GoWorkout Project Documentation

Team Members

- **Peter Safwat** - 231001577
- **Kerolos Wageh** - 231001117
- **Bavly Ramy** - 231000910
- **Karim Ahmed** - 231001537

---

1. Requirements and Design

1.1 Functional Requirements

- User registration and authentication system
- Calorie calculator based on user metrics (weight, height, age, gender)
- Three-tier training programs (Beginner, Intermediate, Advanced)
- User profile management
- Session-based access control

1.2 Non-Functional Requirements

- Responsive web interface compatible with modern browsers
- Secure password storage (6-10 characters enforced)
- Fast page load times
- Reliable MySQL database backend

1.3 Use Cases

Use Case 1: User Registration

- Actor: New User
- Flow: User enters first name, last name, username, and password → System validates input → Account created in database

Use Case 2: Login

- Actor: Registered User
- Flow: User enters credentials → System authenticates → User gains access to training programs

Use Case 3: Calculate Calories

- Actor: Any User (logged in or not)
- Flow: User inputs weight, height, age, gender, and goal → System calculates BMR and adjusts for goal → Daily calorie target displayed

Use Case 4: View Training Program

- Actor: Logged-in User
- Flow: User selects program level → System displays workout schedule and exercises

1.4 System Specifications

- **Frontend**: HTML, CSS, JavaScript
- **Backend**: Node.js with Express.js framework
- **Database**: MySQL
- **Architecture**: Client-Server with RESTful API

1.5 Data Models

User Table Schema:

```
users
├── ID (INT, PRIMARY KEY)
├── firstName (VARCHAR, NOT NULL)
├── secondName (VARCHAR, NOT NULL)
├── username (VARCHAR, NOT NULL)
└── password (VARCHAR, NOT NULL)
```

Logical Flow:

1. Client sends HTTP request → Express server
2. Server processes request → MySQL query
3. Database returns data → Server responds to client
4. Client updates UI

2. Implementation and Testing

2.1 Coding Standards and Best Practices

- **Naming Conventions**: camelCase for variables/functions, PascalCase for constructors
- **Code Organization**: Separate files for frontend (HTML/CSS/JS) and backend (server.js)
- **Error Handling**: Try-catch blocks for async operations, input validation
- **Security**: Password length validation, SQL injection prevention via parameterized queries

2.2 Code Formatting and Commenting

Example from script.js:

```javascript
// Calculate daily calories using BMR and goal
function calculateCalories() {
  const weight = parseFloat(document.getElementById('weight').value);
  const height = parseFloat(document.getElementById('height').value);
  const age = parseInt(document.getElementById('age').value, 10);

  // Positive number validation
  if (isNaN(weight) || weight <= 0) {
    alert('Please enter positive numbers!');
    return;
  }

  // Mifflin-St Jeor equation for BMR
  let bmr = gender === 'male'
    ? (10 * weight + 6.25 * height - 5 * age + 5)
    : (10 * weight + 6.25 * height - 5 * age - 161);
}
```

2.3 Modular Code Structure

- **Frontend Modules**:
  - Authentication functions (login, signup, logout)
  - Calculation functions (calculateCalories)

- Navigation functions (updateNav, goToPrograms)
- Profile management (initProfile)

- **Backend Modules**:
    - Express middleware (CORS, JSON parsing)
    - Database connection handler
    - API routes (signup, login, user profile)

2.4 Testing

Unit Tests (test.js):

```javascript
// Test showProgram function
function testShowProgram() {
  showProgram('beginner');
  console.log(elements.result.innerHTML);
}

// Test calculateCalories function
function testCalculateCalories() {
  setCalorieInputs({ weight: 70, height: 175, age: 25 });
  calculateCalories();
  console.log(elements.result.innerHTML);
}
```

Test Cases:

1. **Valid calorie calculation** (weight: 70kg, height: 175cm, age: 25, gender: male, goal: maintain weight)
2. **Female weight loss calculation** (weight: 60kg, height: 165cm, age: 30, gender: female, goal: lose)
3. **Invalid input handling** (missing fields trigger alert)
4. **Program display** (beginner and advanced programs render correctly)

Integration Testing:

- Signup → Database insertion → Login → Session creation → Profile retrieval
- All endpoints tested with Postman/manual browser testing

3. Version Control and Collaboration

3.1 Git Usage

- **Repository**: Project hosted on GitHub
- **Branching Strategy**:
    - `main` branch for stable releases
    - Feature branches for development: `calorie`, `register`, `training-programs`
- **Commit Frequency**: Regular commits after completing each feature or bug fix

3.2 Commit Messages

- ✅ "Completed the profile functions in the script.js file"
- ✅ "Modified the min and max password length"
- ✅ "Finished styling the whole web site"
- ✅ "Merge remote changes from main"

3.3 Branching and Merging

- Feature branches merged into `main` via pull requests
- Code review by team members before merging
- Resolved merge conflicts during integration

3.4 Project History

- Initial commit: Project structure setup
- Subsequent commits: Incremental feature additions
- Recent commits: Bug fixes and UI improvements
- Well-maintained history with descriptive messages

---

4. Project Management and Documentation

4.1 Timeline and Milestones

Milestone	Status
Frontend pages	✅ Complete
Database setup	✅ Complete
Backend API	✅ Complete
Testing	✅ Complete

4.2 Project Management

- **Tools Used**: GitHub Projects for task tracking

---

5. GitHub Usage

5.1 GitHub Backlog

- Issues created for each feature and bug
- Tasks labeled (enhancement, bug, documentation)
- Assigned to specific team members

5.2 Task Descriptions

Peter Safwat:

- Created MySQL database server with "users" table
- Created the calorie calculator
- Created the profile page

Kerolos Wageh:

- Created the index home page
- Created the style

- Created the server code connection to database

Bavly Ramy:

- Created training programs
- Created class, use case and sequence diagrams

Karim Ahmed:

- Created the login and signup pages
- Created the documentation

Note: All team members worked and modified on the script file which contains the important functions.

5.3 GitHub Actions

- Automated testing on push to `main` branch
- Code quality checks
- Deployment scripts for production

---

6. How to Run the Project

Prerequisites

- Node.js (v14 or higher)
- npm (Node Package Manager)

Installation & Setup

1. **Extract the project to your desired directory**
2. **Install dependencies:**

```bash
npm install
```

3. **Start the server:**

```bash
node server.js
```

4. **Open in your browser:**
   - Navigate to `http://localhost:3000/`

---

7. Getting Started

1. **Sign up for an account or log in**
2. **Use the Calorie Calculator to determine your daily calorie needs**
3. **Explore training programs suited to your fitness level**

---

8. Conclusion

GoWorkout is a functional fitness web application that successfully implements user authentication, calorie calculation, and workout program management. The project demonstrates proper software engineering practices including modular design, version control, testing, and documentation. The MySQL database integration provides reliable data persistence, while the Node.js backend ensures efficient API handling.