authentication)"
		return
	}
	email := payload.Claims["email"]
	checkRegisteredEmail := repositories.GetAccountbyEmail(email.(string))
	if !checkRegisteredEmail.NoRecord {
		token, _ := GenerateToken(&checkRegisteredEmail.Result)
		checkRegisteredEmail.Result.Password = "SECRET"
		s.Result = models.AuthenticatedUser{
			Account: checkRegisteredEmail.Result,
			Token:   token,
		}
		return
	}
	if GoogleAuth.NoRecord {
		if !isAgree {
			s.Exception.BadRequest = true
			s.Exception.Message = "Please agree to the terms and conditions to create an account"
			return
		}
		s.Constructor.OauthProvider = "Google"
```

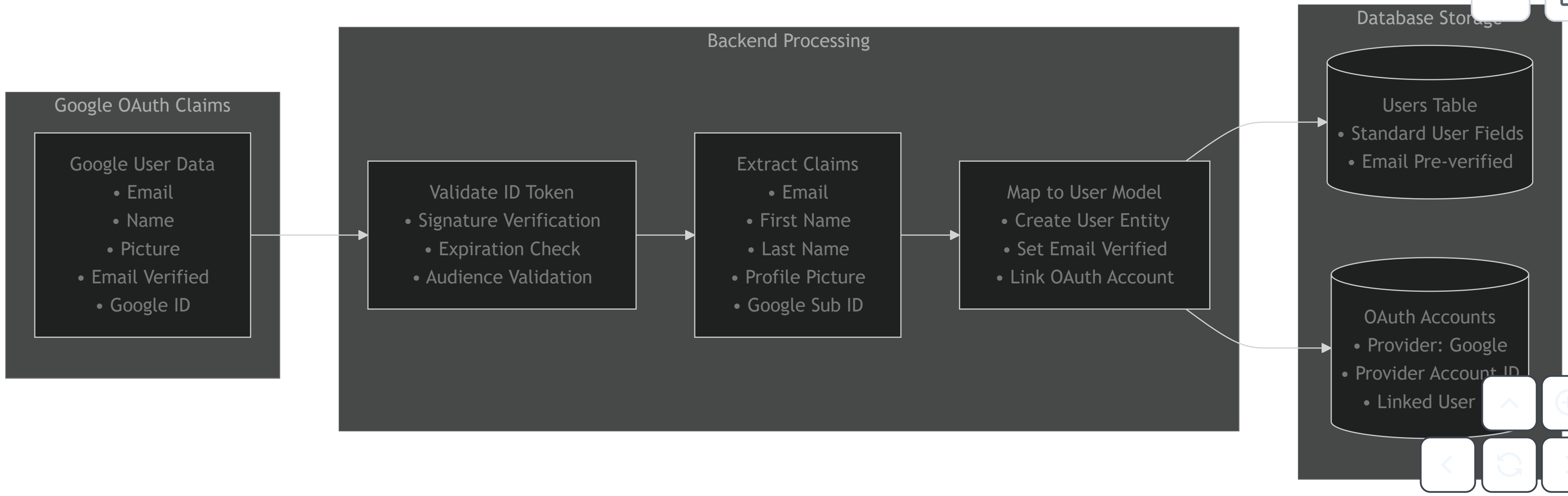
```
        createAccount := repositories.CreateAccount(models.Account{
                Id:               uuid.New(),
                Username:         payload.Claims["name"].(string),
                Email:            email.(string),
                IsEmailVerified: true,
        })

        s.Constructor.AccountId = createAccount.Result.Id
        createGoogleAuth := repositories.CreateExternalAuth(s.Constructor)

        GoogleAuth.Result.AccountId = createGoogleAuth.Result.AccountId
        userProfile := UserProfileService{}
        userProfile.Constructor.AccountId = GoogleAuth.Result.AccountId
        userProfile.Create()
        if userProfile.Error != nil {
                s.Error = userProfile.Error
                return
        }
        s.Error = createGoogleAuth.RowsError
        s.Error = errors.Join(s.Error, createAccount.RowsError)
}

accountData := repositories.GetAccountById(GoogleAuth.Result.AccountId)
token, err_tok := GenerateToken(&accountData.Result)

if err_tok != nil {
        s.Error = errors.Join(s.Error, err_tok)
}

accountData.Result.Password = "SECRET"
s.Result = models.AuthenticatedUser{
        Account: accountData.Result,
        Token:   token,
}
s.Error = accountData.RowsError
```

```
}
```

Multi-layered Authentication



Google OAuth User Data Flow

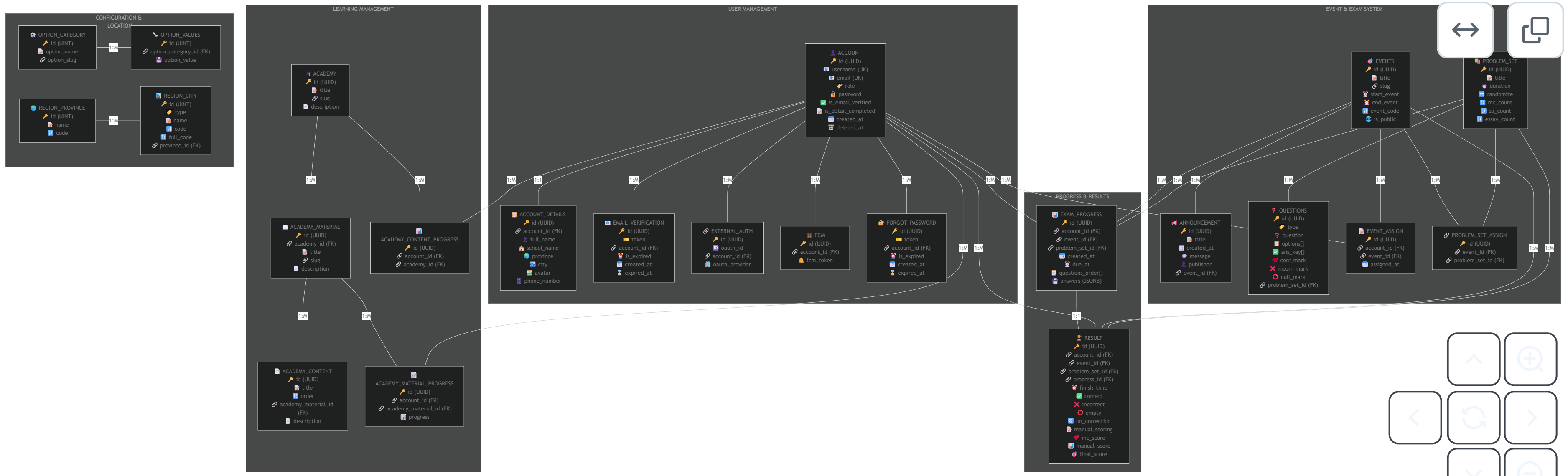


Security Features Implementation

1. **JWT Token System**
    - Access tokens with configurable expiration
    - Secure token storage using httpOnly cookies
    - Token validation middleware for protected routes
2. **Email Verification System**
    - SMTP integration for email delivery
    - Time-limited verification tokens
    - Account activation workflow
3. **Google OAuth 2.0 Integration**
    - Complete OAuth 2.0 flow implementation
    - ID token validation with Google's public keys
    - Automatic user account creation and linking
    - Profile data synchronization
4. **Password Security**
    - Bcrypt hashing for password storage
    - Password strength validation
    - Secure password reset functionality

💾 Database Design & Management

Entity Relationship Diagram



Database Operations & GORM Implementation

CRUD Operations Implementation:

```
dbHost := os.Getenv("DB_HOST")
dbPort := os.Getenv("DB_PORT")
dbUser := os.Getenv("DB_USER")
dbPassword := os.Getenv("DB_PASSWORD")
dbName := os.Getenv("DB_NAME")
Salt := os.Getenv("SALT")
dsn := "host=" + dbHost + " user=" + dbUser + " password=" + dbPassword + " dbname=" + dbName + " port=" + dbPo
DB, err = gorm.Open(postgres.Open(dsn), &gorm.Config{TranslateError: true})
```

```
package repositories

import (
```

```go
		"fmt"
		"godp.abdanhafidz.com/config"
		"gorm.io/gorm"
		"strings"
)

type Repositories interface {
		FindAllPaginate()
		Where()
		Find()
		Create()
		Update()
		CustomQuery()
		Delete()
}
type PaginationConstructor struct {
		Limit    int
		Offset   int
		Filter   string
		FilterBy string
}

type PaginationMetadata struct {
		TotalRecords int `json:"total_records"`
		TotalPages   int `json:"total_pages"`
		CurrentPage  int `json:"current_page"`
		PageSize     int `json:"page_size"`
}

type CustomQueryConstructor struct {
		SQL    string
		Values interface{}
}

type Repository[TConstructor any, TResult any] struct {
		Constructor TConstructor
```

```go
        Pagination  PaginationConstructor
        CustomQuery CustomQueryConstructor
        Result      TResult
        Transaction *gorm.DB
        RowsCount   int
        NoRecord    bool
        RowsError   error
}

func Construct[TConstructor any, TResult any](constructor ...TConstructor) *Repository[TConstructor, TResult] {
        if len(constructor) == 1 {
                return &Repository[TConstructor, TResult]{
                        Constructor: constructor[0],
                        Transaction: config.DB,
                }
        }
        return &Repository[TConstructor, TResult]{
                Constructor: constructor[0],
                Transaction: config.DB.Begin(),
        }
}
func (repo *Repository[T1, T2]) Transactions(transactions ...func(*Repository[T1, T2]) *gorm.DB) {
        for _, tx := range transactions {
                repo.Transaction = tx(repo)
                if repo.RowsError != nil {
                        return
                }
        }
}
func WhereGivenConstructor[T1 any, T2 any](repo *Repository[T1, T2]) *gorm.DB {
        tx := repo.Transaction.Where(&repo.Constructor)
        repo.RowsCount = int(tx.RowsAffected)
        repo.NoRecord = repo.RowsCount == 0
        repo.RowsError = tx.Error
        return tx
}
```

```go
func Find[T1 any, T2 any](repo *Repository[T1, T2]) *gorm.DB {
	tx := repo.Transaction.Find(&repo.Result)
	repo.RowsCount = int(tx.RowsAffected)
	repo.NoRecord = repo.RowsCount == 0
	repo.RowsError = tx.Error
	return tx
}

func FindAllPaginate[T1 any, T2 any](repo *Repository[T1, T2]) *gorm.DB {
	tx := repo.Transaction.Limit(repo.Pagination.Limit).Offset(repo.Pagination.Offset)

	tx = buildFilter(tx, repo.Pagination)

	tx = tx.Find(&repo.Result)

	repo.RowsCount = int(tx.RowsAffected)
	repo.NoRecord = repo.RowsCount == 0
	repo.RowsError = tx.Error

	return tx
}

func Create[T1 any](repo *Repository[T1, T1]) *gorm.DB {
	tx := repo.Transaction.Create(&repo.Constructor)
	repo.RowsCount = int(tx.RowsAffected)
	repo.NoRecord = repo.RowsCount == 0
	repo.RowsError = tx.Error
	repo.Result = repo.Constructor
	return tx
}

func Update[T1 any](repo *Repository[T1, T1]) *gorm.DB {
	tx := repo.Transaction.Save(&repo.Constructor)
	repo.RowsCount = int(tx.RowsAffected)
	repo.NoRecord = repo.RowsCount == 0
	repo.RowsError = tx.Error
```

```go
		repo.Result = repo.Constructor
		return tx
	}

	func Delete[T1 any](repo *Repository[T1, T1]) *gorm.DB {
		tx := repo.Transaction.Delete(&repo.Constructor)
		repo.RowsCount = int(tx.RowsAffected)
		repo.NoRecord = repo.RowsCount == 0
		repo.RowsError = tx.Error
		return tx
	}

	func CustomQuery[T1 any, T2 any](repo *Repository[T1, T2]) *gorm.DB {
		tx := repo.Transaction.Raw(repo.CustomQuery.SQL, repo.CustomQuery.Values).Scan(&repo.Result)
		repo.RowsCount = int(tx.RowsAffected)
		repo.NoRecord = repo.RowsCount == 0
		repo.RowsError = tx.Error
		return tx
	}

	func buildFilter(db *gorm.DB, pagination PaginationConstructor) *gorm.DB {
		if pagination.Filter != "" && pagination.FilterBy != "" {
			filterFields := strings.Split(pagination.FilterBy, ",")
			filterValues := strings.Split(pagination.Filter, ",")

			for i, field := range filterFields {
				if i >= len(filterValues) {
					break
				}
				filterValue := filterValues[i]
				if filterValue != "" {
					condition := fmt.Sprintf("%s ILIKE ?", field)
					db = db.Where(condition, "%"+filterValue+"%")
				}
			}
		}
```

```
        return db
    }
```

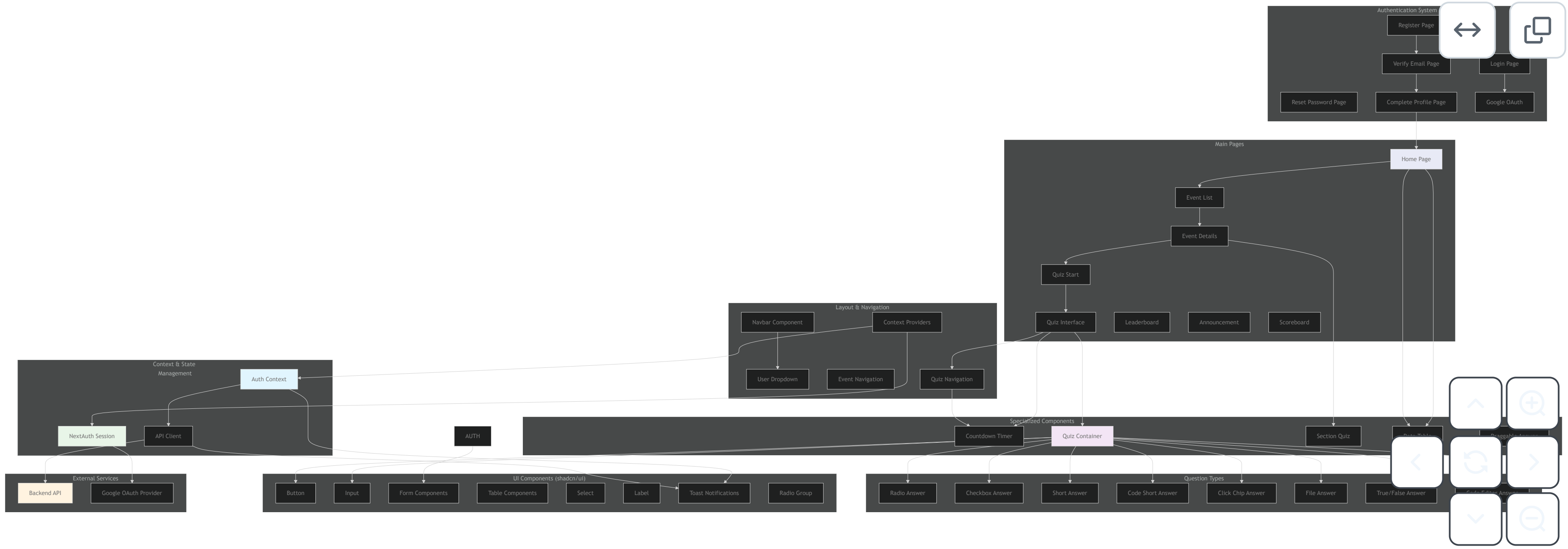
Auto Migration System:

```
        db.Exec("CREATE EXTENSION IF NOT EXISTS \"uuid-ossp\";")
        if err := db.AutoMigrate(&models.Account{}); err != nil {
                log.Fatal(err)
        }
        if err := db.AutoMigrate(&models.AccountDetails{}); err != nil {
                log.Fatal(err)
        }
        if err := db.AutoMigrate(&models.EmailVerification{}); err != nil {
                log.Fatal(err)
        }
        if err := db.AutoMigrate(&models.ExternalAuth{}); err != nil {
                log.Fatal(err)
        }
        if err := db.AutoMigrate(&models.FCM{}); err != nil {
                log.Fatal(err)
        }
        if err := db.AutoMigrate(&models.ForgotPassword{}); err != nil {
                log.Fatal(err)
        }
        if err := db.AutoMigrate(&models.Events{}); err != nil {
                log.Fatal(err)
        }
        if err := db.AutoMigrate(&models.Announcement{}); err != nil {
                log.Fatal(err)
        }
        if err := db.AutoMigrate(&models.ProblemSet{}); err != nil {
                log.Fatal(err)
        }
        if err := db.AutoMigrate(&models.Questions{}); err != nil {
                log.Fatal(err)
```

```
	}
	if err := db.AutoMigrate(&models.EventAssign{}); err != nil {
		log.Fatal(err)
	}
	if err := db.AutoMigrate(&models.ProblemSetAssign{}); err != nil {
		log.Fatal(err)
	}
	if err := db.AutoMigrate(&models.ExamProgress{}); err != nil {
		log.Fatal(err)
	}
	if err := db.AutoMigrate(&models.ExamProgress_Result{}); err != nil {
		log.Fatal(err)
	}
	if err := db.AutoMigrate(&models.Result{}); err != nil {
		log.Fatal(err)
	}
```

📱 Frontend Architecture & Implementation

Component Architecture



React Context Implementation

```
// Authentication Context
"use client"

import React, {
    createContext,
    useContext,
    useState,
    useEffect,
    ReactNode,
} from "react"
import { useRouter, usePathname } from "next/navigation"
import { useSession } from "next-auth/react"
import { getAuthToken, removeAuthToken, syncNextAuthSession } from "@/lib/api"
import { toast } from "@/hooks/use-toast"
import Cookies from "js-cookie"
```

```typescript
interface UserData {
    id?: string
    username: string
    email: string
    avatar?: string
    fullName?: string
    schoolName?: string
    province?: string
    city?: string
    phoneNumber?: string
    isEmailVerified: boolean
    isProfileComplete: boolean
}

interface AuthContextType {
    user: UserData | null
    isLoading: boolean
    isAuthenticated: boolean
    logout: () => Promise<void>
    refreshUserData: () => Promise<void>
    setEmailVerified: (value: boolean) => void
    setProfileComplete: (value: boolean) => void
}

const defaultAuthContext: AuthContextType = {
    user: null,
    isLoading: true,
    isAuthenticated: false,
    logout: async () => {},
    refreshUserData: async () => {},
    setEmailVerified: () => {},
    setProfileComplete: () => {},
}

export const AuthContext = createContext<AuthContextType>(defaultAuthContext)
export const useAuth = () => useContext(AuthContext)

export const AuthProvider = ({ children }: { children: ReactNode }) => {
    const [user, setUser] = useState<UserData | null>(null)
    const [isLoading, setIsLoading] = useState(true)
    const [isAuthenticated, setIsAuthenticated] = useState(false)
    const [initializationComplete, setInitializationComplete] = useState(false)

    const router = useRouter()
    const pathname = usePathname()

    // Get NextAuth session
    const { data: session, status: sessionStatus } = useSession()

    console.log("🔍 AuthContext Debug:", {
        sessionStatus,
        hasSession: !!session,
        hasBackendToken: !!session?.backendToken,
        isLoading,
        initializationComplete,
        isAuthenticated,
        userEmail: user?.email
    })

    const setEmailVerified = (value: boolean) => {
        if (user) {
            setUser({
                ...user,
                isEmailVerified: value,
            })
        }
    }

    const setProfileComplete = (value: boolean) => {
        console.log("🔄 Setting profile complete to:", value)
        if (user) {
            const updatedUser = {
                ...user,
                isProfileComplete: value,
            }
            setUser(updatedUser)
            console.log("✅ User updated in context:", updatedUser)
        } else {
            console.warn("⚠️ No user found when trying to set profile complete")
        }
    }

    // Function to parse JWT and extract data
    const parseJwt = (token: string) => {
        try {
            const base64Url = token.split(".")[1]
            const base64 = base64Url.replace(/-/g, "+").replace(/_/g, "/")
            const jsonPayload = decodeURIComponent(
                atob(base64)
                    .split("")
                    .map((c) => {
                        return (
                            "%" +
                            ("00" + c.charCodeAt(0).toString(16)).slice(-2)
                        )
                    })
                    .join(""),
            )

            return JSON.parse(jsonPayload)
        } catch (e) {
            console.error("Error parsing JWT:", e)
            return null
        }
    }
```

```
// Handle OAuth session (NextAuth)
const handleOAuthSession = async (session: any) => {
    if (!session?.backendToken) return false

    try {
        console.log("🔄 Processing OAuth session...")

        const accountData = await syncNextAuthSession(session)

        if (accountData) {
            console.log("✅ OAuth session processed successfully")

            // Check localStorage for manual profile completion flag
            const manualProfileComplete = localStorage.getItem("profile_completed") === "true"
            const backendProfileComplete = accountData.is_detail_completed || false

            // Use manual flag if backend hasn't been updated yet
            const finalProfileComplete = manualProfileComplete || backendProfileComplete

            console.log("🔍 Profile complete status:", {
                backend: backendProfileComplete,
                manual: manualProfileComplete,
                final: finalProfileComplete
            })

            setUser({
                id: accountData.id,
                username: accountData.username || session.user?.name || "User",
                email: accountData.email || session.user?.email || "",
                avatar: session.user?.image,
                fullName: session.user?.name,
                isEmailVerified: accountData.is_email_verified || false,
                isProfileComplete: finalProfileComplete,
            })

            setIsAuthenticated(true)
```

```
                return true
            }
        } catch (error) {
            console.error("❌ Failed to process OAuth session:", error)
        }

        return false
    }

    // Handle regular authentication (email/password)
    const handleRegularAuth = async () => {
        const token = getAuthToken()

        if (!token) {
            console.log("❌ No auth token found")
            setUser(null)
            setIsAuthenticated(false)
            return false
        }

        try {
            console.log("🔄 Processing regular authentication...")

            // Verify token before making API call
            const tokenData = parseJwt(token)
            const currentTime = Math.floor(Date.now() / 1000)

            if (tokenData && tokenData.exp && tokenData.exp < currentTime) {
                console.log("⏰ Token expired")
                throw new Error("Token expired")
            }

            // Make API call with valid token
            const response = await fetch(
                `${process.env.NEXT_PUBLIC_API_BASE_URL}/user/me`,
                {
```

```
        method: "GET",
        headers: {
            "Content-Type": "application/json",
            "Authorization": "Bearer " + token,
        },
    },
)

if (!response.ok) {
    throw new Error(`API error: ${response.status}`)
}

const userData = await response.json()

if (userData && userData.data && userData.data.account) {
    const account = userData.data.account
    const details = userData.data.details || {}

    console.log("✅ Regular auth processed successfully")

    setUser({
        id: account.id,
        username: account.username,
        email: account.email,
        avatar: details.avatar,
        fullName: details.full_name,
        schoolName: details.school_name,
        province: details.province,
        city: details.city,
        phoneNumber: details.phone_number,
        isEmailVerified: account.is_email_verified || false,
        isProfileComplete: account.is_detail_completed || false,
    })

    setIsAuthenticated(true)
    return true
```

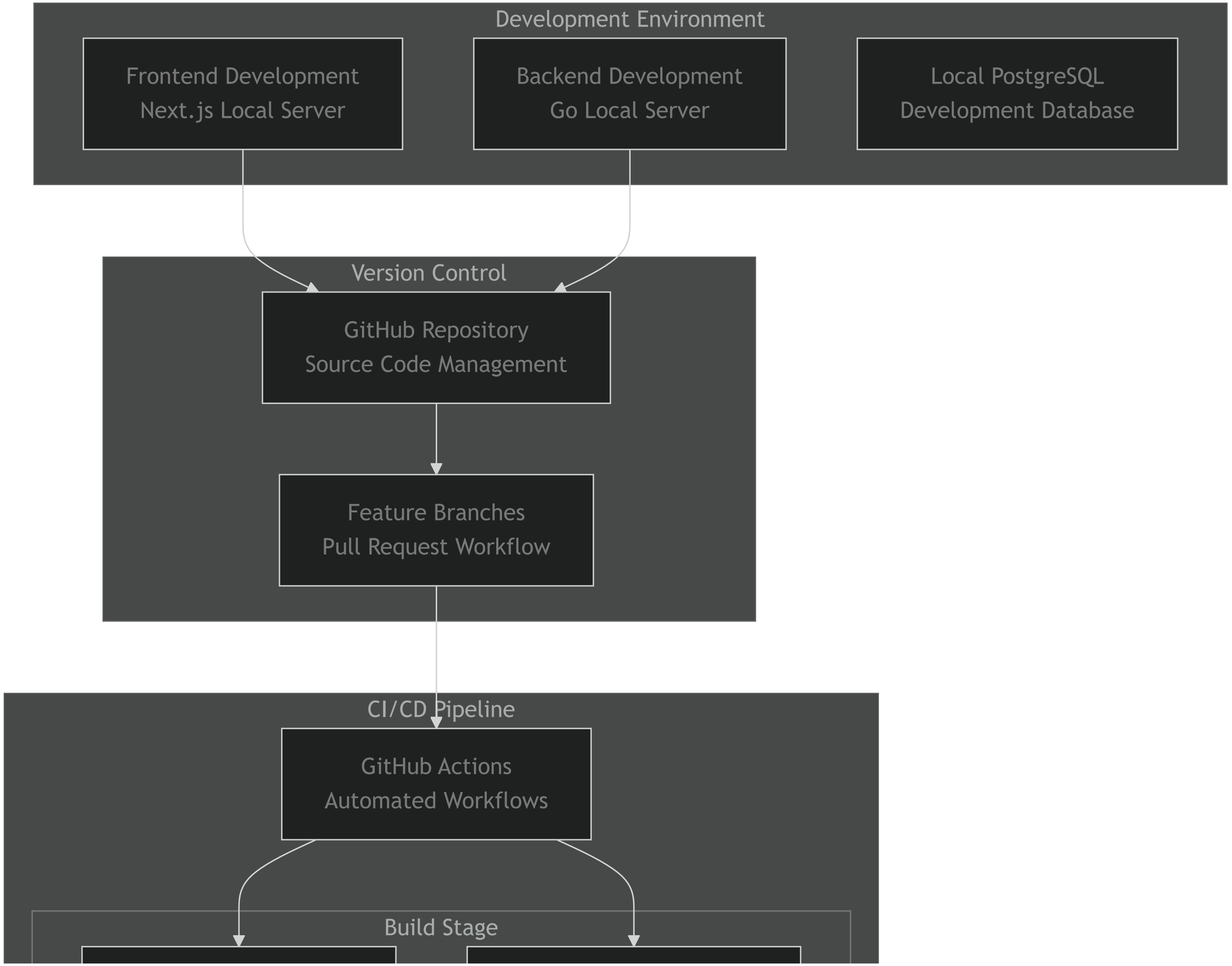
```
            }
        } catch (error) {
            console.error("❌ Regular auth failed:", error)

            if (
                error instanceof Error &&
                (error.message.includes("Token expired") ||
                    error.message.includes("API error: 401"))
            ) {
                // Clear expired token
                removeAuthToken()
                setUser(null)
                setIsAuthenticated(false)
            }
        }

        return false
    }

    // Main initialization effect
    useEffect(() => {
        const initializeAuth = async () => {
            console.log("🚀 Initializing authentication...")

            // Wait for NextAuth to be ready
            if (sessionStatus === "loading") {
                console.log("⏳ Waiting for NextAuth...")
                return
            }

            setIsLoading(true)

            try {
                let authSuccess = false

                // Try OAuth first if available
                if (session?.backendToken) {
                    authSuccess = await handleOAuthSession(session)
                }

                // Fall back to regular auth if OAuth not available or failed
                if (!authSuccess) {
                    authSuccess = await handleRegularAuth()
                }

                // If no authentication method worked
                if (!authSuccess) {
                    console.log("❌ No valid authentication found")
                    setUser(null)
                    setIsAuthenticated(false)
                }

            } catch (error) {
                console.error("❌ Auth initialization failed:", error)
                setUser(null)
                setIsAuthenticated(false)
            } finally {
                setIsLoading(false)
                setInitializationComplete(true)
                console.log("✅ Auth initialization complete")
            }
        }

        initializeAuth()
    }, [session, sessionStatus])

    // Refreshable auth function
    const refreshUserData = async (): Promise<void> => {
        console.log("🔄 Refreshing user data...")
        setIsLoading(true)

        try {
            let success = false

            if (session?.backendToken) {
                success = await handleOAuthSession(session)
            }

            if (!success) {
                success = await handleRegularAuth()
            }

            if (!success) {
                setUser(null)
                setIsAuthenticated(false)
            }
        } catch (error) {
            console.error("❌ Refresh failed:", error)
            setUser(null)
            setIsAuthenticated(false)
        } finally {
            setIsLoading(false)
        }
    }

    // Handle routing based on authentication and profile completion
    useEffect(() => {
        // Don't redirect during initialization
        if (!initializationComplete || isLoading) return

        console.log("🧭 Checking routing rules:", {
            isAuthenticated,
            pathname,
            isProfileComplete: user?.isProfileComplete,
            isEmailVerified: user?.isEmailVerified,
        })

        // Override email verification for testing
```

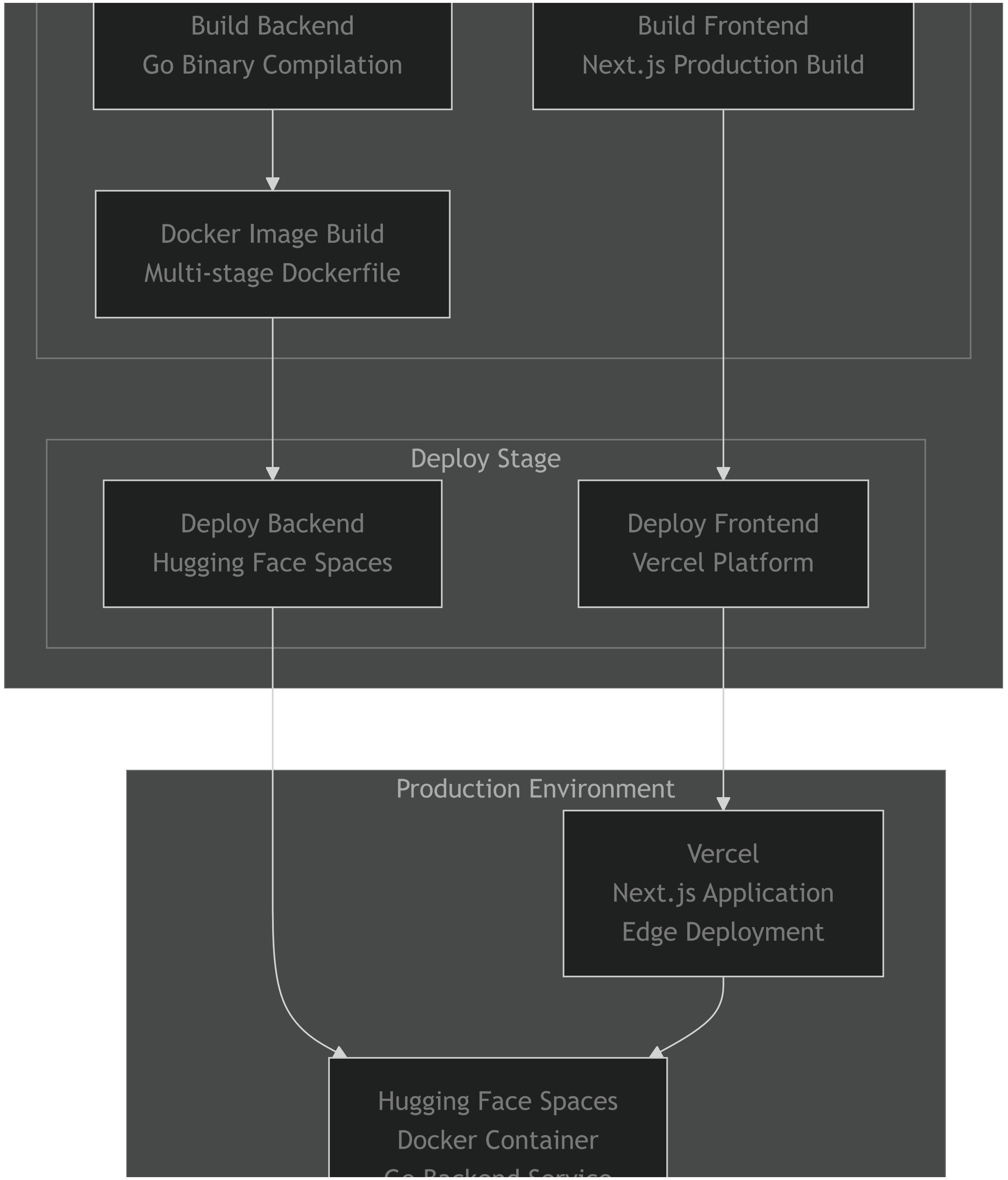
```
        if (user && !user.isEmailVerified) {
            const shouldOverride =
                localStorage.getItem("override_verification") === "true"
            if (shouldOverride) {
                console.log("🔧 OVERRIDING email verification for debugging")
                setEmailVerified(true)
                localStorage.setItem("email_verified", "true")
                return
            }
        }

        const publicPaths = [
            "/login",
            "/register",
            "/forgot-password",
            "/verify-email",
        ]

        const isOnPublicPath = publicPaths.some(
            (path) => pathname === path || pathname.startsWith(`${path}/`),
        )

        if (isOnPublicPath) return

        if (!isAuthenticated) {
            console.log("🔒 Not authenticated, redirecting to login")
            router.push("/login")
            return
        }

        if (user) {
            if (!user.isEmailVerified && !pathname.includes("/verify-email")) {
                console.log("📧 Email not verified, redirecting to verify-email")
                router.push(
                    `/verify-email?email=${encodeURIComponent(user.email)}`,
                )
                return
            }

            if (
                user.isEmailVerified &&
                !user.isProfileComplete &&
                pathname !== "/complete-profile"
            ) {
                console.log("👤 Profile not complete, redirecting to complete-profile")
                router.push("/complete-profile")
                return
            }
        }
    }, [isAuthenticated, initializationComplete, isLoading, user, pathname, router])

    const logout = async () => {
        try {
            console.log("🚪 Logging out...")

            // Clear all auth data
            Cookies.remove("quzuu_auth_token", { path: "/" })
            localStorage.removeItem("email_verified")
            localStorage.removeItem("profile_completed")
            setUser(null)
            setIsAuthenticated(false)
            setInitializationComplete(false)

            toast({
                title: "Logged Out",
                description: "You have been successfully logged out.",
            })

            router.push("/login")
        } catch (error) {
            console.error("❌ Logout failed:", error)
            // Force clear everything even if error
            Cookies.remove("quzuu_auth_token", { path: "/" })
            localStorage.removeItem("email_verified")
            localStorage.removeItem("profile_completed")
            setUser(null)
            setIsAuthenticated(false)
            setInitializationComplete(false)

            router.push("/login")
        }
    }

    return (
        <AuthContext.Provider
            value={{
                user,
                isLoading,
                isAuthenticated,
                logout,
                refreshUserData,
                setEmailVerified,
                setProfileComplete,
            }}
        >
            {children}
        </AuthContext.Provider>
    )
}

// Exam Context
interface ExamContextType {
  currentExam: Exam | null;
  currentQuestion: Question | null;
  answers: Map<string, any>;
  timeRemaining: number;
  submitAnswer: (questionId: string, answer: any) => void;
```

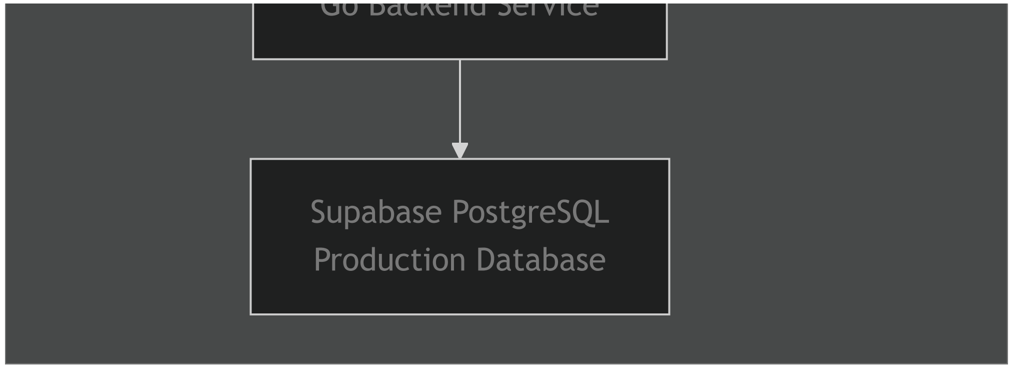
```
    nextQuestion: () => void;
    previousQuestion: () => void;
    submitExam: () => Promise<void>;
  }
```

🚀 DevOps & Deployment Strategy

CI/CD Pipeline Architecture


Development Environment
Frontend Development
Next.js Local Server
Backend Development
Go Local Server
Local PostgreSQL
Development Database
Version Control
GitHub Repository
Source Code Management
Feature Branches
Pull Request Workflow
CI/CD Pipeline
GitHub Actions
Automated Workflows
Build Stage


Build Backend
Go Binary Compilation
Build Frontend
Next.js Production Build
Docker Image Build
Multi-stage Dockerfile
Deploy Stage
Deploy Backend
Hugging Face Spaces
Deploy Frontend
Vercel Platform
Production Environment
Vercel
Next.js Application
Edge Deployment
Hugging Face Spaces
Docker Container



Deployment Configuration

Docker Configuration for Backend:

```
# Gunakan image dasar Golang versi 1.24.1
FROM golang:1.24.1 AS builder

# Set working directory
WORKDIR /app

# Copy go.mod dan go.sum
COPY go.mod go.sum ./

# Download dependencies
RUN go mod download

# Copy seluruh kode
COPY . .

# Buat file .env dengan variabel environment yang dibutuhkan
RUN echo "DB_HOST=aws-0-ap-southeast-1.pooler.supabase.com" >> .env && \
    echo "DB_USER=postgres.yxwraotdmkseklnqrnlp" >> .env && \
    echo "DB_PASSWORD=QUZUU2025" >> .env && \
    echo "DB_PORT=5432" >> .env && \
    echo "DB_NAME=postgres" >> .env && \
    echo "HOST_ADDRESS = 0.0.0.0" >> .env && \
    echo "HOST_PORT = 7860" >> .env && \
    echo "EMAIL_VERIFICATION_DURATION = 2" >> .env

# Build aplikasi
RUN go build -o main .

# Jalankan aplikasi
CMD ["./main"]
```

GitHub Actions Workflow (Backend):

```
name: Deploy to Huggingface
on:
  push:
    branches:
      - master
jobs:
  deploy-to-huggingface:
    runs-on: ubuntu-latest
    steps:
      # Checkout repository
      - name: Checkout Repository
        uses: actions/checkout@v3
      # Setup Git
      - name: Setup Git for Huggingface
        run: |
          git config --global user.email "abdan.hafidz@gmail.com"
          git config --global user.name "abdanhafidz"
      # Clone Huggingface Space Repository
      - name: Clone Huggingface Space
        env:
```

```yaml
        HF_TOKEN: ${{ secrets.HF_TOKEN }}
      run: |
        git clone https://huggingface.co/spaces/lifedebugger/quzuu-api-dev space
    # Update Git Remote URL and Pull Latest Changes
    - name: Update Remote and Pull Changes
      env:
        HF_TOKEN: ${{ secrets.HF_TOKEN }}
      run: |
        cd space
        git remote set-url origin https://lifedebugger:$HF_TOKEN@huggingface.co/spaces/lifedebugger/quzuu-api-dev
        git pull origin main || echo "No changes to pull"
    # Clean Space Directory - Delete all files except .git
    - name: Clean Space Directory
      run: |
        cd space
        find . -mindepth 1 -not -path "./.git*" -delete
    # Copy Files to Huggingface Space
    - name: Copy Files to Space
      run: |
        rsync -av --exclude='.git' ./ space/
    # Commit and Push to Huggingface Space
    - name: Commit and Push to Huggingface
      env:
        HF_TOKEN: ${{ secrets.HF_TOKEN }}
      run: |
        cd space
        git add .
        git commit -m "Deploy files from GitHub repository" || echo "No changes to commit"
        git push origin main || echo "No changes to push"
```

📊 Project Implementation Summary

Technical Implementation

- **Architecture**: Successfully implemented microservice architecture with Next.js frontend and Go backend
- **Database**: PostgreSQL with Supabase hosting and GORM for database operations
- **Authentication**: JWT tokens with Google OAuth 2.0 and email verification
- **Deployment**: Automated CI/CD with GitHub Actions to Hugging Face (backend) and Vercel (frontend)

Key Features Achieved

- **Diverse Question Types**: 8 different question formats including innovative Block Code Puzzles
- **Interactive Programming Assessment**: Drag-and-drop code block interface
- **Secure Authentication**: Multi-method login with OAuth integration
- **Automated Deployment**: Push-to-deploy workflow with Docker containerization

🎓 Academic Learning Outcomes

Web Programming Course Integration

This project successfully demonstrates the application of key web programming concepts taught in the course:

1. Full-Stack Development

- Frontend-backend separation with clear API boundaries
- RESTful API design principles
- Asynchronous programming patterns

2. Database Management

- Relational database design and normalization
- CRUD operations implementation
- ORM usage and best practices
- Database migration and version control

3. Authentication & Authorization

- Session management and security
- OAuth 2.0 implementation
- JWT token-based authentication
- Multi-factor authentication concepts

4. Modern Web Technologies

- TypeScript for type safety
- React hooks and context API
- Server-side rendering with Next.js
- Go microservice architecture

5. DevOps & Deployment

- Containerization with Docker
- CI/CD pipeline implementation
- Cloud platform deployment
- Automated testing and quality assurance

Innovation Beyond Curriculum

The Block Code Puzzle feature represents an innovative extension beyond traditional web programming coursework, demonstrating advanced user interface design and educational technology integration.

📈 Project Outcomes

Educational Learning Objectives Achieved

- **Full-Stack Development**: Complete frontend-backend integration

- **Database Management**: CRUD operations with ORM and auto-migration
- **Modern Authentication**: JWT and OAuth 2.0 implementation
- **DevOps Practices**: CI/CD pipeline with Docker containerization
- **API Development**: RESTful API design and implementation

📝 Conclusion

The Quzuu Interactive Online Examination Platform successfully demonstrates the practical application of web programming concepts learned throughout the course. The project integrates modern technologies including Next.js with TypeScript for the frontend, Go with Gin framework for the backend, and PostgreSQL with Supabase for data management.

Key achievements include implementing a comprehensive microservice architecture, creating an innovative Block Code Puzzle system for programming assessments, establishing secure authentication with both email verification and Google OAuth 2.0, and deploying the application using automated CI/CD pipelines with GitHub Actions.

The platform's standout Block Code Puzzle feature showcases the potential for interactive educational technology, providing students with hands-on programming assessment tools that go beyond traditional multiple-choice questions. Through its modular architecture and automated deployment system, Quzuu represents a production-ready educational platform that successfully fulfills the requirements of the web programming final project while demonstrating mastery of full-stack development principles.