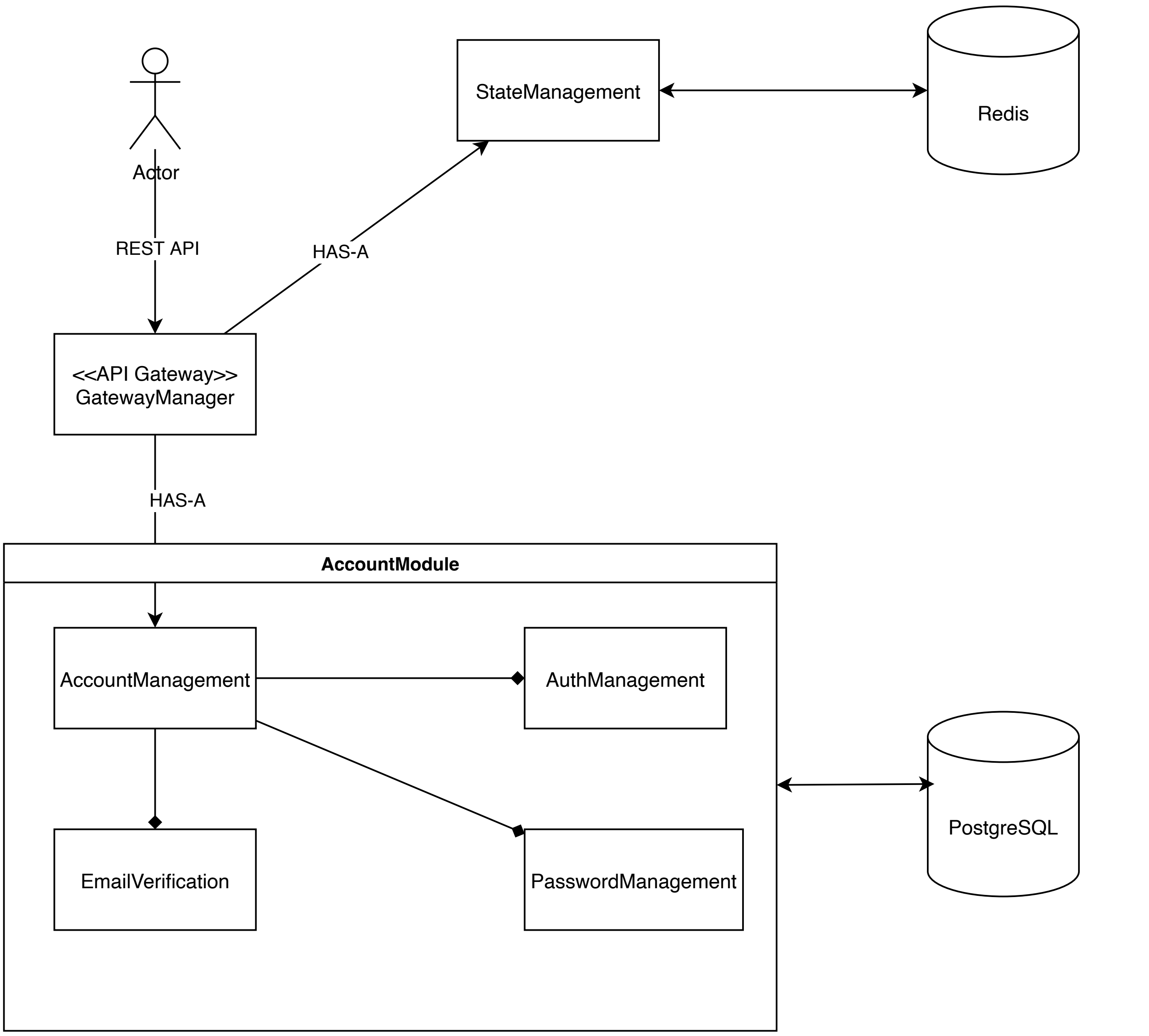
Introduction and Overview

The pomodoro backend system provides functionality that runs the single pomodoro at server end with persistence.

The system provides account managmenet that includes autherization, registration, password management, and email verification.

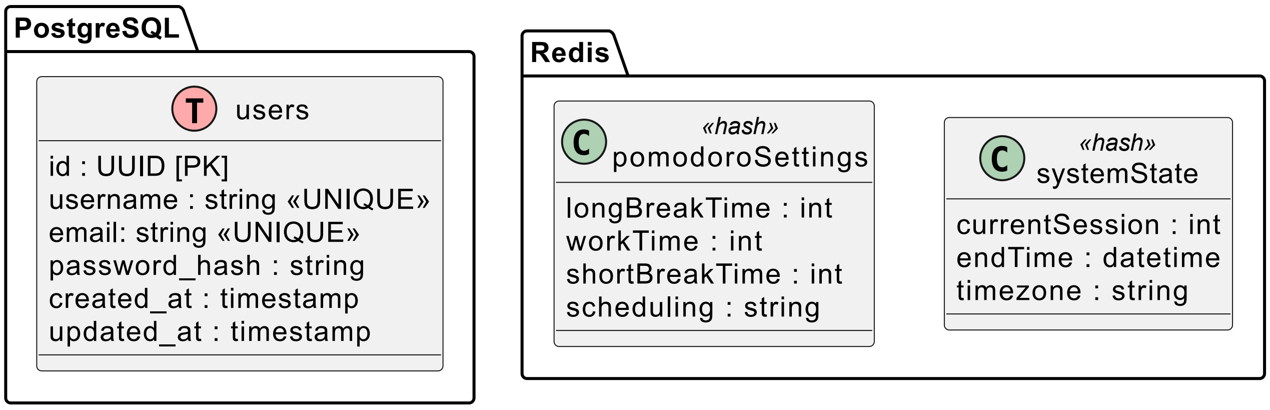
Requirements

System Architecture

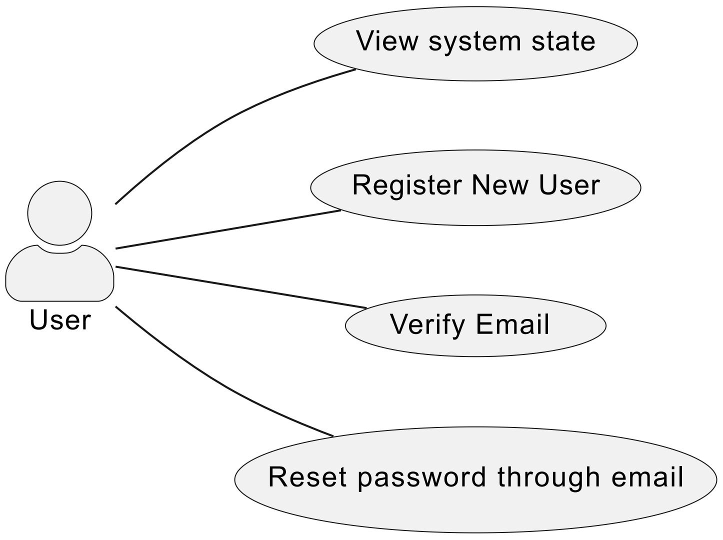


Rationale

Data Design



User Interface Design



API

API Reference

See full OpenAPI spec: openapi.yaml

Data Models

Session Type Values

- `WORK`: 25 min work session
- `SHORT_BREAK` : 5 min short break
- `LONG_BREAK` : 20 min long break

Authentication & Authorization

The API uses JWT Bearer token authentication with role-based access control (RBAC) for protected endpoints.

JWT Authentication

Include the token in the Authorization header:

```
Authorization: Bearer <your-jwt-token>
```

JWT tokens expire after **24 hours** and contain user identification information:

- User ID
- Username

Note: User roles are not stored in JWT tokens. Instead, the current role is fetched from the database on each request to ensure role changes take effect immediately without requiring token refresh.

Role-Based Access Control (RBAC)

The system implements two user roles with different privilege levels:

User Roles

- **USER**: Basic user role (default for new registrations)
- **ADMIN**: Administrative role with elevated privileges

Permission Matrix

Endpoint	USER Role	ADMIN Role	Description
`POST /auth/register`	Public	Public	User registration (creates USER role)
`POST /auth/login`	Public	Public	User authentication
`GET /auth/profile`	✅	✅	View user profile
`GET /system/state`	✅	✅	View system state
`POST /system/start`	❌	✅	Start/modify pomodoro system

API Endpoints

Authentication Endpoints

- `POST /auth/register` - Register new user (creates USER role)
- `POST /auth/register-admin` - Register admin user (creates ADMIN role) - **DEVELOPMENT ONLY**
- `POST /auth/login` - User login with JWT token response
- `GET /auth/profile` - Get user profile (requires authentication)

System Endpoints

- `GET /system/state` - Get current pomodoro system state (requires USER+ role)

- `POST /system/start` - Start new pomodoro session (requires ADMIN role)

Testing Role-Based Access Control

1. Register Users

```
# Register regular user (USER role)
curl -X POST http://localhost:8080/auth/register \
  -H "Content-Type: application/json" \
  -d
'{"username":"user1","password":"password","email":"user1@example.com"}'

# Register admin user (ADMIN role) - Development only!
curl -X POST http://localhost:8080/auth/register-admin \
  -H "Content-Type: application/json" \
  -d
'{"username":"admin","password":"adminpass","email":"admin@example.com"}'
```

2. Login and Get Tokens

```
# Login as regular user
curl -X POST http://localhost:8080/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"user1","password":"password"}'

# Login as admin user
curl -X POST http://localhost:8080/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"adminpass"}'
```

3. Test Permissions

USER Role Permissions:

```
# ✅ Can view system state
curl -H "Authorization: Bearer <user_token>"
http://localhost:8080/system/state

# ✅ Can view profile
curl -H "Authorization: Bearer <user_token>"
http://localhost:8080/auth/profile

# ❌ Cannot start pomodoro (403 Forbidden)
curl -X POST http://localhost:8080/system/start \
  -H "Authorization: Bearer <user_token>"
```

ADMIN Role Permissions:

```
# ✅ Can view system state
curl -H "Authorization: Bearer <admin_token>"
http://localhost:8080/system/state

# ✅ Can view profile
curl -H "Authorization: Bearer <admin_token>"
http://localhost:8080/auth/profile

# ✅ Can start pomodoro
curl -X POST http://localhost:8080/system/start \
  -H "Authorization: Bearer <admin_token>"
```

Security Notes

- **Admin Registration**: The `/auth/register-admin` endpoint should be **disabled in production**
- **JWT Secret**: Change the default JWT secret in production environments
- **Password Hashing**: Uses bcrypt with cost factor 12 for secure password hashing
- **Input Validation**: All endpoints validate and sanitize input data
- **Error Handling**: Appropriate HTTP status codes (401 Unauthorized, 403 Forbidden)
- **Role Changes**: User roles are fetched from database on each request, ensuring immediate effect of role changes without requiring logout/login

Role Change Behavior

Before (Old Approach):

- User role stored in JWT token
- Role changes require user to logout and login again
- Old tokens remain valid with outdated role information

After (Current Approach):

- User role fetched from database on each request
- Role changes take effect immediately on next API call
- No need for users to logout/login when roles change
- More secure and administratively convenient