TAB-8 Assessment Platform - ПОЛНЫЙ КОД ВСЕХ ФАЙЛОВ

ЧАСТЬ 1: КОРНЕВАЯ ДИРЕКТОРИЯ

`.env.example`
```bash
# Environment
NODE_ENV=development

# Database
POSTGRES_USER=tab8user
POSTGRES_PASSWORD=tab8pass
POSTGRES_DB=tab8db
DATABASE_URL=postgresql://tab8user:tab8pass@localhost:5432/tab8db

# Redis
REDIS_PASSWORD=tab8redis
REDIS_URL=redis://:tab8redis@localhost:6379

# Security
JWT_SECRET=your-super-secret-jwt-key-change-this
SESSION_SECRET=your-session-secret-change-this
ENCRYPTION_KEY=your-encryption-key-change-this
COOKIE_SECRET=your-cookie-secret-change-this

# Frontend URLs
FRONTEND_URL=http://localhost:3000
NEXT_PUBLIC_API_URL=http://localhost:3001/api
NEXT_PUBLIC_APP_URL=http://localhost:3000
NEXT_PUBLIC_WS_URL=ws://localhost:3001

# Email
SENDGRID_API_KEY=your-sendgrid-api-key
SENDGRID_FROM_EMAIL=noreply@tab8assessment.com

# AWS (for file storage)
AWS_ACCESS_KEY_ID=your-aws-access-key
AWS_SECRET_ACCESS_KEY=your-aws-secret-key
AWS_REGION=us-east-1
AWS_S3_BUCKET=tab8-uploads

# Monitoring
SENTRY_DSN=https://your-sentry-dsn@sentry.io/project-id
GA_ID=G-XXXXXXXXXX

# Grafana
GRAFANA_USER=admin
GRAFANA_PASSWORD=admin

# CORS
ALLOWED_ORIGINS=http://localhost:3000,https://tab8assessment.com

# Rate Limiting
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=100
```

```
# Workers
ENABLE_WORKERS=true
WORKER_CONCURRENCY=5

# Feature Flags
ENABLE_ANALYTICS=true
ENABLE_MONITORING=true
ENABLE_WEBSOCKET=true
```

`.gitignore`
```
# Dependencies
node_modules/
.pnp
.pnp.js

# Testing
coverage/
.nyc_output

# Next.js
.next/
out/
build/

# Production
dist/
build/

# Misc
.DS_Store
*.pem

# Debug
npm-debug.log*
yarn-debug.log*
yarn-error.log*
.pnpm-debug.log*

# Local env files
.env
.env.local
.env.development.local
.env.test.local
.env.production.local

# Vercel
.vercel

# Typescript
*.tsbuildinfo
next-env.d.ts

# Logs
logs/
*.log
```

```
# Data
data/
uploads/
backups/

# IDE
.vscode/
.idea/
*.swp
*.swo

# OS
.DS_Store
Thumbs.db

# Prisma
prisma/migrations/dev/
```

`docker-compose.prod.yml`
```yaml
version: '3.9'

services:
backend:
build:
target: production
environment:
NODE_ENV: production
volumes:
- ./uploads:/app/uploads
- ./logs/backend:/app/logs
deploy:
replicas: 3
restart_policy:
condition: any
delay: 5s
max_attempts: 3
resources:
limits:
cpus: '1'
memory: 1G
reservations:
cpus: '0.5'
memory: 512M

frontend:
build:
target: production
environment:
NODE_ENV: production
deploy:
replicas: 2
restart_policy:
condition: any
delay: 5s
```

max_attempts: 3
resources:
limits:
cpus: '0.5'
memory: 512M

postgres:
volumes:
- postgres_data:/var/lib/postgresql/data
- ./backups:/backups
deploy:
placement:
constraints:
- node.role == manager

redis:
deploy:
replicas: 1
```

### `README.md`
```markdown
# TAB-8 Assessment Platform

Научно обоснованная платформа для оценки талантов и подбора карьеры.

##  Быстрый старт

### Требования
- Docker & Docker Compose
- Node.js 18+
- PostgreSQL 15+
- Redis 7+

### Установка одной командой

```bash
make install
```

Запуск

```bash
# Development
make dev

# Production
make prod
```

Доступ

- Frontend: http://localhost:3000
- Backend API: http://localhost:3001
- Monitoring: http://localhost:3005
- API Docs: http://localhost:3001/api-docs

Основные команды

```bash
make help # Показать все команды
make test # Запустить тесты
make build # Собрать production образы
make deploy # Развернуть на сервере
make backup # Создать резервную копию БД
make logs # Показать логи
make monitor # Открыть мониторинг
```

🏛️ Архитектура

- **Frontend**: Next.js 14, React 18, TypeScript, Tailwind CSS
- **Backend**: Node.js, Express, Prisma ORM, TypeScript
- **Database**: PostgreSQL 15 с индексами и партиционированием
- **Cache**: Redis для сессий и кеширования
- **Queue**: Bull для фоновых задач
- **Monitoring**: Prometheus + Grafana
- **Deployment**: Docker, Kubernetes ready

Безопасность

- JWT авторизация
- Rate limiting
- CSRF защита
- XSS защита
- SQL injection защита
- Шифрование данных
- Двухфакторная аутентификация
- Аудит логи

🧪 Тестирование

```bash
# Backend тесты
cd backend && npm test

# Frontend тесты
cd frontend && npm test

# E2E тесты
npm run test:e2e
```

Мониторинг

- Prometheus метрики: http://localhost:9090
- Grafana дашборды: http://localhost:3005
- Health check: http://localhost:3001/health
- Метрики: http://localhost:3001/metrics

Production деплой

1. Настройте переменные окружения

2. Соберите образы: `make build`
3. Запустите: `make prod`
4. Настройте SSL сертификаты
5. Настройте мониторинг

Лицензия

MIT License - см. LICENSE файл

Контрибьюция

1. Fork репозитория
2. Создайте feature branch
3. Commit изменения
4. Push в branch
5. Создайте Pull Request
```

## ЧАСТЬ 2: BACKEND

### `backend/Dockerfile`
```dockerfile
# Base stage
FROM node:18-alpine AS base
RUN apk add --no-cache libc6-compat
WORKDIR /app

# Dependencies stage
FROM base AS deps
COPY package*.json ./
RUN npm ci

# Build stage
FROM base AS builder
COPY --from=deps /app/node_modules ./node_modules
COPY . .
RUN npx prisma generate
RUN npm run build

# Development stage
FROM base AS development
ENV NODE_ENV=development
COPY --from=deps /app/node_modules ./node_modules
COPY . .
RUN npx prisma generate
EXPOSE 3001
CMD ["npm", "run", "dev"]

# Production stage
FROM base AS production
ENV NODE_ENV=production
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nodejs -u 1001

COPY --from=builder --chown=nodejs:nodejs /app/dist ./dist
COPY --from=builder --chown=nodejs:nodejs /app/node_modules ./node_modules
```
```

```
COPY --from=builder --chown=nodejs:nodejs /app/package*.json ./
COPY --from=builder --chown=nodejs:nodejs /app/prisma ./prisma

USER nodejs
EXPOSE 3001
CMD ["node", "dist/server.js"]
```

`backend/tsconfig.json`
```json
{
"compilerOptions": {
"target": "ES2022",
"module": "commonjs",
"lib": ["ES2022"],
"outDir": "./dist",
"rootDir": "./src",
"strict": true,
"esModuleInterop": true,
"skipLibCheck": true,
"forceConsistentCasingInFileNames": true,
"resolveJsonModule": true,
"moduleResolution": "node",
"allowJs": true,
"noUnusedLocals": true,
"noUnusedParameters": true,
"noImplicitReturns": true,
"noFallthroughCasesInSwitch": true,
"allowSyntheticDefaultImports": true,
"emitDecoratorMetadata": true,
"experimentalDecorators": true,
"sourceMap": true,
"incremental": true,
"typeRoots": ["./node_modules/@types", "./src/types"],
"baseUrl": ".",
"paths": {
"@/*": ["src/*"],
"@config/*": ["src/config/*"],
"@controllers/*": ["src/controllers/*"],
"@services/*": ["src/services/*"],
"@middleware/*": ["src/middleware/*"],
"@utils/*": ["src/utils/*"],
"@types/*": ["src/types/*"]
}
},
"include": ["src/**/*"],
"exclude": ["node_modules", "dist", "tests"]
}
```

`backend/.eslintrc.js`
```javascript
module.exports = {
parser: '@typescript-eslint/parser',
parserOptions: {
project: 'tsconfig.json',
tsconfigRootDir: __dirname,
```

sourceType: 'module',
},
plugins: ['@typescript-eslint/eslint-plugin'],
extends: [
'plugin:@typescript-eslint/recommended',
'plugin:prettier/recommended',
],
root: true,
env: {
node: true,
jest: true,
},
ignorePatterns: ['.eslintrc.js', 'dist', 'node_modules'],
rules: {
'@typescript-eslint/interface-name-prefix': 'off',
'@typescript-eslint/explicit-function-return-type': 'off',
'@typescript-eslint/explicit-module-boundary-types': 'off',
'@typescript-eslint/no-explicit-any': 'off',
'@typescript-eslint/no-unused-vars': ['error', { argsIgnorePattern: '^_' }],
'prettier/prettier': ['error', { endOfLine: 'auto' }],
},
};
```

### `backend/jest.config.js`
```javascript
module.exports = {
preset: 'ts-jest',
testEnvironment: 'node',
roots: ['<rootDir>/src', '<rootDir>/tests'],
testMatch: ['**/__tests__/**/*.ts', '**/?(*.)+(spec|test).ts'],
transform: {
'^.+\\.ts$': 'ts-jest',
},
collectCoverageFrom: [
'src/**/*.ts',
'!src/**/*.d.ts',
'!src/**/*.spec.ts',
'!src/**/*.test.ts',
],
coverageDirectory: 'coverage',
coverageReporters: ['text', 'lcov', 'html'],
moduleNameMapper: {
'^@/(.*)$': '<rootDir>/src/$1',
'^@config/(.*)$': '<rootDir>/src/config/$1',
'^@controllers/(.*)$': '<rootDir>/src/controllers/$1',
'^@services/(.*)$': '<rootDir>/src/services/$1',
'^@middleware/(.*)$': '<rootDir>/src/middleware/$1',
'^@utils/(.*)$': '<rootDir>/src/utils/$1',
'^@types/(.*)$': '<rootDir>/src/types/$1',
},
setupFilesAfterEnv: ['<rootDir>/tests/setup.ts'],
testTimeout: 10000,
};
```

`backend/prisma/schema.prisma`
```

```prisma
generator client {
provider = "prisma-client-js"
}

datasource db {
provider = "postgresql"
url = env("DATABASE_URL")
}

model User {
id String @id @default(uuid())
email String @unique
passwordHash String?
name String?
age Int?
gender String?
education String?
country String?
emailVerified Boolean @default(false)
subscriptionTier String @default("free")
mfaSecret String?
mfaEnabled Boolean @default(false)
createdAt DateTime @default(now())
updatedAt DateTime @updatedAt

testSessions TestSession[]
auditLogs AuditLog[]

@@index([email])
@@map("users")
}

model TestSession {
id String @id @default(uuid())
userId String
user User @relation(fields: [userId], references: [id])
startedAt DateTime @default(now())
completedAt DateTime?
ipAddress String?
userAgent String?
testVersion String @default("1.0")
status String @default("in_progress")
metadata Json?

responses Response[]
result Result?

@@index([userId, status])
@@index([startedAt])
@@map("test_sessions")
}

model Response {
id BigInt @id @default(autoincrement())
sessionId String
session TestSession @relation(fields: [sessionId], references: [id], onDelete: Cascade)
```

```
module String
questionId Int
answer String
timeSpent Int? // в секундах
createdAt DateTime @default(now())

@@unique([sessionId, module, questionId])
@@index([sessionId])
@@map("responses")
}

model Result {
id String @id @default(uuid())
sessionId String @unique
session TestSession @relation(fields: [sessionId], references: [id], onDelete: Cascade)

// Модуль 1: Когнитивные
cognitiveRaw Int
cognitivePercentile Int
iqEquivalent Int
verbalScore Int
numericalScore Int
abstractScore Int

// Модуль 2: RIASEC
realistic Int
investigative Int
artistic Int
social Int
enterprising Int
conventional Int
riasecCode String

// Модуль 3: Big Five
openness Float
conscientiousness Float
extraversion Float
agreeableness Float
neuroticism Float

// Модуль 4: Карьерные якоря
technical Int
managerial Int
autonomy Int
security Int
entrepreneurial Int
service Int
challenge Int
lifestyle Int
topAnchors String[]

// Рекомендации
recommendedCareers Json
matchScores Json
developmentAreas String[]

createdAt DateTime @default(now())
```

```
@@index([createdAt])
@@map("results")
}

model AuditLog {
id BigInt @id @default(autoincrement())
userId String?
user User? @relation(fields: [userId], references: [id])
action String
entity String?
entityId String?
metadata Json?
ipAddress String?
userAgent String?
createdAt DateTime @default(now())

@@index([userId, action])
@@index([createdAt])
@@map("audit_logs")
}

model Question {
id Int @id @default(autoincrement())
module String
type String
category String?
text String
options Json?
correct String?
metadata Json?
active Boolean @default(true)
version String @default("1.0")

@@index([module, active])
@@map("questions")
}

model Career {
id Int @id @default(autoincrement())
title String
description String
riasecCode String
minIQ Int
maxIQ Int
idealPersonality Json
requiredAnchors String[]
skills String[]
salary String
growthPath String
education String
active Boolean @default(true)

@@index([riasecCode])
@@index([active])
@@map("careers")
}
```

```

### `backend/prisma/seed.ts`
```typescript
import { PrismaClient } from '@prisma/client';
import { hash } from 'bcryptjs';
import { questions } from '../src/data/questions';
import { careers } from '../src/data/careers';

const prisma = new PrismaClient();

async function main() {
console.log('      Seeding database...');

// Create test users
const testUser = await prisma.user.upsert({
where: { email: 'test@example.com' },
update: {},
create: {
email: 'test@example.com',
passwordHash: await hash('Test123456', 10),
name: 'Test User',
emailVerified: true,
},
});

const adminUser = await prisma.user.upsert({
where: { email: 'admin@tab8.com' },
update: {},
create: {
email: 'admin@tab8.com',
passwordHash: await hash('Admin123456', 10),
name: 'Admin User',
emailVerified: true,
subscriptionTier: 'admin',
},
});

console.log('✅ Users created');

// Seed questions
const allQuestions = [
...questions.cognitive.verbal,
...questions.cognitive.numerical,
...questions.cognitive.abstract,
...questions.interests,
...questions.personality,
...questions.values,
];

for (const question of allQuestions) {
await prisma.question.upsert({
where: { id: question.id },
update: {},
create: {
id: question.id,
```
```

```
module: question.module || getModuleFromId(question.id),
type: question.type || getTypeFromId(question.id),
category: question.category,
text: question.text,
options: question.options,
correct: question.correct,
metadata: question.metadata || {},
},
});
}

console.log('✅ Questions seeded');

// Seed careers
for (const career of careers) {
await prisma.career.upsert({
where: { id: career.id },
update: {},
create: {
id: career.id,
title: career.title,
description: career.description,
riasecCode: career.riasecCode,
minIQ: career.minIQ,
maxIQ: career.maxIQ,
idealPersonality: career.idealPersonality,
requiredAnchors: career.requiredAnchors,
skills: career.skills,
salary: career.salary,
growthPath: career.growthPath,
education: career.education,
},
});
}

console.log('✅ Careers seeded');
console.log('      Seeding completed!');
}

function getModuleFromId(id: number): string {
if (id <= 30) return 'cognitive';
if (id <= 78) return 'interests';
if (id <= 118) return 'personality';
return 'values';
}

function getTypeFromId(id: number): string {
if (id <= 10) return 'verbal';
if (id <= 20) return 'numerical';
if (id <= 30) return 'abstract';
if (id <= 78) return 'likert';
if (id <= 118) return 'likert';
return 'pairs';
}

main()
```

```
.catch((e) => {
console.error(e);
process.exit(1);
})
.finally(async () => {
await prisma.$disconnect();
});
```

`backend/src/app.ts`
```typescript
import express, { Application } from 'express';
import 'express-async-errors';
import { setupMiddleware, setupErrorHandling } from './middleware';
import { setupRoutes } from './routes';
import { MetricsCollector } from './monitoring/metrics';

export function createApp(): Application {
const app = express();
const metrics = new MetricsCollector();

// Setup middleware
setupMiddleware(app, metrics);

// Setup routes
setupRoutes(app);

// Setup error handling
setupErrorHandling(app);

return app;
}
```

`backend/src/server.ts`
```typescript
import 'reflect-metadata';
import dotenv from 'dotenv';
dotenv.config();

import { createServer } from 'http';
import { Server } from 'socket.io';
import { PrismaClient } from '@prisma/client';
import Redis from 'ioredis';
import * as Sentry from '@sentry/node';
import { createApp } from './app';
import { logger } from './utils/logger';
import { setupWebSocket } from './websocket';
import { setupWorkers } from './workers';
import { HealthChecker } from './monitoring/health';

// Initialize app
const app = createApp();
const server = createServer(app);
const io = new Server(server, {
cors: {
origin: process.env.FRONTEND_URL,
```

```
credentials: true,
},
});

// Database connections
export const prisma = new PrismaClient({
log: ['query', 'error', 'warn'],
errorFormat: 'pretty',
});

export const redis = new Redis({
host: process.env.REDIS_HOST || 'redis',
port: parseInt(process.env.REDIS_PORT || '6379'),
password: process.env.REDIS_PASSWORD,
retryStrategy: (times: number) => {
const delay = Math.min(times * 50, 2000);
return delay;
},
});

// Health checker
const healthChecker = new HealthChecker(prisma, redis);

// Sentry initialization
if (process.env.SENTRY_DSN) {
Sentry.init({
dsn: process.env.SENTRY_DSN,
integrations: [
new Sentry.Integrations.Http({ tracing: true }),
new Sentry.Integrations.Express({ app }),
new Sentry.Integrations.Prisma({ client: prisma }),
],
tracesSampleRate: process.env.NODE_ENV === 'production' ? 0.1 : 1.0,
environment: process.env.NODE_ENV,
});
}

// Setup application
async function bootstrap() {
try {
// Test database connection
await prisma.$connect();
logger.info('✓ Database connected');

// Test Redis connection
await redis.ping();
logger.info('✓ Redis connected');

// Setup WebSocket
setupWebSocket(io);

// Setup background workers
if (process.env.ENABLE_WORKERS !== 'false') {
await setupWorkers();
}
```

```
// Health check endpoints
app.get('/health', async (req, res) => {
const health = await healthChecker.check();
res.status(health.status === 'healthy' ? 200 : 503).json(health);
});

// Start server
const PORT = process.env.PORT || 3001;
server.listen(PORT, () => {
logger.info(`      Server running on port ${PORT}`);
logger.info(`    WebSocket ready`);
logger.info(`      Environment: ${process.env.NODE_ENV}`);
});

// Graceful shutdown
const gracefulShutdown = async () => {
logger.info('Shutting down gracefully...');

server.close(() => {
logger.info('HTTP server closed');
});

await prisma.$disconnect();
await redis.quit();

process.exit(0);
};

process.on('SIGTERM', gracefulShutdown);
process.on('SIGINT', gracefulShutdown);
} catch (error) {
logger.error('Failed to start server:', error);
process.exit(1);
}
}

// Handle unhandled errors
process.on('unhandledRejection', (reason, promise) => {
logger.error('Unhandled Rejection at:', promise, 'reason:', reason);
Sentry.captureException(reason);
});

process.on('uncaughtException', (error) => {
logger.error('Uncaught Exception:', error);
Sentry.captureException(error);
process.exit(1);
});

// Start application
bootstrap();
```


`backend/src/routes/index.ts`
```typescript
import { Application } from 'express';
import authRoutes from './auth.routes';
```

```
import testRoutes from './test.routes';
import resultsRoutes from './results.routes';
import userRoutes from './user.routes';
import adminRoutes from './admin.routes';

export function setupRoutes(app: Application): void {
// API routes
app.use('/api/auth', authRoutes);
app.use('/api/test', testRoutes);
app.use('/api/results', resultsRoutes);
app.use('/api/users', userRoutes);
app.use('/api/admin', adminRoutes);

// API documentation
if (process.env.NODE_ENV !== 'production') {
const swaggerUi = require('swagger-ui-express');
const swaggerDocument = require('../docs/swagger.json');
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
}

// Root endpoint
app.get('/', (req, res) => {
res.json({
name: 'TAB-8 Assessment API',
version: '1.0.0',
status: 'running',
timestamp: new Date().toISOString(),
});
});
}
```

`backend/src/routes/auth.routes.ts`
```typescript
import { Router } from 'express';
import { AuthController } from '../controllers/auth.controller';
import { validateRequest } from '../middleware/validation';
import { registerSchema, loginSchema, resetPasswordSchema } from '../schemas/auth.schema';

const router = Router();
const authController = new AuthController();

// Public routes
router.post('/register', validateRequest(registerSchema), authController.register);
router.post('/login', validateRequest(loginSchema), authController.login);
router.post('/refresh', authController.refreshToken);
router.post('/forgot-password', authController.forgotPassword);
router.post('/reset-password', validateRequest(resetPasswordSchema), authController.resetPassword);
router.get('/verify-email/:token', authController.verifyEmail);

// Protected routes
router.post('/logout', authController.logout);
router.post('/change-password', authController.changePassword);
router.get('/me', authController.getCurrentUser);
router.post('/enable-2fa', authController.enable2FA);
router.post('/verify-2fa', authController.verify2FA);
```

```
export default router;
```

`backend/src/routes/test.routes.ts`
```typescript
import { Router } from 'express';
import { TestController } from '../controllers/test.controller';
import { authenticateToken } from '../middleware/auth';
import { validateRequest } from '../middleware/validation';
import { startTestSchema, submitResponseSchema } from '../schemas/test.schema';

const router = Router();
const testController = new TestController();

// All routes require authentication
router.use(authenticateToken);

// Test session management
router.post('/start', validateRequest(startTestSchema), testController.startTest);
router.post('/response', validateRequest(submitResponseSchema), testController.submitResponse);
router.post('/complete/:sessionId', testController.completeTest);
router.get('/progress/:sessionId', testController.getProgress);
router.post('/resume', testController.resumeTest);

// Question management
router.get('/questions/:module', testController.getModuleQuestions);
router.get('/question/:module/:questionId', testController.getQuestion);

// Test status
router.get('/active', testController.getActiveSession);
router.get('/history', testController.getTestHistory);

export default router;
```

`backend/src/controllers/auth.controller.ts`
```typescript
import { Request, Response } from 'express';
import { PrismaClient } from '@prisma/client';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import { authenticator } from 'otplib';
import QRCode from 'qrcode';
import { AuthService } from '../services/auth.service';
import { EmailService } from '../services/email.service';
import { logger } from '../utils/logger';
import { AppError } from '../utils/errors';

const prisma = new PrismaClient();
const authService = new AuthService(prisma);
const emailService = new EmailService();

export class AuthController {
async register(req: Request, res: Response) {
try {
const { email, password, name, age, gender, education, country } = req.body;
```

```
// Check if user exists
const existingUser = await prisma.user.findUnique({
where: { email },
});

if (existingUser) {
throw new AppError('Email already registered', 400);
}

// Hash password
const passwordHash = await bcrypt.hash(password, 12);

// Create user
const user = await prisma.user.create({
data: {
email,
passwordHash,
name,
age,
gender,
education,
country,
},
});

// Send verification email
const verificationToken = authService.generateVerificationToken(user.id);
await emailService.sendVerificationEmail(email, verificationToken);

// Generate tokens
const accessToken = authService.generateAccessToken(user);
const refreshToken = authService.generateRefreshToken(user);

// Save refresh token
await authService.saveRefreshToken(user.id, refreshToken);

// Log audit
await prisma.auditLog.create({
data: {
userId: user.id,
action: 'USER_REGISTERED',
entity: 'User',
entityId: user.id,
ipAddress: req.ip,
userAgent: req.get('user-agent'),
},
});

res.status(201).json({
message: 'Registration successful. Please verify your email.',
user: {
id: user.id,
email: user.email,
name: user.name,
},
tokens: {
accessToken,
```

```
refreshToken,
},
});
} catch (error) {
logger.error('Registration error:', error);
throw error;
}
}

async login(req: Request, res: Response) {
try {
const { email, password, totpCode } = req.body;

// Find user
const user = await prisma.user.findUnique({
where: { email },
});

if (!user) {
throw new AppError('Invalid credentials', 401);
}

// Verify password
const isValidPassword = await bcrypt.compare(password, user.passwordHash);
if (!isValidPassword) {
throw new AppError('Invalid credentials', 401);
}

// Check email verification
if (!user.emailVerified) {
throw new AppError('Please verify your email first', 403);
}

// Check 2FA
if (user.mfaEnabled) {
if (!totpCode) {
return res.status(200).json({
requiresTwoFactor: true,
message: 'Please provide 2FA code',
});
}

const isValid = authenticator.verify({
token: totpCode,
secret: user.mfaSecret,
});

if (!isValid) {
throw new AppError('Invalid 2FA code', 401);
}
}

// Generate tokens
const accessToken = authService.generateAccessToken(user);
const refreshToken = authService.generateRefreshToken(user);

// Save refresh token
```

```
await authService.saveRefreshToken(user.id, refreshToken);

// Update last login
await prisma.user.update({
where: { id: user.id },
data: { updatedAt: new Date() },
});

// Log audit
await prisma.auditLog.create({
data: {
userId: user.id,
action: 'USER_LOGIN',
entity: 'User',
entityId: user.id,
ipAddress: req.ip,
userAgent: req.get('user-agent'),
},
});

res.json({
user: {
id: user.id,
email: user.email,
name: user.name,
subscriptionTier: user.subscriptionTier,
},
tokens: {
accessToken,
refreshToken,
},
});
} catch (error) {
logger.error('Login error:', error);
throw error;
}
}

async refreshToken(req: Request, res: Response) {
try {
const { refreshToken } = req.body;

if (!refreshToken) {
throw new AppError('Refresh token required', 400);
}

// Verify refresh token
const decoded = jwt.verify(
refreshToken,
process.env.JWT_REFRESH_SECRET
) as any;

// Check if token exists in database
const isValid = await authService.validateRefreshToken(
decoded.userId,
refreshToken
);
```

```
if (!isValid) {
throw new AppError('Invalid refresh token', 401);
}

// Get user
const user = await prisma.user.findUnique({
where: { id: decoded.userId },
});

if (!user) {
throw new AppError('User not found', 404);
}

// Generate new tokens
const newAccessToken = authService.generateAccessToken(user);
const newRefreshToken = authService.generateRefreshToken(user);

// Update refresh token
await authService.updateRefreshToken(user.id, refreshToken, newRefreshToken);

res.json({
tokens: {
accessToken: newAccessToken,
refreshToken: newRefreshToken,
},
});
} catch (error) {
logger.error('Refresh token error:', error);
throw error;
}
}

async logout(req: Request, res: Response) {
try {
const userId = req.user.id;
const { refreshToken } = req.body;

if (refreshToken) {
await authService.revokeRefreshToken(userId, refreshToken);
}

// Log audit
await prisma.auditLog.create({
data: {
userId,
action: 'USER_LOGOUT',
entity: 'User',
entityId: userId,
ipAddress: req.ip,
userAgent: req.get('user-agent'),
},
});

res.json({ message: 'Logout successful' });
} catch (error) {
logger.error('Logout error:', error);
```

```
throw error;
}
}

async forgotPassword(req: Request, res: Response) {
try {
const { email } = req.body;

const user = await prisma.user.findUnique({
where: { email },
});

if (!user) {
// Don't reveal if user exists
return res.json({
message: 'If the email exists, a reset link has been sent.',
});
}

// Generate reset token
const resetToken = authService.generatePasswordResetToken(user.id);

// Send email
await emailService.sendPasswordResetEmail(email, resetToken);

res.json({
message: 'If the email exists, a reset link has been sent.',
});
} catch (error) {
logger.error('Forgot password error:', error);
throw error;
}
}

async resetPassword(req: Request, res: Response) {
try {
const { token, newPassword } = req.body;

// Verify token
const decoded = jwt.verify(
token,
process.env.JWT_SECRET
) as any;

if (decoded.type !== 'password_reset') {
throw new AppError('Invalid token', 400);
}

// Hash new password
const passwordHash = await bcrypt.hash(newPassword, 12);

// Update password
await prisma.user.update({
where: { id: decoded.userId },
data: { passwordHash },
});
```

```
// Log audit
await prisma.auditLog.create({
data: {
userId: decoded.userId,
action: 'PASSWORD_RESET',
entity: 'User',
entityId: decoded.userId,
ipAddress: req.ip,
userAgent: req.get('user-agent'),
},
});

res.json({ message: 'Password reset successful' });
} catch (error) {
logger.error('Reset password error:', error);
throw error;
}
}

async verifyEmail(req: Request, res: Response) {
try {
const { token } = req.params;

// Verify token
const decoded = jwt.verify(
token,
process.env.JWT_SECRET
) as any;

if (decoded.type !== 'email_verification') {
throw new AppError('Invalid token', 400);
}

// Update user
await prisma.user.update({
where: { id: decoded.userId },
data: { emailVerified: true },
});

res.json({ message: 'Email verified successfully' });
} catch (error) {
logger.error('Email verification error:', error);
throw error;
}
}

async changePassword(req: Request, res: Response) {
try {
const userId = req.user.id;
const { currentPassword, newPassword } = req.body;

// Get user
const user = await prisma.user.findUnique({
where: { id: userId },
});

// Verify current password
```

```
const isValid = await bcrypt.compare(currentPassword, user.passwordHash);
if (!isValid) {
throw new AppError('Current password is incorrect', 400);
}

// Hash new password
const passwordHash = await bcrypt.hash(newPassword, 12);

// Update password
await prisma.user.update({
where: { id: userId },
data: { passwordHash },
});

res.json({ message: 'Password changed successfully' });
} catch (error) {
logger.error('Change password error:', error);
throw error;
}
}

async getCurrentUser(req: Request, res: Response) {
try {
const userId = req.user.id;

const user = await prisma.user.findUnique({
where: { id: userId },
select: {
id: true,
email: true,
name: true,
age: true,
gender: true,
education: true,
country: true,
subscriptionTier: true,
emailVerified: true,
mfaEnabled: true,
createdAt: true,
},
});

res.json({ user });
} catch (error) {
logger.error('Get current user error:', error);
throw error;
}
}

async enable2FA(req: Request, res: Response) {
try {
const userId = req.user.id;

// Generate secret
const secret = authenticator.generateSecret();

// Generate QR code
```