# Authentication Documentation

## Symfony General Behavior

Every incoming HTTP request is redirected to the front controller located in `public/index.php`. The front controller's role is to boot the Symfony kernel, analyze the request, ask the appropriate controller to build the response, and send this response back to the user.

Here is a detailed breakdown of what the front controller does:

1. Loads the environment variables set in the `.env.local` file (or `.env` if there is no `.env.local` file) located at the root of the project folder.
2. Instantiates a `App\Kernel` object based on the chosen environment.
3. Creates a `Symfony\Component\HttpFoundation\Request` object containing all the request data.
4. Passes the request instance to the kernel to build a `Symfony\Component\HttpFoundation\Response` object.
5. Sends the response back to the user.
6. Executes the last kernel operations before closure.

During this process, useful events are dispatched, on which listeners can be hooked.

## Authentication in the Project

### Prerequisites

To represent users, a class implementing the `Symfony\Component\Security\Core\User\UserInterface` interface should be implemented.

### Configuration

The authentication parameters are defined in the `security.yaml` file. Here is an example configuration:

```yaml
security:
    # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
    # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
    providers:
        # used to reload user from session & other features (e.g. switch_user)
        app_user_provider:
            entity:
                class: App\Entity\User
                property: username
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            provider: app_user_provider

            # activate different ways to authenticate
            # https://symfony.com/doc/current/security.html#the-firewall
            form_login:
                login_path: app_login
                check_path: app_login
                enable_csrf: true

            logout:
                path: app_logout
                target: app_homepage
            # https://symfony.com/doc/current/security/impersonating_user.html
            # switch_user: true

    # Easy way to control access for large sections of your site
    # Note: Only the *first* access control that matches will be used
    access_control:
        - { path: ^/admin, roles: ROLE_ADMIN }
        - { path: ^/users, roles: ROLE_ADMIN }
        - { path: ^/users/create, roles: ROLE_ADMIN }
        - { path: ^/tasks, roles: ROLE_USER }
        - { path: ^/users, roles: ROLE_USER }

when@test:
    security:
        password_hashers:
            # By default, password hashers are resource intensive and take time. This is
            # important to generate secure password hashes. In tests however, secure hashes
            # are not important, waste resources and increase test times. The following
            # reduces the work factor to the lowest possible values.
            Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
                algorithm: auto
                cost: 4 # Lowest possible value for bcrypt
                time_cost: 3 # Lowest possible value for argon
                memory_cost: 10 # Lowest possible value for argon
```

## Authentication Process

The authentication process is handled by the `Symfony\Component\Security\Http\Firewall` class. This class is responsible for calling the authentication listeners and the authentication failure handlers.

The authentication listeners are responsible for authenticating the user. They are called in the order they are defined in the `security.yaml` file. The first listener that returns a non-null value is considered to be the authenticator. If no authenticator is found, the authentication process fails.

The authentication failure handlers are responsible for handling the authentication failure. They are called in the order they are defined in the `security.yaml` file. The first handler that returns a non-null value is considered to be the failure handler. If no failure handler is found, the authentication process fails.

### Authentication Listeners

Authentication listeners are responsible for authenticating the user. They are called in the order they are defined in the `security.yaml` file. The first listener that returns a non-null value is considered to be the authenticator. If no authenticator is found, the authentication process fails.

#### Authentication Failure Handlers

Authentication failure handlers are responsible for handling the authentication failure. They are called in the order they are defined in the `security.yaml` file. The first handler that returns a non-null value is considered to be the failure handler. If no failure handler is found, the authentication process fails.

#### Authentication Success Handlers

Authentication success handlers are responsible for handling the authentication success. They are called in the order they are defined in the `security.yaml` file. The first handler that returns a non-null value is considered to be the success handler. If no success handler is found, the authentication process fails.

#### Authentication Events

Authentication events are dispatched during the authentication process. They can be used to hook listeners that will be executed when the event is dispatched.

##### Authentication Events List

| Event Name | Description |
|---|---|
| `Symfony\Component\Security\Http\Event\LoginSuccessEvent` | Dispatched when the user successfully logs in. |
| `Symfony\Component\Security\Http\Event\LoginFailureEvent` | Dispatched when the user fails to log in. |
| `Symfony\Component\Security\Http\Event\LogoutEvent` | Dispatched when the user logs out. |

#### Authentication Events Listeners

Authentication events listeners are responsible for executing code when the event is dispatched. They are called in the order they are defined in the `services.yaml` file.

#### Authentication Events Listeners List

| Event Name | Listener Class | Description |
|---|---|---|
| `Symfony\Component\Security\Http\Event\LoginSuccessEvent` | `App\EventListener\Authentication\LoginSuccessListener` | Logs the user login. |
| `Symfony\Component\Security\Http\Event\LoginFailureEvent` | `App\EventListener\Authentication\LoginFailureListener` | Logs the user login failure. |
| `Symfony\Component\Security\Http\Event\LogoutEvent` | `App\EventListener\Authentication\LogoutListener` | Logs the user logout. |

### Authentication Providers

Authentication providers are responsible for authenticating the user. They are called in the order they are defined in the `security.yaml` file. The first provider that returns a non-null value is considered to be the authenticator. If no authenticator is found, the authentication process fails.

## Access Control (How to modify the access to a page)

Access control is defined in the `security.yaml` file. Here is an example configuration:

```yaml
security:
    access_control:
        - { path: ^/admin, roles: ROLE_ADMIN }
        - { path: ^/users, roles: ROLE_ADMIN }
        - { path: ^/users/create, roles: ROLE_ADMIN }
        - { path: ^/tasks, roles: ROLE_USER }
        - { path: ^/users, roles: ROLE_USER }
```

You can also use the `@IsGranted` annotation to restrict access to a controller action. Here is an example:

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;

class UserController extends AbstractController
{
    /**
     * @Route("/users", name="user_index")
     * @IsGranted("ROLE_USER")
     */
    public function index()
    {
        return $this->render('user/index.html.twig', [
            'controller_name' => 'UserController',
        ]);
    }
}
```

## Log In

### Log In Form

The log in form is defined in the `templates/security/login.html.twig` file. It is displayed when the user tries to access a page that requires authentication.

### Log In Form Controller

The log in form controller is defined in the `src/Controller/SecurityController.php` file. It is responsible for displaying the log in form and authenticating the user.

### Log In Form Template

The log in form template is defined in the `templates/security/login.html.twig` file. It is responsible for displaying the log in form.

## Log Out

### Log Out Controller

The log out controller is defined in the `src/Controller/SecurityController.php` file. It is responsible for logging out the user.

## User Registration

### User Registration Form

The user registration form is defined in the `templates/user/create.html.twig` file. It is displayed when the user tries to access a page that requires authentication.

### User Registration Form Controller

The user registration form controller is defined in the `src/Controller/UserController.php` file. It is responsible for displaying the user registration form and registering the user.

### User Registration Form Template

The user registration form template is defined in the `templates/user/create.html.twig` file. It is responsible for displaying the user registration form.

## User Profile

### User Profile Form

The user profile form is defined in the `templates/user/edt.html.twig` file. It is displayed when the user tries to access a page that requires authentication.

### User Profile Form Controller

The user profile form controller is defined in the `src/Controller/UserController.php` file. It is responsible for displaying the user profile form and updating the user profile.

### User Profile Form Template

The user profile form template is defined in the `templates/user/edt.html.twig` file. It is responsible for displaying the user profile form.

# Path: public/support/docs/Authentication/authorization.md