car-service-station-app ~ backend

backend/README.md

```
# 🚗 Car Service Station Backend API

A complete backend API for the Car Service Station application built with Node.js, Express, MySQL, and Sequelize ORM.

## 🚀 Quick Start

### Prerequisites
- Node.js 18+ installed
- MySQL 8.0+ installed and running
- Git (for cloning)

### 1. Clone & Setup
```bash
# Clone the repository
git clone <repository-url>
cd backend

# Install dependencies
npm install

# Setup environment
cp .env.example .env
```

2. Configure Environment

Edit the `.env` file with your database credentials:

```
PORT=5000
NODE_ENV=development
DB_HOST=localhost
DB_PORT=3306
DB_NAME=car_service_station_database
DB_USER=w3-93109
DB_PASS=omkarkar
JWT_SECRET=your-super-secret-jwt-key-change-in-production
JWT_REFRESH_SECRET=your-super-secret-refresh-key-change-in-production
```

3. Setup Database

```
# Create database and seed sample data
mysql -u w3-93109 -pomkarkar < migrations/init_db.sql
```

4. Start the Server

```
# Development mode (auto-restart on changes)
npm run dev

# Production mode
npm start
```

Server will start at: `http://localhost:5000`

📡 API Endpoints

🔐 Authentication

Method	Endpoint	Description	Access
POST	`/api/auth/register`	Register new user	Public
POST	`/api/auth/login`	Login user	Public
POST	`/api/auth/refresh`	Refresh access token	Public
POST	`/api/auth/logout`	Logout user	Authenticated
GET	`/api/auth/profile`	Get user profile	Authenticated
PUT	`/api/auth/profile`	Update profile	Authenticated

🛠️ Services

Method	Endpoint	Description	Access
GET	`/api/services`	Get all services	Public
```

Method	Endpoint	Description	Access
GET	`/api/services/:id`	Get service by ID	Public
GET	`/api/services/station/:stationId`	Get services by station	Public

⛽ Stations

Method	Endpoint	Description	Access
GET	`/api/stations`	Get all stations	Public
GET	`/api/stations/:id`	Get station by ID	Public

📅 Bookings

Method	Endpoint	Description	Access
POST	`/api/bookings`	Create booking	Authenticated
GET	`/api/bookings`	Get user bookings	Authenticated
GET	`/api/bookings/:id`	Get booking by ID	Authenticated
PATCH	`/api/bookings/:id/status`	Update booking status	Authenticated

👑 Admin (Admin/Superadmin only)

Method	Endpoint	Description
GET	`/api/admin/users`	Get all users
GET	`/api/admin/bookings`	Get all bookings
GET	`/api/admin/statistics`	Get system statistics

🧪 Testing the API

Using curl/PowerShell

1. Health Check

```
# PowerShell
Invoke-WebRequest -Uri http://localhost:5000/api/health -Method GET

# curl
curl http://localhost:5000/api/health
```

2. Register a New User

```
$body = @{
    email = "test@example.com"
    password = "password123"
    firstName = "John"
    lastName = "Doe"
    phone = "+12345567890"
} | ConvertTo-Json

Invoke-WebRequest -Uri http://localhost:5000/api/auth/register `
    -Method POST `
    -ContentType "application/json" `
    -Body $body
```

3. Login

```
$body = @{
    email = "test@example.com"
    password = "password123"
} | ConvertTo-Json

Invoke-WebRequest -Uri http://localhost:5000/api/auth/login `
    -Method POST `
    -ContentType "application/json" `
    -Body $body
```

4. Get Services (Public)

```
Invoke-WebRequest -Uri http://localhost:5000/api/services -Method GET
```

5. Get Stations (Public)

```
Invoke-WebRequest -Uri http://localhost:5000/api/stations -Method GET
```

Using Postman

1. Import the Postman collection from `postman_collection.json`
2. Set base URL: `http://localhost:5000/api`
3. Test endpoints in order:
   - Register → Login → Services → Stations → Bookings

📊 Database Schema

Tables Created:

- `users` - User accounts and authentication
- `services` - Available car services
- `stations` - Service station locations
- `station_service_prices` - Prices for services at each station
- `bookings` - Customer bookings
- `booking_status_history` - Booking status changes
- `receipts` - Service receipts
- `messages` - Chat messages

Sample Data Seeded:

- 3 users (client@example.com, admin@example.com, superadmin@example.com)
- 2 service stations
- 5 services
- Station-service prices

🔐 Default Users (Password: "password")

Email	Password	Role
client@example.com	password	Client
admin@example.com	password	Admin
superadmin@example.com	password	Super Admin

🛠️ Development

Project Structure

```
backend/
├── config/           # Configuration files
├── controllers/      # Route controllers
├── middleware/       # Custom middleware
├── models/          # Sequelize models
├── routes/          # API routes
├── migrations/      # Database migrations
├── tests/           # Test files
├── utils/           # Utility functions
├── uploads/         # File uploads
├── server.js        # Main server file
└── package.json
```

Available Scripts

```
npm run dev          # Start development server with nodemon
npm start           # Start production server
npm test           # Run tests
npm run migrate    # Run database migrations
```

Environment Variables

Variable	Description	Default
PORT	Server port	5000
NODE_ENV	Environment	development
DB_HOST	Database host	localhost
DB_PORT	Database port	3306
DB_NAME	Database name	car_service_station_database
DB_USER	Database user	w3-93109
DB_PASS	Database password	omkarkar
JWT_SECRET	JWT signing secret	-

Variable	Description	Default
JWT_REFRESH_SECRET	JWT refresh secret	-

🧪 Running Tests

```
# Run all tests
npm test

# Run specific test file
npm test -- tests/auth.test.js

# Run tests in watch mode
npm run test:watch
```

🐛 Troubleshooting

Database Connection Issues

```
# Test MySQL connection
mysql -u w3-93109 -pomkarkar -e "SHOW DATABASES;"

# Check if database exists
mysql -u w3-93109 -pomkarkar -e "USE car_service_station_database; SHOW TABLES;"
```

Port Already in Use

```
# Find process using port 5000
netstat -ano | findstr :5000

# Kill the process
taskkill /PID <PID> /F
```

Missing Dependencies

```
# Delete and reinstall
rm -rf node_modules package-lock.json
npm install
```

📝 API Response Format

Success Response

```
{
  "success": true,
  "message": "Operation successful",
  "data": {
    // Response data
  }
}
```

Error Response

```
{
  "success": false,
  "message": "Error description",
  "errors": [
    // Validation errors
  ],
  "error": "Detailed error (development only)"
}
```

🔐 Authentication Flow

1. **Register** → Get user data and tokens
2. **Login** → Get new tokens
3. **Use access token** in Authorization header: `Bearer <token>`
4. **Token expires** → Use refresh token to get new access token
5. **Logout** → Invalidate refresh token

📦 Dependencies

Production

- `express` - Web framework
- `mysql2` - MySQL driver

- `sequelize` - ORM
- `jsonwebtoken` - JWT authentication
- `bcryptjs` - Password hashing
- `socket.io` - Real-time communication
- `puppeteer` - PDF generation
- `express-validator` - Request validation

Development

- `nodemon` - Auto-restart server
- `jest` - Testing framework
- `supertest` - HTTP testing

📄 License

MIT

👥 Contributors

- [Your Name]
- [Groupmate Names]

📞 Support

For issues or questions:

1. Check the troubleshooting section
2. Review error logs in terminal
3. Contact the development team

```
## **backend/QUICK_TEST.md** (For your groupmate)

```markdown
# 🚀 Quick Test Guide for Groupmate

## ⏱️ 5-Minute Setup & Test

### Step 1: Clone and Navigate
```bash
git clone <repository-url>
cd backend
```

Step 2: One-Command Setup

```
# Run this single command to setup everything
npm install && cp .env.example .env && npm run dev
```

Step 3: Quick Test Commands

Open a new terminal and run these tests:

Test 1: Health Check

```
Invoke-WebRequest -Uri http://localhost:5000/api/health -Method GET
```

✅ Should return: `{"success": true, "message": "Server is running"}`

Test 2: View Services (No login needed)

```
Invoke-WebRequest -Uri http://localhost:5000/api/services -Method GET
```

✅ Should return a list of 5 car services

Test 3: Create Test Account

```
# Generate unique email to avoid "already exists" error
$timestamp = Get-Date -Format "yyyyMMddHHmmss"
$email = "test$timestamp@example.com"

$body = @{
    email = $email
    password = "password123"
    firstName = "Test"
    lastName = "User"
    phone = "+1234567890"
} | ConvertTo-Json
```

```
Invoke-WebRequest -Uri http://localhost:5000/api/auth/register `
    -Method POST `
    -ContentType "application/json" `
    -Body $body
```

✅ Should create account and return tokens

Test 4: Login with Pre-existing Account

```
$body = @{
    email = "client@example.com"
    password = "password"
} | ConvertTo-Json

Invoke-WebRequest -Uri http://localhost:5000/api/auth/login `
    -Method POST `
    -ContentType "application/json" `
    -Body $body
```

✅ Should login successfully (password is "password")

🎯 Test All Major Features

Feature 1: Authentication ✅

- [ ] Register new user
- [ ] Login with credentials
- [ ] Get user profile (requires token)
- [ ] Logout

Feature 2: Services & Stations ✅

- [ ] View all services
- [ ] View all stations
- [ ] View service details

Feature 3: Bookings ✅

- [ ] Create a booking (need service & station IDs)
- [ ] View user bookings
- [ ] Cancel a booking

Feature 4: Admin Features (if admin)

- [ ] View all users (admin only)
- [ ] View all bookings (admin only)
- [ ] View statistics (admin only)

🧮 Example Test Script

Save this as `test.ps1` and run it:

```
Write-Host "🚀 Testing Car Service Station Backend" -ForegroundColor Green
Write-Host ""

# 1. Health check
Write-Host "1. Health Check..." -ForegroundColor Yellow
try {
    $health = Invoke-WebRequest -Uri http://localhost:5000/api/health -Method GET
    Write-Host "   ✅ Server is running" -ForegroundColor Green
} catch {
    Write-Host "   ❌ Server not responding" -ForegroundColor Red
    exit 1
}

# 2. Test services
Write-Host "2. Testing Services API..." -ForegroundColor Yellow
try {
    $services = Invoke-WebRequest -Uri http://localhost:5000/api/services -Method GET
    $servicesData = $services.Content | ConvertFrom-Json
    Write-Host "   ✅ Found $($servicesData.data.Count) services" -ForegroundColor Green
} catch {
    Write-Host "   ❌ Services API failed" -ForegroundColor Red
}

# 3. Create test user
Write-Host "3. Creating test user..." -ForegroundColor Yellow
$timestamp = Get-Date -Format "yyyyMMddHHmmss"
$testEmail = "test$timestamp@example.com"

$registerBody = @{
```

```
    email = $testEmail
    password = "password123"
    firstName = "Test"
    lastName = "User"
    phone = "+1234567890"
} | ConvertTo-Json

try {
    $register = Invoke-WebRequest -Uri http://localhost:5000/api/auth/register `
        -Method POST `
        -ContentType "application/json" `
        -Body $registerBody
    Write-Host "   ✅ User created: $testEmail" -ForegroundColor Green
} catch {
    Write-Host "   ❌ User creation failed" -ForegroundColor Red
}

Write-Host ""
Write-Host "🎉 Basic tests complete!" -ForegroundColor Green
Write-Host "Backend URL: http://localhost:5000" -ForegroundColor Cyan
Write-Host "API Base: http://localhost:5000/api" -ForegroundColor Cyan
```

🦠 Common Issues & Fixes

Issue: "User already exists"

Fix: Use a unique email address (append timestamp)

Issue: "Connection refused"

Fix: Make sure server is running (`npm run dev`)

Issue: "Database connection failed"

Fix: Check MySQL is running and credentials in `.env`

Issue: "JWT secret missing"

Fix: Ensure `.env` file has JWT_SECRET and JWT_REFRESH_SECRET

📞 Need Help?

1. **Server logs** show detailed error messages
2. **Check .env file** has correct database credentials
3. **MySQL must be running** on port 3306
4. **Default credentials** in `.env.example` work with seeded data

🎉 Success Checklist

- [ ] Server starts without errors
- [ ] `/api/health` returns success
- [ ] `/api/services` returns 5 services
- [ ] Can register new user
- [ ] Can login with created user
- [ ] Can view stations

```
## **backend/TEST_EXAMPLES.ps1** (Ready-to-run test script)

```powershell
# Car Service Station Backend Test Script
# Save as test.ps1 and run: .\test.ps1

Write-Host "========================================" -ForegroundColor Cyan
Write-Host "🚗 Car Service Station Backend Test Suite" -ForegroundColor Cyan
Write-Host "========================================" -ForegroundColor Cyan
Write-Host ""

function Test-Health {
    Write-Host "[1/6] Testing Health Check..." -ForegroundColor Yellow
    try {
        $response = Invoke-WebRequest -Uri "http://localhost:5000/api/health" -Method GET
        $data = $response.Content | ConvertFrom-Json
        if ($data.success) {
            Write-Host "   ✅ Health Check PASSED" -ForegroundColor Green
            Write-Host "   Server: $($data.message)" -ForegroundColor Gray
            return $true
        }
    } catch {
        Write-Host "   ❌ Health Check FAILED" -ForegroundColor Red
        Write-Host "   Error: $_" -ForegroundColor Red
    }
```
```

```
    return $false
}

function Test-Services {
    Write-Host "[2/6] Testing Services API..." -ForegroundColor Yellow
    try {
        $response = Invoke-WebRequest -Uri "http://localhost:5000/api/services" -Method GET
        $data = $response.Content | ConvertFrom-Json
        if ($data.success) {
            Write-Host "   ✅ Services API PASSED" -ForegroundColor Green
            Write-Host "   Found $($data.data.Count) services" -ForegroundColor Gray
            return $data.data[0]  # Return first service for later tests
        }
    } catch {
        Write-Host "   ❌ Services API FAILED" -ForegroundColor Red
    }
    return $null
}

function Test-Stations {
    Write-Host "[3/6] Testing Stations API..." -ForegroundColor Yellow
    try {
        $response = Invoke-WebRequest -Uri "http://localhost:5000/api/stations" -Method GET
        $data = $response.Content | ConvertFrom-Json
        if ($data.success) {
            Write-Host "   ✅ Stations API PASSED" -ForegroundColor Green
            Write-Host "   Found $($data.data.Count) stations" -ForegroundColor Gray
            return $data.data[0]  # Return first station for later tests
        }
    } catch {
        Write-Host "   ❌ Stations API FAILED" -ForegroundColor Red
    }
    return $null
}

function Test-Register {
    Write-Host "[4/6] Testing User Registration..." -ForegroundColor Yellow
    $timestamp = Get-Date -Format "yyyyMMddHHmmss"
    $testEmail = "testuser$timestamp@example.com"

    $body = @{
        email = $testEmail
        password = "TestPassword123!"
        firstName = "Test"
        lastName = "User"
        phone = "+1234567890"
    } | ConvertTo-Json

    try {
        $response = Invoke-WebRequest -Uri "http://localhost:5000/api/auth/register" `
            -Method POST `
            -ContentType "application/json" `
            -Body $body

        $data = $response.Content | ConvertFrom-Json
        if ($data.success) {
            Write-Host "   ✅ Registration PASSED" -ForegroundColor Green
            Write-Host "   User: $testEmail" -ForegroundColor Gray
            Write-Host "   ID: $($data.data.user.id)" -ForegroundColor Gray
            return @{
                email = $testEmail
                password = "TestPassword123!"
                token = $data.data.tokens.accessToken
                user = $data.data.user
            }
        }
    } catch {
        Write-Host "   ❌ Registration FAILED" -ForegroundColor Red
        if ($_.Exception.Response) {
            $errorContent = $_.ErrorDetails.Message | ConvertFrom-Json
            Write-Host "   Error: $($errorContent.message)" -ForegroundColor Red
        }
    }
    return $null
}

function Test-Login {
    Write-Host "[5/6] Testing User Login..." -ForegroundColor Yellow

    # Try with seeded user first
    $body = @{
        email = "client@example.com"
        password = "password"
    } | ConvertTo-Json
```

```
    try {
        $response = Invoke-WebRequest -Uri "http://localhost:5000/api/auth/login" `
            -Method POST `
            -ContentType "application/json" `
            -Body $body

        $data = $response.Content | ConvertFrom-Json
        if ($data.success) {
            Write-Host "   ✅ Login PASSED (with seeded user)" -ForegroundColor Green
            Write-Host "   User: client@example.com" -ForegroundColor Gray
            return @{
                token = $data.data.tokens.accessToken
                user = $data.data.user
            }
        }
    } catch {
        Write-Host "   ⚠️  Seeded user login failed, trying admin..." -ForegroundColor Yellow

        # Try admin
        $body = @{
            email = "admin@example.com"
            password = "password"
        } | ConvertTo-Json

        try {
            $response = Invoke-WebRequest -Uri "http://localhost:5000/api/auth/login" `
                -Method POST `
                -ContentType "application/json" `
                -Body $body

            $data = $response.Content | ConvertFrom-Json
            if ($data.success) {
                Write-Host "   ✅ Login PASSED (with admin)" -ForegroundColor Green
                return @{
                    token = $data.data.tokens.accessToken
                    user = $data.data.user
                }
            }
        } catch {
            Write-Host "   ❌ Login FAILED" -ForegroundColor Red
        }
    }
    return $null
}

function Test-Profile($token) {
    if (-not $token) {
        Write-Host "[6/6] Skipping Profile Test (no token)" -ForegroundColor Yellow
        return $false
    }

    Write-Host "[6/6] Testing Profile API..." -ForegroundColor Yellow

    $headers = @{
        "Authorization" = "Bearer $token"
    }

    try {
        $response = Invoke-WebRequest -Uri "http://localhost:5000/api/auth/profile" `
            -Method GET `
            -Headers $headers

        $data = $response.Content | ConvertFrom-Json
        if ($data.success) {
            Write-Host "   ✅ Profile API PASSED" -ForegroundColor Green
            Write-Host "   User: $($data.data.user.email)" -ForegroundColor Gray
            return $true
        }
    } catch {
        Write-Host "   ❌ Profile API FAILED" -ForegroundColor Red
    }
    return $false
}

# Main Test Execution
Write-Host "Starting tests..." -ForegroundColor Cyan
Write-Host ""

$health = Test-Health
if (-not $health) {
    Write-Host ""
    Write-Host "❌ Server not responding. Please start the server with:" -ForegroundColor Red
    Write-Host "   npm run dev" -ForegroundColor Yellow
    exit 1
}
```

```
$service = Test-Services
$station = Test-Stations
$registeredUser = Test-Register
$loggedInUser = Test-Login
$profileTest = Test-Profile -token $loggedInUser.token

Write-Host ""
Write-Host "========================================" -ForegroundColor Cyan
Write-Host "📊 TEST RESULTS SUMMARY" -ForegroundColor Cyan
Write-Host "========================================" -ForegroundColor Cyan
Write-Host ""

$tests = @(
    @{Name="Health Check"; Result=$health},
    @{Name="Services API"; Result=($service -ne $null)},
    @{Name="Stations API"; Result=($station -ne $null)},
    @{Name="User Registration"; Result=($registeredUser -ne $null)},
    @{Name="User Login"; Result=($loggedInUser -ne $null)},
    @{Name="Profile API"; Result=$profileTest}
)

$passed = 0
$total = $tests.Count

foreach ($test in $tests) {
    if ($test.Result) {
        Write-Host "✅ $($test.Name)" -ForegroundColor Green
        $passed++
    } else {
        Write-Host "❌ $($test.Name)" -ForegroundColor Red
    }
}

Write-Host ""
Write-Host "🎯 Score: $passed/$total tests passed" -ForegroundColor Cyan

if ($passed -eq $total) {
    Write-Host ""
    Write-Host "🎉 CONGRATULATIONS! All tests passed!" -ForegroundColor Green
    Write-Host "Your backend is fully functional!" -ForegroundColor Green
} elseif ($passed -ge 3) {
    Write-Host ""
    Write-Host "⚠️  Basic functionality working. Some tests failed." -ForegroundColor Yellow
} else {
    Write-Host ""
    Write-Host "❌ Multiple tests failed. Check server logs." -ForegroundColor Red
}

Write-Host ""
Write-Host "🔗 API Endpoints:" -ForegroundColor Cyan
Write-Host "   Health: http://localhost:5000/api/health" -ForegroundColor Gray
Write-Host "   Services: http://localhost:5000/api/services" -ForegroundColor Gray
Write-Host "   Stations: http://localhost:5000/api/stations" -ForegroundColor Gray
Write-Host "   Register: POST http://localhost:5000/api/auth/register" -ForegroundColor Gray
Write-Host "   Login: POST http://localhost:5000/api/auth/login" -ForegroundColor Gray
Write-Host ""
Write-Host "📝 Default Users (password: 'password'):" -ForegroundColor Cyan
Write-Host "   client@example.com (Client)" -ForegroundColor Gray
Write-Host "   admin@example.com (Admin)" -ForegroundColor Gray
Write-Host "   superadmin@example.com (Super Admin)" -ForegroundColor Gray
```

How your groupmate should use this:

1. **Share the Git repository** with just the backend folder
2. **Tell them to run**:

   ```
   git clone <your-repo>
   cd backend
   ```

3. **Follow the README.md** for setup
4. **Run the test script**:

   ```
   .\TEST_EXAMPLES.ps1
   ```

The backend is now **fully documented and testable** by your groupmate! 🎉