Implementation Details

Gym Management System

---

1. TECHNOLOGY STACK

1.1 Backend Technologies

- **Runtime**: Node.js v18+
- **Framework**: Express.js 4.18.2
- **Language**: TypeScript 5.3.3
- **Database**: PostgreSQL 12+ (Neon.tech)
- **ORM**: Prisma 5.7.1
- **Authentication**: JWT (jsonwebtoken 9.0.2)
- **Password Hashing**: bcryptjs 2.4.3
- **Validation**: express-validator

1.2 Frontend Technologies

- **Framework**: React 19
- **Build Tool**: Vite 7.2.4
- **Language**: TypeScript
- **Styling**: Tailwind CSS 4.1.17
- **HTTP Client**: Fetch API

1.3 Deployment

- **Backend**: Vercel (Serverless)
- **Frontend**: Vercel
- **Database**: Neon.tech (Cloud PostgreSQL)

---

2. PROJECT STRUCTURE

```
GymManagement/
│
├── backend/
```

```
│   ├── prisma/
│   │   ├── schema.prisma           # Database schema
│   │   ├── seed.ts                 # Database seeding
│   │   └── migrations/             # Migration files
│   │
│   ├── src/
│   │   ├── config/
│   │   │   └── db.ts               # Database configuration
│   │   │
│   │   ├── utils/
│   │   │   ├── hash.ts             # Password hashing utilities
│   │   │   └── jwt.ts              # JWT token utilities
│   │   │
│   │   ├── middleware/
│   │   │   ├── auth.ts             # Authentication middleware
│   │   │   └── roleGuard.ts        # Role-based access control
│   │   │
│   │   ├── modules/
│   │   │   ├── auth/
│   │   │   │   ├── auth.controller.ts
│   │   │   │   ├── auth.service.ts
│   │   │   │   └── auth.routes.ts
│   │   │   │
│   │   │   ├── member/
│   │   │   │   ├── member.controller.ts
│   │   │   │   ├── member.service.ts
│   │   │   │   └── member.routes.ts
│   │   │   │
│   │   │   └── trainer/
│   │   │       ├── trainer.controller.ts
│   │   │       ├── trainer.service.ts
│   │   │       └── trainer.routes.ts
│   │   │
│   │   ├── app.ts                  # Express app setup
│   │   └── serverless.ts           # Vercel serverless entry
│   │
│   ├── package.json
│   ├── tsconfig.json
│   ├── vercel.json
│   └── .env                        # Environment variables
│
├── frontend/
│   ├── src/
│   │   ├── App.tsx                 # Main React component
│   │   ├── main.tsx                # Entry point
│   │   └── index.css               # Tailwind CSS
│   │
│   ├── package.json
│   ├── vite.config.ts
│   ├── tsconfig.json
│   └── tailwind.config.js
│
└── docs/
```

```
    ├── 1_SRS_Document.md
    ├── 2_ER_Diagrams.md
    ├── 3_UML_Diagrams.md
    ├── 4_Relational_Schemas.md
    ├── 5_Implementation.md
    └── 6_Test_Cases.md
```

---

3. DATABASE IMPLEMENTATION

3.1 Prisma Schema (schema.prisma)

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

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

model Member {
  member_id      String          @id @default(uuid())
  name           String
  email          String          @unique
  password       String
  age            Int
  gender         String
  phone          String
  join_date      DateTime        @default(now())
  status         String          @default("active")

  workoutPlans   WorkoutPlan[]
  dietPlans      DietPlan[]
  attendances    Attendance[]
  progress       Progress[]

  @@map("members")
}

model Trainer {
  trainer_id      String          @id @default(uuid())
  name            String
  email           String          @unique
  password        String
  specialization  String

  workoutPlans    WorkoutPlan[]
  dietPlans       DietPlan[]
```

```
  progress        Progress[]

  @@map("trainers")
}

model WorkoutPlan {
  plan_id         String    @id @default(uuid())
  member_id       String
  trainer_id      String
  plan_details    String
  created_at      DateTime  @default(now())

  member          Member    @relation(fields: [member_id], references: [member_id], onDelet
  trainer         Trainer   @relation(fields: [trainer_id], references: [trainer_id], onDel

  @@map("workout_plans")
}

model DietPlan {
  diet_id         String    @id @default(uuid())
  member_id       String
  trainer_id      String
  diet_details    String
  created_at      DateTime  @default(now())

  member          Member    @relation(fields: [member_id], references: [member_id], onDelet
  trainer         Trainer   @relation(fields: [trainer_id], references: [trainer_id], onDel

  @@map("diet_plans")
}

model Attendance {
  attendance_id   String    @id @default(uuid())
  member_id       String
  date            DateTime  @default(now())
  status          String

  member          Member    @relation(fields: [member_id], references: [member_id], onDelet

  @@map("attendances")
}

model Progress {
  progress_id     String    @id @default(uuid())
  member_id       String
  trainer_id      String
  weight          Float
  body_fat        Float
  muscle_mass     Float
  notes           String?
  updated_at      DateTime  @default(now())

  member          Member    @relation(fields: [member_id], references: [member_id], onDelet
```

```
  trainer        Trainer   @relation(fields: [trainer_id], references: [trainer_id], onDel

  @@map("progress")
}
```

3.2 Database Seeding (seed.ts)

```
import { PrismaClient } from '@prisma/client';
import { hashPassword } from '../src/utils/hash';

const prisma = new PrismaClient();

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

  // Create trainers
  const trainer1 = await prisma.trainer.create({
    data: {
      name: 'John Smith',
      email: 'john@gym.com',
      password: await hashPassword('trainer123'),
      specialization: 'Strength Training',
    },
  });

  const trainer2 = await prisma.trainer.create({
    data: {
      name: 'Sarah Johnson',
      email: 'sarah@gym.com',
      password: await hashPassword('trainer123'),
      specialization: 'Cardio & Weight Loss',
    },
  });

  console.log(' Created trainers:', { trainer1, trainer2 });
  console.log(' Seeding completed!');
}

main()
  .catch((e) => {
    console.error(' Seeding failed:', e);
    process.exit(1);
  })
  .finally(async () => {
    await prisma.$disconnect();
  });
```

4. AUTHENTICATION IMPLEMENTATION

4.1 Password Hashing (utils/hash.ts)

```
import bcrypt from 'bcryptjs';

const SALT_ROUNDS = 10;

export const hashPassword = async (password: string): Promise<string> => {
  return await bcrypt.hash(password, SALT_ROUNDS);
};

export const comparePassword = async (
  password: string,
  hash: string
): Promise<boolean> => {
  return await bcrypt.compare(password, hash);
};
```

4.2 JWT Token Generation (utils/jwt.ts)

```
import jwt from 'jsonwebtoken';

export interface TokenPayload {
  userId: string;
  role: 'member' | 'trainer';
}

export const generateAccessToken = (payload: TokenPayload): string => {
  const secret = process.env.JWT_ACCESS_SECRET || 'default-secret-key';
  const expiresIn = process.env.JWT_ACCESS_EXPIRY || '15m';
  return jwt.sign(payload, secret, { expiresIn });
};

export const generateRefreshToken = (payload: TokenPayload): string => {
  const secret = process.env.JWT_REFRESH_SECRET || 'default-refresh-secret';
  const expiresIn = process.env.JWT_REFRESH_EXPIRY || '7d';
  return jwt.sign(payload, secret, { expiresIn });
};

export const verifyAccessToken = (token: string): TokenPayload => {
  const secret = process.env.JWT_ACCESS_SECRET || 'default-secret-key';
  return jwt.verify(token, secret) as TokenPayload;
};

export const verifyRefreshToken = (token: string): TokenPayload => {
  const secret = process.env.JWT_REFRESH_SECRET || 'default-refresh-secret';
```

```
  return jwt.verify(token, secret) as TokenPayload;
};
```

4.3 Authentication Middleware (middleware/auth.ts)

```
import { Request, Response, NextFunction } from 'express';
import { verifyAccessToken } from '../utils/jwt';

export interface AuthRequest extends Request {
  user?: {
    userId: string;
    role: 'member' | 'trainer';
  };
}

export const authenticate = (
  req: AuthRequest,
  res: Response,
  next: NextFunction
): void => {
  try {
    const authHeader = req.headers.authorization;

    if (!authHeader || !authHeader.startsWith('Bearer ')) {
      res.status(401).json({ error: 'No token provided' });
      return;
    }

    const token = authHeader.substring(7);
    const decoded = verifyAccessToken(token);

    req.user = decoded;
    next();
  } catch (error) {
    res.status(401).json({ error: 'Invalid or expired token' });
  }
};
```

4.4 Role-Based Access Control (middleware/roleGuard.ts)

```
import { Response, NextFunction } from 'express';
import { AuthRequest } from './auth';

export const requireRole = (allowedRole: 'member' | 'trainer') => {
  return (req: AuthRequest, res: Response, next: NextFunction): void => {
    if (!req.user) {
      res.status(401).json({ error: 'Authentication required' });
      return;
    }
```

```
    if (req.user.role !== allowedRole) {
      res.status(403).json({ error: 'Insufficient permissions' });
      return;
    }

    next();
  };
};
```

---

5. API IMPLEMENTATION

5.1 Authentication Module

Auth Service (modules/auth/auth.service.ts)

```
import { PrismaClient } from '@prisma/client';
import { hashPassword, comparePassword } from '../../utils/hash';
import { generateAccessToken, generateRefreshToken } from '../../utils/jwt';

const prisma = new PrismaClient();

export const registerMember = async (data: {
  name: string;
  email: string;
  password: string;
  age: number;
  gender: string;
  phone: string;
}) => {
  const hashedPassword = await hashPassword(data.password);

  const member = await prisma.member.create({
    data: {
      ...data,
      password: hashedPassword,
    },
  });

  const accessToken = generateAccessToken({
    userId: member.member_id,
    role: 'member',
  });

  const refreshToken = generateRefreshToken({
    userId: member.member_id,
    role: 'member',
  });
```

```
  return { member, accessToken, refreshToken };
};

export const loginMember = async (email: string, password: string) => {
  const member = await prisma.member.findUnique({ where: { email } });

  if (!member) {
    throw new Error('Invalid credentials');
  }

  const isValidPassword = await comparePassword(password, member.password);

  if (!isValidPassword) {
    throw new Error('Invalid credentials');
  }

  const accessToken = generateAccessToken({
    userId: member.member_id,
    role: 'member',
  });

  const refreshToken = generateRefreshToken({
    userId: member.member_id,
    role: 'member',
  });

  return { member, accessToken, refreshToken };
};

export const loginTrainer = async (email: string, password: string) => {
  const trainer = await prisma.trainer.findUnique({ where: { email } });

  if (!trainer) {
    throw new Error('Invalid credentials');
  }

  const isValidPassword = await comparePassword(password, trainer.password);

  if (!isValidPassword) {
    throw new Error('Invalid credentials');
  }

  const accessToken = generateAccessToken({
    userId: trainer.trainer_id,
    role: 'trainer',
  });

  const refreshToken = generateRefreshToken({
    userId: trainer.trainer_id,
    role: 'trainer',
  });
```

```
  return { trainer, accessToken, refreshToken };
};
```

Auth Controller (modules/auth/auth.controller.ts)

```
import { Request, Response } from 'express';
import * as authService from './auth.service';

export const signup = async (req: Request, res: Response): Promise<void> => {
  try {
    const { name, email, password, age, gender, phone } = req.body;
    const result = await authService.registerMember({
      name,
      email,
      password,
      age: parseInt(age),
      gender,
      phone,
    });

    res.status(201).json({
      message: 'Member registered successfully',
      member: {
        id: result.member.member_id,
        name: result.member.name,
        email: result.member.email,
      },
      accessToken: result.accessToken,
      refreshToken: result.refreshToken,
    });
  } catch (error: any) {
    res.status(400).json({ error: error.message });
  }
};

export const login = async (req: Request, res: Response): Promise<void> => {
  try {
    const { email, password } = req.body;
    const result = await authService.loginMember(email, password);

    res.status(200).json({
      message: 'Login successful',
      member: {
        id: result.member.member_id,
        name: result.member.name,
        email: result.member.email,
      },
      accessToken: result.accessToken,
      refreshToken: result.refreshToken,
    });
  } catch (error: any) {
```

```
    res.status(401).json({ error: error.message });
  }
};

export const trainerLogin = async (req: Request, res: Response): Promise<void> => {
  try {
    const { email, password } = req.body;
    const result = await authService.loginTrainer(email, password);

    res.status(200).json({
      message: 'Trainer login successful',
      trainer: {
        id: result.trainer.trainer_id,
        name: result.trainer.name,
        email: result.trainer.email,
      },
      accessToken: result.accessToken,
      refreshToken: result.refreshToken,
    });
  } catch (error: any) {
    res.status(401).json({ error: error.message });
  }
};
```

Auth Routes (modules/auth/auth.routes.ts)

```
import { Router } from 'express';
import * as authController from './auth.controller';

const router = Router();

router.post('/signup', authController.signup);
router.post('/login', authController.login);
router.post('/trainer/login', authController.trainerLogin);

export default router;
```

5.2 Member Module

Member Service (modules/member/member.service.ts)

```
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

export const getMemberProfile = async (memberId: string) => {
  return await prisma.member.findUnique({
    where: { member_id: memberId },
    select: {
```

```
      member_id: true,
      name: true,
      email: true,
      age: true,
      gender: true,
      phone: true,
      join_date: true,
      status: true,
    },
  });
};

export const getMyWorkoutPlans = async (memberId: string) => {
  return await prisma.workoutPlan.findMany({
    where: { member_id: memberId },
    include: { trainer: { select: { name: true, specialization: true } } },
    orderBy: { created_at: 'desc' },
  });
};

export const getMyDietPlans = async (memberId: string) => {
  return await prisma.dietPlan.findMany({
    where: { member_id: memberId },
    include: { trainer: { select: { name: true, specialization: true } } },
    orderBy: { created_at: 'desc' },
  });
};

export const getMyAttendance = async (memberId: string) => {
  return await prisma.attendance.findMany({
    where: { member_id: memberId },
    orderBy: { date: 'desc' },
  });
};

export const getMyProgress = async (memberId: string) => {
  return await prisma.progress.findMany({
    where: { member_id: memberId },
    include: { trainer: { select: { name: true } } },
    orderBy: { updated_at: 'desc' },
  });
};
```

5.3 Trainer Module

Trainer Service (modules/trainer/trainer.service.ts)

```
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();
```

```
export const getAllMembers = async () => {
  return await prisma.member.findMany({
    select: {
      member_id: true,
      name: true,
      email: true,
      age: true,
      gender: true,
      phone: true,
      join_date: true,
      status: true,
    },
  });
};

export const assignWorkoutPlan = async (
  memberId: string,
  trainerId: string,
  planDetails: string
) => {
  return await prisma.workoutPlan.create({
    data: {
      member_id: memberId,
      trainer_id: trainerId,
      plan_details: planDetails,
    },
  });
};

export const assignDietPlan = async (
  memberId: string,
  trainerId: string,
  dietDetails: string
) => {
  return await prisma.dietPlan.create({
    data: {
      member_id: memberId,
      trainer_id: trainerId,
      diet_details: dietDetails,
    },
  });
};

export const recordAttendance = async (
  memberId: string,
  status: string
) => {
  return await prisma.attendance.create({
    data: {
      member_id: memberId,
      status,
    },
  });
```

```
};

export const updateProgress = async (
  memberId: string,
  trainerId: string,
  data: {
    weight: number;
    body_fat: number;
    muscle_mass: number;
    notes?: string;
  }
) => {
  return await prisma.progress.create({
    data: {
      member_id: memberId,
      trainer_id: trainerId,
      ...data,
    },
  });
};
```

6. EXPRESS APPLICATION SETUP

6.1 Main Application (app.ts)

```
import express from 'express';
import cors from 'cors';
import authRoutes from './modules/auth/auth.routes';
import memberRoutes from './modules/member/member.routes';
import trainerRoutes from './modules/trainer/trainer.routes';

const app = express();

// Middleware
app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/member', memberRoutes);
app.use('/api/trainer', trainerRoutes);

// Health check
app.get('/health', (_req, res) => {
  res.json({ status: 'OK', timestamp: new Date().toISOString() });
});
```

```
// 404 handler
app.use((_req, res) => {
  res.status(404).json({ error: 'Route not found' });
});

// Error handler
app.use((err: any, _req: express.Request, res: express.Response, _next: express.NextFuncti
  console.error('Error:', err);
  res.status(500).json({ error: 'Internal server error' });
});

export default app;
```

6.2 Serverless Entry Point (serverless.ts)

```
import app from './app';

export default app;
```

---

7. DEPLOYMENT CONFIGURATION

7.1 Vercel Configuration (vercel.json)

```
{
  "version": 2,
  "builds": [
    {
      "src": "src/serverless.ts",
      "use": "@vercel/node"
    }
  ],
  "routes": [
    {
      "src": "/(.*)",
      "dest": "src/serverless.ts"
    }
  ]
}
```

7.2 Environment Variables (.env)

```
# Database
DATABASE_URL="postgresql://user:password@host:5432/database"
```

```
# JWT Configuration
JWT_ACCESS_SECRET="your-super-secret-access-key-change-in-production"
JWT_REFRESH_SECRET="your-super-secret-refresh-key-change-in-production"
JWT_ACCESS_EXPIRY="15m"
JWT_REFRESH_EXPIRY="7d"

# Server Configuration
PORT=3000
NODE_ENV="development"
```

---

8. PACKAGE DEPENDENCIES

8.1 Backend Dependencies (package.json)

```
{
  "dependencies": {
    "@prisma/client": "^5.7.1",
    "express": "^4.18.2",
    "cors": "^2.8.5",
    "bcryptjs": "^2.4.3",
    "jsonwebtoken": "^9.0.2",
    "dotenv": "^16.3.1"
  },
  "devDependencies": {
    "@types/express": "^4.17.21",
    "@types/node": "^20.10.5",
    "@types/bcryptjs": "^2.4.6",
    "@types/jsonwebtoken": "^9.0.5",
    "@types/cors": "^2.8.17",
    "typescript": "^5.3.3",
    "prisma": "^5.7.1",
    "ts-node": "^10.9.2"
  }
}
```

---

9. KEY IMPLEMENTATION FEATURES

9.1 Security Features

✅ Password hashing with bcrypt (10 salt rounds)
✅ JWT-based authentication
✅ Token expiration (Access: 15min, Refresh: 7 days)
✅ Role-based access control

- ✅ SQL injection protection via Prisma ORM
- ✅ CORS enabled for cross-origin requests

9.2 Database Features

- ✅ UUID primary keys
- ✅ Foreign key constraints
- ✅ Cascade delete operations
- ✅ Unique email constraints
- ✅ Default timestamps
- ✅ Database seeding

9.3 API Features

- ✅ RESTful architecture
- ✅ JSON request/response
- ✅ Error handling middleware
- ✅ 404 route handler
- ✅ Health check endpoint
- ✅ Authenticated routes