FitFlow API - Authentication System

Implementation Complete

The authentication system is fully implemented with JWT tokens, bcrypt password hashing, and secure session management.

Features Implemented

1. **User Registration** (`POST /api/auth/register`)
   - Email validation
   - Password hashing with bcrypt (10 rounds)
   - Automatic 30-day free trial subscription
   - Returns access token + refresh token
2. **User Login** (`POST /api/auth/login`)
   - Credential validation
   - Password verification
   - JWT token generation
   - Session creation in database
3. **Get Current User** (`GET /api/auth/me`)
   - Protected endpoint (requires Bearer token)
   - Returns full user profile
   - Validates token on every request
4. **Refresh Token** (`POST /api/auth/refresh`)
   - Exchanges refresh token for new access token
   - Validates refresh token against stored session
   - Access token expires in 15 minutes
   - Refresh token expires in 7 days
5. **Logout** (`POST /api/auth/logout`)
   - Removes session from database
   - Invalidates refresh token
   - Protected endpoint

Security Features

Password Security

- bcrypt hashing with 10 salt rounds
- Passwords never stored in plaintext
- Password validation on login

JWT Tokens

- Access tokens: 15 minute expiration
- Refresh tokens: 7 day expiration
- Signed with separate secrets
- Bearer token authentication

Session Management

- Refresh tokens stored as hashed values
- Session tracking with user agent and IP
- Automatic session expiration
- Logout invalidates sessions

API Security

- Protected endpoints use auth middleware
- 401 Unauthorized for missing/invalid tokens
- 403 Forbidden for insufficient permissions
- Input validation with Zod

API Endpoints

```
POST   /api/auth/register    // Register new user
POST   /api/auth/login        // Login and get tokens
GET    /api/auth/me           // Get current user (protected)
POST   /api/auth/refresh      // Refresh access token
POST   /api/auth/logout       // Logout (protected)
```

Request Examples

Register:

```
curl -X POST http://localhost:4000/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "email": "user@example.com",
    "password": "securepassword",
    "name": "John Doe"
  }'
```

Login:

```
curl -X POST http://localhost:4000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "email": "user@example.com",
```

```
    "password": "securepassword"
  }'
```

Get Current User:

```
curl -X GET http://localhost:4000/api/auth/me \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN"
```

Refresh Token:

```
curl -X POST http://localhost:4000/api/auth/refresh \
  -H "Content-Type: application/json" \
  -d '{
    "refreshToken": "YOUR_REFRESH_TOKEN"
  }'
```

Response Format

Success (Register/Login):

```
{
  "ok": true,
  "data": {
    "user": {
      "id": "...",
      "email": "user@example.com",
      "name": "John Doe",
      "role": "user",
      "subscription": {
        "plan": "free",
        "status": "active",
        "expiresAt": "2025-12-01T..."
      }
    },
    "accessToken": "eyJhbGci...",
    "refreshToken": "eyJhbGci..."
  }
}
```

Error:

```
{
  "ok": false,
  "error": {
    "message": "Invalid credentials"
```

```
  }
}
```

Testing

Run the complete auth test suite:

```
./test-auth.sh
```

This tests:

- □ User registration
- □ Protected endpoint access
- □ Unauthorized access blocking
- □ Login validation
- □ Wrong password rejection
- □ Token refresh flow
- □ Logout and session cleanup

Integration with Other Routes

To protect any route, use the `authenticate` middleware:

```
import { authenticate, requireAdmin } from '../middleware/auth';

// Protected route (any authenticated user)
router.get('/profile', authenticate, handler);

// Admin-only route
router.get('/admin/users', authenticate, requireAdmin, handler);
```

Access user info in controllers:

```
export async function myHandler(req: AuthRequest, res: Response) {
  const userId = req.user?.userId;
  const userRole = req.user?.role;
  // ...
}
```

Files Created

- `src/utils/auth.ts` - Password hashing, JWT sign/verify
- `src/middleware/auth.ts` - Authentication middleware
- `src/controllers/authController.ts` - Auth endpoint handlers

- `src/routes/auth.ts` - Auth route definitions
- `test-auth.sh` - Comprehensive test suite

Next Steps

With auth complete, you can now:

1. □ Protect diet/workout/progress routes with `authenticate` middleware
2. □ Implement AI-powered plan generation (OpenRouter + RapidAPI)
3. □ Add cron jobs for daily diet generation
4. □ Connect frontend to backend API

Status: □ Production Ready