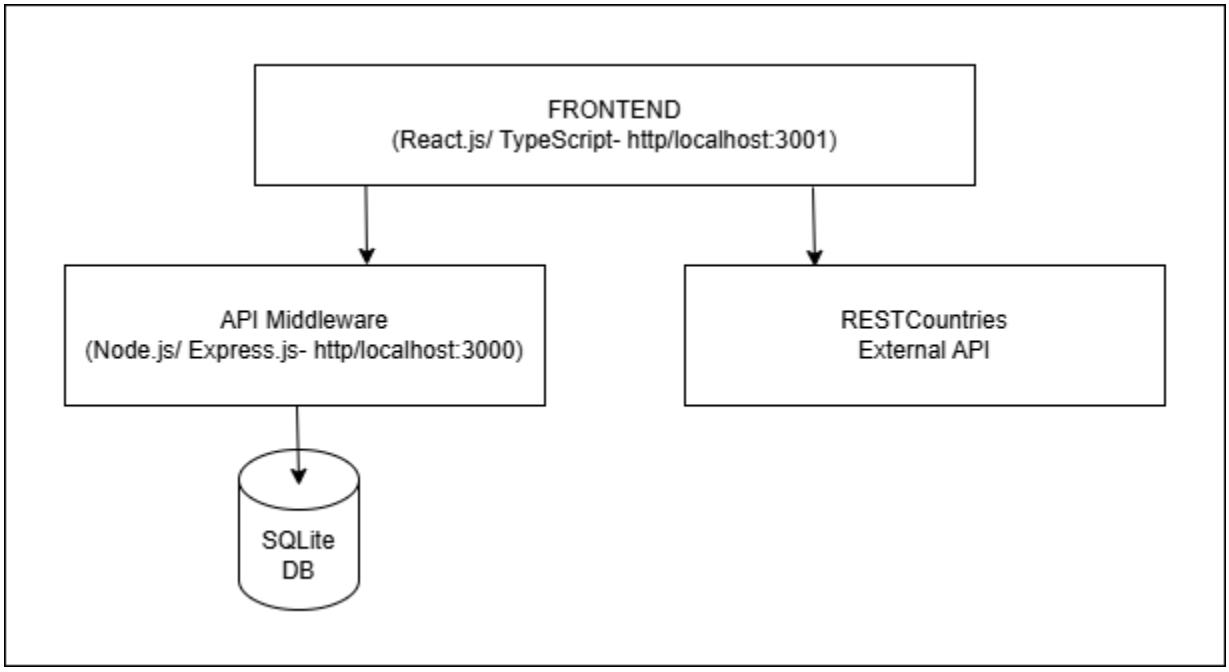
System Architecture

Overview

The Country API Middleware is a secure Node.js service that interfaces with RestCountries.com, providing filtered country data with authentication and API key management. This follows MVC (Model View Controller) Architecture.

Architecture Diagram



Flow Description

1. Frontend makes requests to our middleware API
2. Middleware authentication requests (session or API key)
3. For country data, middleware fetches from RestCountries.com
4. Middleware processes and filters the data
5. Response returned to frontend in standardized format

Security Implementation

- Password hashing with bcrypt
- CSRF protection for forms
- Rate limiting (100 requests/15 mins)
- API key authentication
- Session management with secure cookies

- Input validation on all endpoints
- Helmet for security headers
- CORS restricted to frontend origin

Architectural Decisions

- Clear separation of concerns
- Middleware pipeline for cross-cutting concerns
- Service layer for external API integration
- Repository pattern for data access
- RESTful API design principles

Performance Considerations

- Rate limiting to prevent exhaustion.
- Efficient database queries with proper indexing
- Minimal data transformation
- Async/await for non-blocking operations
- Connection pooling for database

Technical Stack

Backend

- Runtime: Node.js 18.x
- Framework: Express.js
- Database: SQLite with Sequelize ORM
- Authentication: Session-based + API keys
- Security: Helmet, CORS, CSRF protection, rate limiting
- Logging: Winston
- API Documentation: Swagger/OpenAPI

Frontend

- Framework: React.js with TypeScript
- UI Library: React Bootstrap
- State Management: Context API
- Routing: React Router
- HTTP Client: Axios

DevOps

- Containerization: Docker
- Orchestration: Docker Compose
- Web Server: Nginx (production)

API Documentation

Base URL

http://localhost:3000/api

Authentication

- Session cookies for frontend
- API keys for programmatic access (x-api-key header)

Endpoints

Auth Routes

- POST /auth/register - User registration
- POST /auth/login - User login
- POST /auth/logout - User logout
- GET /auth/me - Get current user

Country Routes

- GET /countries - Get all countries
- GET /countries/name/:name - Get country by name

API Key Routes

- POST /keys - Create new API key
- GET /keys - List user's API keys
- DELETE /keys/:id - Revoke API key

Admin Routes

- GET /admin/users - List all users (admin only)
- DELETE /admin/users/:id - Delete user (admin only)
- GET /admin/api-keys - List all API keys (admin only)
- GET /admin/stats/usage - Get API usage stats

Database Schema

Users Table

```
USER {
  string id PK
  string username
  string email
```

```
    string password
    boolean isAdmin
    boolean isActive
    datetime createdAt
    datetime updatedAt
}
```

API Keys Table

```
API_KEY {
    string id PK
    string key
    string name
    string userId FK
    datetime lastUsed
    integer usageCount
    boolean isActive
    datetime createdAt
    datetime updatedAt
}
```

Usage Logs Table

```
USAGE_LOG {
    string id PK
    string endpoint
    string apiKeyId FK
    string userId FK
    string ipAddress
    datetime createdAt
}
```

Security Implementation

Key Security Features

1. Authentication:
    - Session-based for frontend
    - API key for programmatic access
    - Password hashing with bcryptjs
2. Protection Middleware:
    - Helmet for secure headers
    - CORS restricted to frontend origin
3. Data Security:
    - SQLite database encryption

    - API keys hashed before storage
    - Sensitive fields encrypted
4. Session Management:
    - Secure, HTTP-only cookies
    - Session expiry (24 hours)
    - Server-side session storage

Deployment Guide

Docker Deployment

1. Ensure Docker and Docker Compose are installed
2. Clone the repository
3. Create .env files for both backend and frontend
4. Run: docker-compose up --build

Environment Variables

Backend (country-api-middleware/.env):
NODE_ENV=production
PORT=3000
DB_STORAGE=/data/database.sqlite
SESSION_SECRET=your-secret-key
CORS_ORIGIN=http://localhost:3001
Frontend (client/.env):
REACT_APP_API_URL=http://localhost:3000

Accessing the Application

- Frontend: http://localhost:3001
- Backend API: http://localhost:3000

Development Setup

Prerequisites

- Node.js 18.x
- npm
- SQLite3

Backend Setup

1. Navigate to country-api-middleware
2. Run npm install
3. Create .env file

4. Run npm run dev for development

Frontend Setup

1. Navigate to client
2. Run npm install
3. Create .env file
4. Run npm start

Testing Strategy

Test Types

1. Unit Tests: Jest for individual functions
2. Integration Tests: API endpoint testing
3. E2E Tests: Frontend interaction flows

Test Coverage

- Core middleware functionality
- Authentication flows
- API key management
- Admin operations
- Error handling

Future Improvements

Planned Enhancements

1. Enhanced API Features:
    - Country filtering/sorting
    - More detailed country information
    - Caching layer for RestCountries API
2. Security:
    - Two-factor authentication
    - API key permissions/scopes
    - Automated key rotation
3. Scalability:
    - Migrate to PostgreSQL
    - Redis for session storage
    - Load balancing
4. Monitoring:
    - Prometheus metrics
    - Enhanced logging

    - Alerting system

Documentation Improvements

- Interactive API docs with examples
- Deployment guides for cloud providers