Elevator Simulation System

Project Documentation

Version 1.0
Date: April 7, 2025

Table of Contents

➢ Project Overview

The Elevator Simulation System is a web application built with Symfony 7.2 that simulates the operation of elevator systems within buildings. This application allows users to create and manage virtual elevator systems, control individual elevators, and monitor system performance through a responsive web interface.

The system implements a realistic elevator dispatching algorithm that optimizes elevator selection based on multiple factors including current position, direction, and queue status. The application provides a comprehensive simulation environment that accurately models real-world elevator behavior and management.

➢ Key Capabilities:

Creation and management of multiple elevator systems
Individual elevator control (movement, door operations)
Smart elevator allocation based on efficiency algorithms
Real-time monitoring of elevator status
System-wide emergency controls
Usage statistics and performance metrics

➢ System Requirements

To run the Elevator Simulation System, your environment must meet the following requirements:

- ✓ **PHP**: Version 8.2 or higher
- ✓ **Composer**: Latest version recommended
- ✓ **Web Server**: Apache or Nginx (or Symfony's built-in web server for development)

- ✓ **Database**: SQLite 3 (pre-configured for simplicity)
- ✓ **Browser**: Modern web browser with JavaScript enabled (Chrome, Firefox, Safari, Edge)
- ✓ **Operating System**: Any OS that supports PHP 8.2+ (Windows, macOS, Linux)

➢ Project Structure

The application follows the standard Symfony MVC architecture:

```
elevator-symfony/
├── bin/                          # Symfony console and other executables
├── config/                       # Configuration files
│   ├── packages/                 # Package-specific configurations
│   ├── routes/                   # Route definitions
│   └── services.yaml             # Service container configuration
├── migrations/                   # Database migrations
├── public/                       # Web root directory
│   ├── css/                      # CSS stylesheets
│   ├── js/                       # JavaScript files
│   └── index.php                 # Front controller
├── src/                          # Application source code
│   ├── Controller/               # Controllers
│   │   ├── Api/                  # API controllers
│   │   ├── ElevatorController.php
│   │   └── ElevatorSystemController.php
│   ├── Entity/                   # Doctrine entities
│   │   ├── Elevator.php
│   │   └── ElevatorSystem.php
│   ├── Repository/               # Doctrine repositories
│   ├── Service/                  # Business logic services
│   │   ├── ElevatorService.php
│   │   └── ElevatorSystemService.php
│   └── Kernel.php                # Application kernel
├── templates/                    # Twig templates
│   ├── elevator_system/          # System-specific templates
│   ├── home/                     # Home page templates
│   └── base.html.twig            # Base template
├── var/                          # Cache and logs
│   ├── cache/                    # Cache files
│   └── data.db                   # SQLite
database
│   └── log/                      # Log files
├── vendor/                       #
Dependencies
├── .env                          # Environment configuration
├── .env.dev                      # Development environment configuration
├── composer.json                 # Composer configuration
├── composer.lock                 # Composer lock file
├── compose.yaml                  # Docker Compose configuration
├── compose.override.yaml         # Docker Compose override configuration
└── symfony.lock                  # Symfony lock file
```

➢ Dependencies

The project relies on the following key dependencies as defined in the composer.json file:

1) **Core Symfony Components (7.2.*)**

   i. **symfony/framework-bundle**: The core Symfony framework
   ii. **symfony/console**: Command-line interface tools
   iii. **symfony/dotenv**: Environment variable handling
   iv. **symfony/yaml**: YAML configuration support
   v. **symfony/twig-bundle**: Twig templating engine
   vi. **symfony/security-bundle**: Security components
   vii. **symfony/form**: Form handling
   viii. **symfony/validator**: Validation components
   ix. **symfony/serializer**: Serialization for API responses
   x. **symfony/asset**: Asset management
   xi. **symfony/property-access**: Property access component
   xii. **symfony/property-info**: Property info component

2) **Database & ORM**

   I. **doctrine/orm**: Version 3.3.* - Object-Relational Mapping
   II. **doctrine/doctrine-bundle**: Version 2.14.* - Doctrine integration for Symfony
   III. **doctrine/doctrine-migrations-bundle**: Version 3.4.* - Database migrations
   IV. **doctrine/dbal**: Version 3.* - Database Abstraction Layer

3) **Development Tools**

   I. **symfony/maker-bundle**: Version 1.62.* - Code generation tools
   II. **phpdocumentor/reflection-docblock**: Version 5.6.* - DocBlock parsing
   III. **phpstan/phpdoc-parser**: Version 2.1.* - PHP DocBlock parser

4) **Front-end Libraries**

   I. **Bootstrap 5.3**: Responsive UI framework (loaded via CDN)
   II. **jQuery**: JavaScript library for DOM manipulation (loaded via CDN)

➢ **Installation & Setup**

Follow these steps to install and configure the Elevator Simulation System:

1. **Copy the project**: simply copy the project files to your desired location.

2. **Install dependencies**: Navigate to the project directory and run:

```
cd elevator-symfony
composer install
```

3. **Set up environment variables**: The project already includes .env and .env.dev files. For production, create a .env.local file:

```
cp .env .env.local
```

Edit `.env.local` to configure your specific environment settings.

4. **Create the database**: The project is configured to use SQLite by default. Create the database with:

```
php bin/console doctrine:database:create
```

5. **Run migrations**:

```
php bin/console doctrine:migrations:migrate
```

Database Configuration

The application is pre-configured to use SQLite for simplicity. The database configuration can be found in the `.env` file:

```
# .env
DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
```

To switch to a different database system (MySQL, PostgreSQL), uncomment and modify the appropriate `DATABASE_URL` in your `.env.local` file:

```
# MySQL example
DATABASE_URL="mysql://username:password@127.0.0.1:3306/elevator_simulation?
serverVersion=8.0.32&charset=utf8mb4"

# PostgreSQL example
DATABASE_URL="postgresql://username:password@127.0.0.1:5432/elevator_simulation?
serverVersion=16&charset=utf8"
```

➢ **Running the Application**

There are several ways to run the application:

1. Using Symfony CLI (Recommended for Development)

If you have the Symfony CLI installed:

```
cd D:\elevator\elevator-symfony
symfony serve
```

This will start the development server, typically at `http://localhost:8000`.

2. Using PHP's Built-in Web Server

```
cd D:\elevator\elevator-symfony
php -S 127.0.0.1:8000 -t public/
```

3. Using Docker (with Docker Compose)

The project includes Docker Compose configuration files. To run the application with Docker:

```
cd D:\elevator\elevator-symfony
docker-compose up -d
```

4. Using Apache or Nginx

Configure your web server to point to the `public/` directory as the document root.

- **Apache Example** (the project already includes a `.htaccess` file in the public directory):

```
<VirtualHost *:80>
    ServerName elevator-simulation.local
    DocumentRoot D:/elevator/elevator-symfony/public

    <Directory D:/elevator/elevator-symfony/public>
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/elevator_error.log
    CustomLog ${APACHE_LOG_DIR}/elevator_access.log combined
</VirtualHost>
```

- **Nginx Example**:

```
server {
    listen 80;
    server_name elevator-simulation.local;
    root D:/elevator/elevator-symfony/public;

    location / {
        try_files $uri /index.php$is_args$args;
    }

    location ~ ^/index\.php(/|$) {
        fastcgi_pass unix:/var/run/php/php8.2-fpm.sock;  # Adjust for your PHP-FPM
setup
        fastcgi_split_path_info ^(.+\.php)(/.*)$;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        fastcgi_param DOCUMENT_ROOT $realpath_root;
        internal;
    }

    location ~ \.php$ {
        return 404;
    }

    error_log /var/log/nginx/elevator_error.log;
    access_log /var/log/nginx/elevator_access.log;
}
```

➢ **Core Features:**

- **Elevator System Management**

The application allows users to:

1. **Create New Systems**:
    I. Define minimum and maximum floor numbers
    II. Specify the number of elevators in the system
    III. Configure system-wide parameters
2. **Monitor System Status**:
    I. View all elevators in real-time
    II. Track system-wide statistics
    III. Identify busy periods and usage patterns
3. **Emergency Controls**:
    I. System-wide emergency stop
    II. Resume normal operations
    III. Override normal allocation algorithms

- **Individual Elevator Control**

Each elevator can be controlled through:

1. **Movement Operations**:
    I. Move to specific floors
    II. Change direction (up/down)
    III. Stop at current position
2. **Door Operations**:
    I. Open doors
    II. Close doors
    III. Hold doors open
3. **Destination Management**:
    I. Add floors to destination queue
    II. Clear destination queue
    III. Prioritize specific destinations

- **Allocation Algorithm**

When a user calls an elevator from a specific floor, the system uses a sophisticated algorithm to select the optimal elevator based on:

1. **Distance Factor**: Current distance to the requested floor
2. **Direction Factor**: Whether the elevator is already moving in the required direction
3. **Occupancy Factor**: Current queue length and capacity
4. **Status Factor**: Whether the elevator is in service or has doors open

The algorithm calculates a composite score for each available elevator and selects the one with the best score to respond to the call.

- **Statistics and Monitoring**

The system provides detailed statistics including:

1. **Usage Patterns**:
    I. Busiest floors
    II. Peak usage times
    III. Average wait times
2. **System Performance**:
    I. Average response time
    II. Efficiency metrics
    III. Comparative elevator usage
3. **Historical Data**:
    I. Call history
    II. Movement patterns
    III. System event log

➢ **Entity Relationships**

The application revolves around two main entities with a one-to-many relationship:

I. ElevatorSystem Entity

This entity represents a building with multiple elevators:

Properties:
- id: Unique identifier for the system
- name: Name of the elevator system (e.g., "Office Building A")
- minFloor: Lowest floor number in the building
- maxFloor: Highest floor number in the building
- status: Current status of the entire system (e.g., "operational",
- "maintenance", "emergency")
- callHistory: JSON array storing historical call data for analytics
- elevators: One-to-many relationship with Elevator entities

II. Elevator Entity

This entity represents an individual elevator within a system:

Properties:
- id: Unique identifier for the elevator
- name: Name/identifier of the elevator (e.g., "Elevator A")
- currentFloor: The floor where the elevator is currently located
- status: Current status of the elevator (e.g., "idle", "moving",
- "maintenance", "emergency")
- direction: Current movement direction ("up", "down", "stationary")
- doorOpen: Boolean indicating whether the elevator doors are open
- destinationQueue: JSON array of floors the elevator needs to visit
- elevatorSystem: Many-to-one relationship back to the parent system

➢ **API Documentation**

The application provides a RESTful API for integrating with external systems, located in the `src/Controller/Api` directory:

- **Base URL**

```
/api/v1
```

- **Authentication**

API requests require authentication using API tokens. Include the token in the request header:

```
Authorization: Bearer {your-api-token}
```

Elevator System Endpoints

Method	Endpoint	Description
GET	/api/v1/systems	List all elevator systems
GET	/api/v1/systems/{id}	Get details of a specific system
POST	/api/v1/systems	Create a new elevator system
PUT	/api/v1/systems/{id}	Update an elevator system
DELETE	/api/v1/systems/{id}	Delete an elevator system
POST	/api/v1/systems/{id}/call	Call an elevator to a specific floor
POST	/api/v1/systems/{id}/emergency-stop	Trigger system-wide emergency stop
POST	/api/v1/systems/{id}/resume	Resume system operation after emergency

Elevator Endpoints:

Method	Endpoint	Description
GET	/api/v1/elevators	List all elevators
GET	/api/v1/elevators/{id}	Get elevator details
PUT	/api/v1/elevators/{id}	Update elevator status
POST	/api/v1/elevators/{id}/move	Move elevator to a floor
POST	/api/v1/elevators/{id}/door	Open/close elevator door
POST	/api/v1/elevators/{id}/add-destination	Add destination to queue
POST	/api/v1/elevators/{id}/clear-queue	Clear destination queue
POST	/api/v1/elevators/{id}/emergency-stop	Emergency stop individual elevator

❖ **Example API Request**

```
curl -X POST \
  http://localhost:8000/api/v1/systems/1/call \
  -H 'Authorization: Bearer your-api-token' \
  -H 'Content-Type: application/json' \
  -d '{
    "floor": 5,
    "direction": "up"
}'
```

➢ **User Interface**

The application features a responsive Bootstrap interface organized into several key views:

- **Dashboard**

The main dashboard provides an overview of all elevator systems with key metrics and status indicators:

I. System status indicators (operational, maintenance, emergency)
II. Quick access to control panels
III. Summary statistics
IV. Recent activity log

- **System Detail View**

This view shows comprehensive information about a specific elevator system:

I. Visual representation of all elevators in the building
II. Current position and status of each elevator
III. Floor call buttons
IV. System-wide controls

- **Elevator Control Panel**

The control panel for individual elevators includes:

I. Current floor indicator
II. Direction indicator
III. Door status
IV. Destination queue display
V. Movement controls
VI. Door controls

- **Auto-Refresh Functionality**

The user interface automatically refreshes every 5 seconds to update elevator status and position information. This is implemented through JavaScript timers that reload specific components without requiring a full-page refresh.

➢ **Troubleshooting**

Common Issues and Solutions:

1. **Database Connection Issues**:

    I. Verify database path in .env file (for SQLite)
    II. Ensure SQLite file is writable by the web server
    III. For SQLite, check that the var/ directory exists and is writable

2. **Permission Problems**:

    I. Ensure proper file permissions on var/ directory
    II. Web server user must have right access to var/cache/ and var/log/

```
chmod -R 777 var/
```

3. **Page Refresh Not Working**:

    - Check if JavaScript is enabled in the browser
    - Inspect browser console for errors
    - Verify that the auto-refresh script is loaded

4. **Elevator Movement Issues**:

    - Check database for corrupt status values
    - Verify that the elevator service is functioning correctly
    - Clear browser cache to ensure latest JavaScript is loaded

Logs:

Application logs are stored in the var/log/ directory:

dev.log: Development environment logs
prod.log: Production environment logs

To increase log verbosity, modify the logging level in config/packages/monolog.yaml.