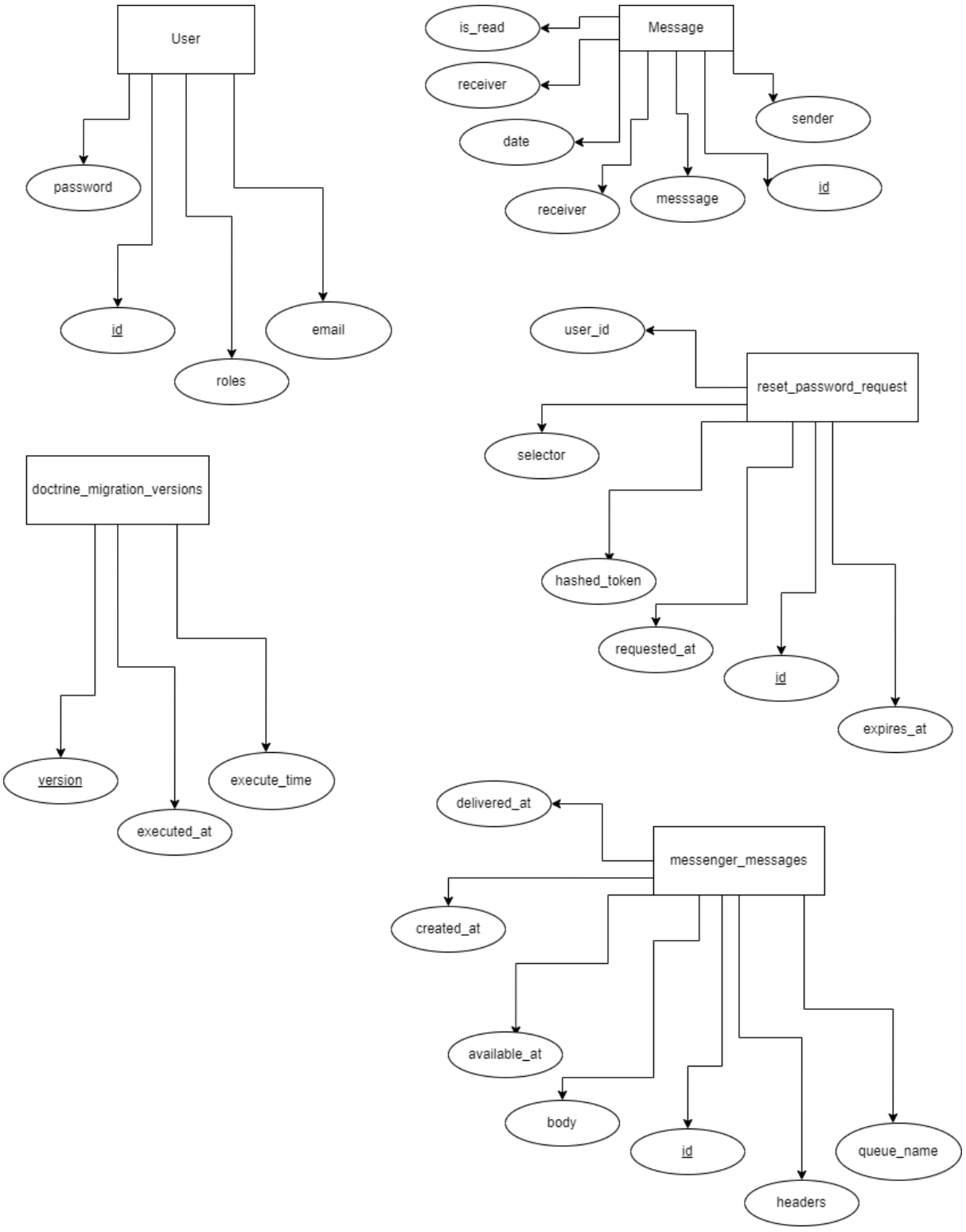
Symfony ChatApp Documentation

By: Félix Martínez Bendicho

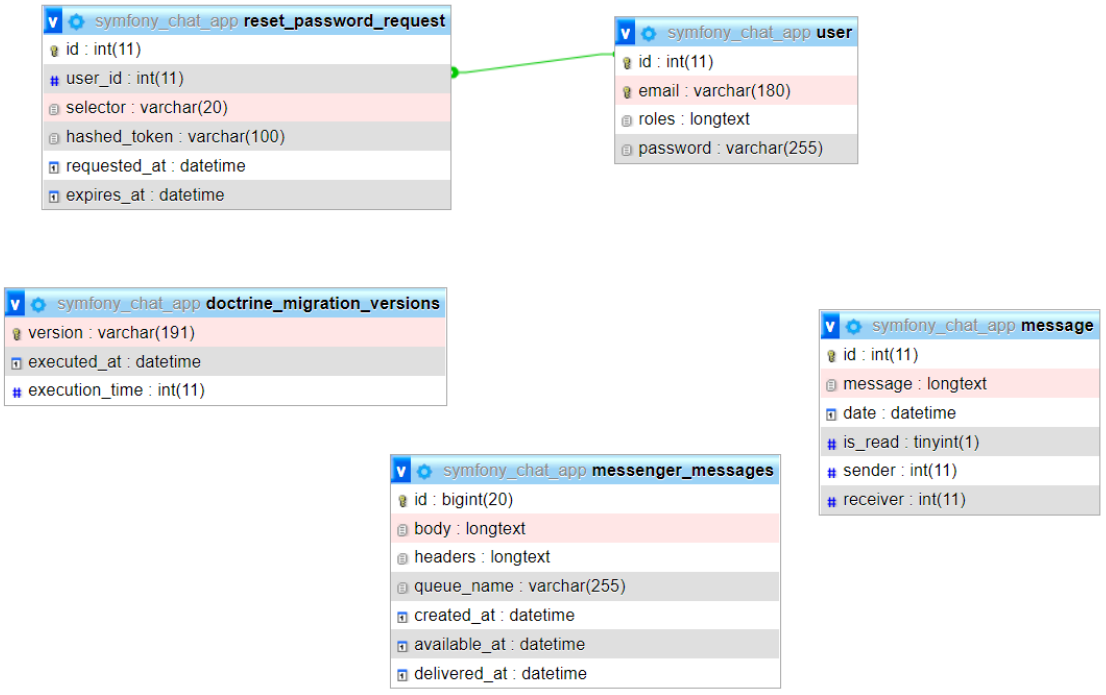
Extensions

ENLARGEMENT	MADE (Y/N)
A1 Self-registration	Y ½
A2 Password Recovery	Y ½
A3 Message to multiple recipients	N
A4 Encrypted user password	Y
A5 User avatar	N
A6 User profile	N
A7 Friendship	N
A8 Groups	N
A9 Administration zone	Y
A10 Attached files	N
A11 Images	N
A12 Testing	N
A13 Outbox	Y
A14 Presentation	Y
A15 Use AJAX	N
A16 SPA	N

Database diagram


User
password
id
email
roles
Message
is_read
receiver
date
receiver
messsage
sender
id
user_id
reset_password_request
selector
hashed_token
requested_at
id
expires_at
doctrine_migration_versions
version
execute_time
executed_at
delivered_at
messenger_messages
created_at
available_at
body
id
queue_name
headers

E/R Scheme



Symfony automatically creates the following tables:

- reset_password_request
- messenger_messages
- doctriner_migration_versions

These tables are used for sending emails, managing migrations and reset user passwords.

Provided data

The provided data is located inside the docs folder in the root folder. Just execute the provided sql statements in order to load the samples.

What includes:

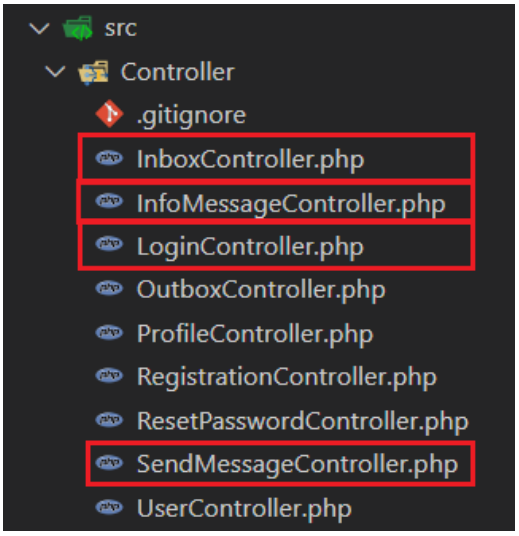
- Admin user:
  - Email: admin@admin.com
  - Password: admin
- Normal user 1:
  - Email: example1@example1.com
  - Password: 123456
- Normal user 2:
  - Email: example2@example3.com
  - Password: 123456
- Message 1:

    - From: example1@example1.com
    - To: example2@example2.com
- Message 2:
    - From: example2@example2.com
    - To: example1@example1.com

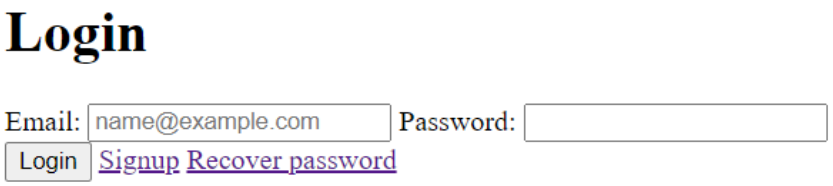
Technical explanation

Basic functionality

Controllers:



By default the app opens the *LoginController* route, which renders the *login/index.html.twig* template:



Once you are logged in it redirects to the inbox page, this was defined in the *security.yaml* file:

```yaml
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: login
            check_path: login
            default_target_path: inbox
            always_use_default_target_path: true
        # https://symfony.com/doc/current/security/impersonating_user.html
        # switch_user: true
        logout:
            path: app_logout
```

There you have different options:

Outbox Profile Send Message Logout

If you click on any of the existing messages you will see it in more detail:

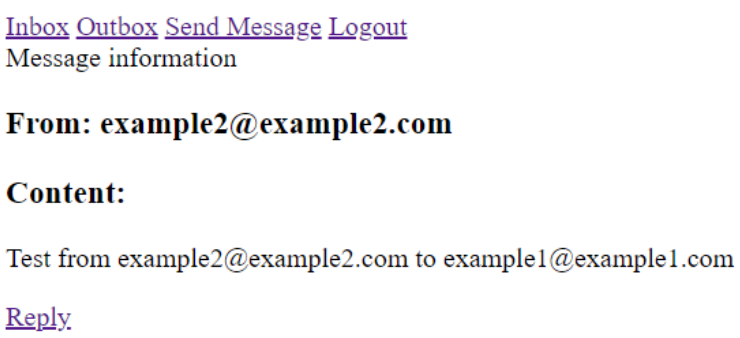
Inbox

example2@example2.com

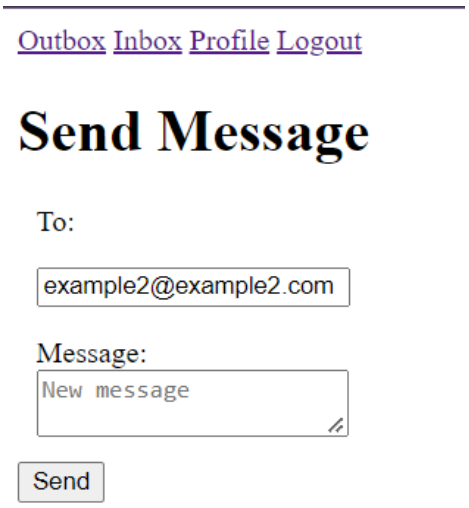
Test from example2@example2.com to example1@example1.com

Read

Once you have clicked on it, you will be redirected to the following page:



If you click to the *Reply* link, you will be redirected to the same *send_message* form, but with the email automatically set as the addressee:



A1. Self-registration

From the default index login page you have a *sign up* link, which redirects you to the registration route, where the *RegistrationController* renders the *registration/register.html.twig*:

```
return $this->render('registration/register.html.twig', [
    'registrationForm' => $form->createView()
]);
```

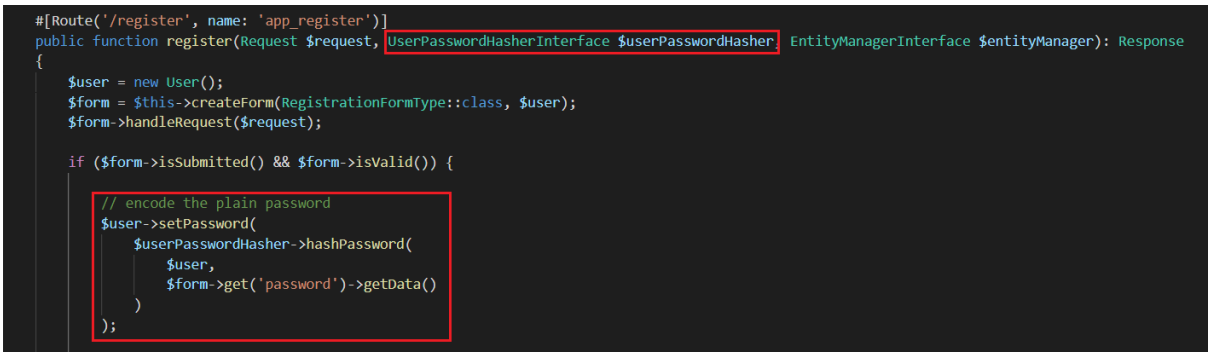
When the required inputs are valid the user will be redirected to the login page, where a login would be necessary if the user wants to access the app for the first time. Although at this point, the user is already stored in the database.

A2. Password recovery

As we can see, there are some issues with mails in Symfony. Currently all the code from the *ResetPasswordController* should work, but instead this doesn't happen. Since the email can't be sent the user isn't able to change the password.

A4. Encrypted user passwords

As this app uses the default Symfony user control, password hashing is implemented by default:



A9. Administration zone

The administration zone has been made through a CRUD Controller from the Symfony documentation with the following command: *bin/console make:crud*.

This command will create the following files:

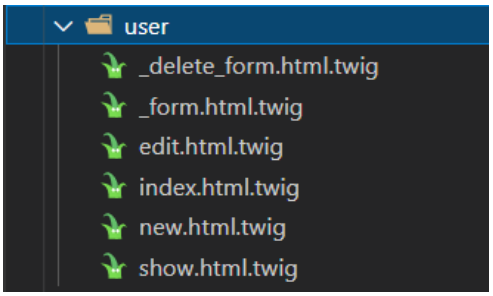
```php
#[Route('/admin')]
class UserController extends AbstractController
{
    #[Route('/', name: 'user_index', methods: ['GET'])]
    public function index(UserRepository $userRepository): Response
    {…
    }

    #[Route('/new', name: 'user_new', methods: ['GET', 'POST'])]
    public function new(Request $request, EntityManagerInterface $entityManager): Response
    {…
    }

    #[Route('/{id}', name: 'user_show', methods: ['GET'])]
    public function show(User $user): Response
    {…
    }

    #[Route('/{id}/edit', name: 'user_edit', methods: ['GET', 'POST'])]
    public function edit(Request $request, User $user, EntityManagerInterface $entityManager): Response
    {…
    }

    #[Route('/{id}', name: 'user_delete', methods: ['POST'])]
    public function delete(Request $request, User $user, EntityManagerInterface $entityManager): Response
    {…
    }
}
```



As we can see it provides us with all the utilities in order to create, read, update and delete from the selected entity, in this case this CRUD has been made from the User entity.

To access the admin zone you must be an administrator, this was implemented following a RBAC system (Role Based Access Control) in the *security.yaml* configuration file:

```yaml
# 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: ^/profile, roles: ROLE_USER }
```

These user roles are provided in the *RegistrationController*. Although, currently, the only possible admin user is the one in the sample data.

A13. Outbox

When the user clicks on the outbox link within the navigation links, it will be redirected to the outbox template: *outbox/index.html.twig*. Passing through the messages and the users.

```php
class OutboxController extends AbstractController
{
    /**
     * This will handle the outbox route itself
     */
    #[Route('/outbox', name: 'outbox')]
    public function index(MessageRepository $messageRepository, UserRepository $userRepository): Response
    {
        /** @var \App\Entity\User $user */
        $user = $this->getUser();
        $allUsers = $userRepository->findAll();

        // Get all the messages from the repository where the sender is the current user
        $messages = $messageRepository->findBy(
            ['sender' => $user->getId()]
        );

        // Render the inbox html.twig with the messages object, users and the role,
        // in order to allow the user to access to the admin zone if it's an admin.
        return $this->render('outbox/index.html.twig', [
            'outbox_message' => $messages,
            'users' => $allUsers,
            'user_role' => $user->getRoles()[0]
        ]);
    }
}
```

In the outbox template we run two loops, one for displaying the sent messages, and another one for displaying the users to which the message was sent.

```twig
<div>
    <h1>Outbox</h1>

    {% for message in outbox_message %}
        <div style="width: 18rem; border: 1px solid black; margin: 10px; overflox: hidden">
            <div>
                {% for user in users %}
                    {% if user.id == message.receiver %}
                        <h6>{{ user.email }}</h6>
                    {% endif %}
                {% endfor %}
                <p>{{ message.message }}</p>
            </div>
        </div>
    {% endfor %}
</div>
```

RUN THE PROJECT

In order to run the project just run `*composer install*` in the root folder. Then create the database with the following doctrine command: `*php bin/console doctrine:database:create*`.

Finally create the required tables from the provided migration with the following command: `*php bin/console doctrine:migrations:migrate*`.

If you want to include the sample data just run the provided .sql file into PHPMyAdmin to load them.