1

Symfony 4 - Quickstart CRUD generation

1.1 What we'll create

This quickstart is a set of steps to create a database-drive website from scratch, with web forms for CRUD (Create-Read-Update-Delete). You aren't expected to understand all this if you're new to Symfony, but hopefully you'll see how powerful using a framework can be, and see how object-orientede programming and databases tie together with website front ends to make creating powerful applications straightforward, when you have a clear idea of the features and user-interaction you wish your application to support.

Note: Much of what we are doing is documented in the **Doctrine** pages of the Symfony website, if you wanted to read more around this topic:

`- (https://symfony.com/doc/current/doctrine.html)[https://symfony.com/doc/current/doctrine.html]`

1.2 Software tools

You will need the following working on your computer:

1. the Symfony command line tool
2. The SQLite PHP database extension

TUDublin computers should already have the Symfony command line setup and working (we'll test it in the next step).

This worksheet will show you how to enable SQLite on a University computer (if it's not already setup).

1.3 Open a CLI terminal window

Many professional software tools run from the command line. A CLI is a Command Line Interface, in other words a text-based terminal window. On a Windows computer startup a `cmd` window. On a Mac run the `Terminal` application.

One aspect of working with a CLI terminal is that we need to `cd` (change directory) to the location on the computer's hard disk where we want to work. On University computers it's often easier to work on the `Desktop`, and later you can backup your project folder to a network drive, USB, or cloud disk like OneDrive.

1.3.1 Windows `cd` to Desktop in command window

Do the following to get to the `Desktop` folder at the command line in Windows:

1. start `cmd` terminal window
2. ensure you are on the `C:` drive, if not, change to the `C:` drive by entering `C:` and pressing `<RETURN>`
    - you should now see `C:>` as your command line prompt
3. change into the `Desktop` folder for your logged-on user, by typing the following

    ```
    C:> cd \Users\B11223344\Desktop
    ```

    - of course, you need to replay `B11223344` with your own student number
4. you should now be at your user's `Desktop` in the command line - so when you create a project folder you should see it on your desktop

1.3.2 Mac OSX `cd` to Desktop in Terminal window

Do the following to get to the `Desktop` folder at the command line in Windows:

1. start `Terminal` application
2. change into the `Desktop` folder for your logged-on user, by typing the following

    ```
    matt$ cd ~/Desktop
    ```

3. you should now be at your user's `Desktop` in the command line - so when you create a project folder you should see it on your desktop

1.4 Check computer has requirements to run Symfony

At the CLI terminal command line type:

```
symfony check:requirements
```

You should see something like the following appear - the importart bit is the message saying `Your system is ready to run Symfony projects`:

```
C:\Desktop> symfony check:requirements

Symfony Requirements Checker
~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> PHP is using the following php.ini file:
/usr/local/php5/lib/php.ini

> Checking Symfony requirements:
......................W.........
[OK] Your system is ready to run Symfony projects  <<<<<<< yay -Symfony is supported!!

Optional recommendations to improve your setup
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * a PHP accelerator should be installed
   > Install and/or enable a PHP accelerator (highly recommended).
```

Don't worry about any optional recommendations.

1.5 Enabling the SQLite PHP database extension (Windows)

If Symfony is working, you have PHP setup on your computer. However, the SQLite database extension may not be setup.

You can either work ahead, hoping it is setup, and fix it if you hit a problem when trying to create a database. Or you can check, and fix it now.

PHP extensions are already installed with PHP, but may not be activated. All we have to do is ensure there is no semi-colon character `;` at the beginning of the line 'extension=php_pdo_sqlite.dll'.

Do the following:

1. Use Notepad++ to open file `C:\php\php.ini`
2. Search (`<CTRL>-F`) for `pdo_sqlite`

3. If the line reads 'extension=php_pdo_sqlite.dll' remove the first character and save the changed file.
4. That's it - you have now enabled SQLite for PHP applications.

If you want to enable the SQLite database as well, then do the same for the line saying 'extension=php_pdo_sqlite.dll'.

1.6 Create new project named "student1"

At the CLI terminal let's tell the Symfony command line tool to create a new, website application skeleton for us, in a new folder to be nbamed `student1` (you can choose a different project-folder name if you wish):

1. At the terminal type the following:

   ```
   symfony new --full student1
   ```

2. You should see the following as Symfony downloads a skeleton project - it might take a few minutes depending on the speed of your computer, internet connection, and whether it's the first time you've done this (later times may use 'cached' remembnerd copies of some files, and so be quicker!):

   ```
   C:\Users\B11223344\Desktop> symfony new --full student1

   * Creating a new Symfony project with Composer
     (running omposer create-project symfony/website-skeleton student1)

   * Setting up the project under Git version control
     (running git init student1)

    [OK] Your project is now ready in C:\Users\B11223344\Desktop\student1
   ```

1.7 Open project folder `student1` in your IDE editor

At this point we should now open our new project folder in an IDE editor like PHPStorm. Since we are going to want to examine and edit the project code.

Do the following:

1. Start your IDE editor
2. Use your editor to open your Desktop project folder `student1`

3. Either open a terminal window in your IDE, or keep your terminal application window open, and `cd` into the `student1` folder

Figure 1.1 shows a screenshot of PHPStorm and our new skeleton project.

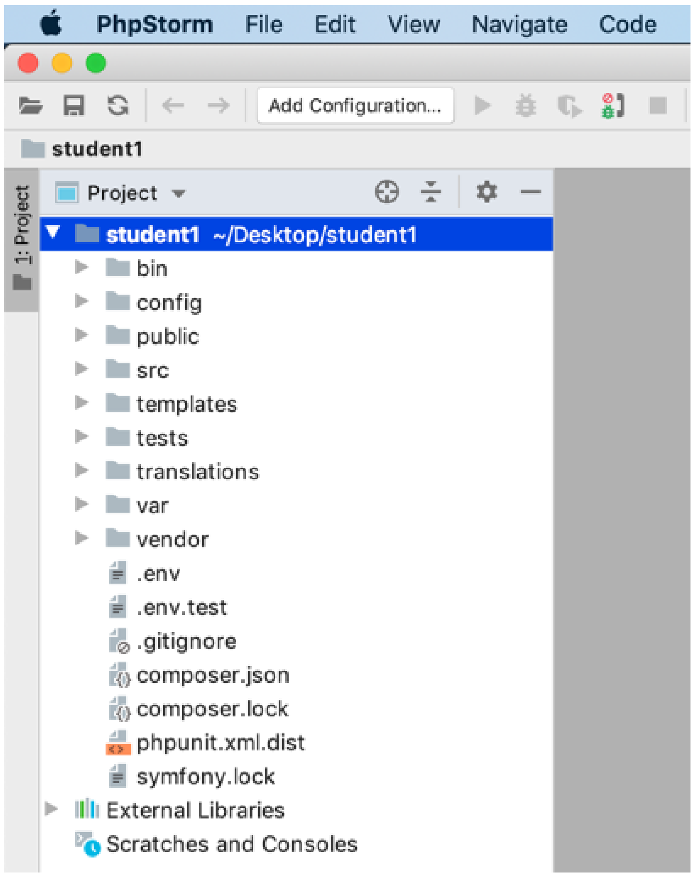


Figure 1.1: PHPStorm IDE new project.

1.8 Change project to work with a simple SQLite database

Symfony projects provide default settings to work with MySQL databases. But it's quicker to start working with an SQLite database file.

We need to add 1 line to the special `.env` environment settings file. In your IDE open file `.env` and fine this section about the **Doctrine** database settings (around line 20 in the settings file):

```
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/c
# For an SQLite database, use: "sqlite:///%kernel.project_dir%/var/data.db"
# Configure your db driver and server_version in config/packages/doctrine.yaml
```

```
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name
###< doctrine/doctrine-bundle ###
```

We need to add an extra line (to override the default `DATABASE_URL` variable):

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

Add this extra line so this section now looks as follows:

```
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/refe
# For an SQLite database, use: "sqlite:///%kernel.project_dir%/var/data.db"
# Configure your db driver and server_version in config/packages/doctrine.yaml
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name
###< doctrine/doctrine-bundle ###
DATABASE_URL=sqlite:///%kernel.project_dir%/var/data.db
```

1.9 Create the database

We can now use the command line tool to create the database for the project, using the command `php bin/console doctrine:database:create`:

```
> php bin/console doctrine:database:create
Created database /Users/matt/Desktop/student1/var/data.db for connection named default
```

We have now created an SQLite database file `var/data.db`. See Figure 1.2 shows a screenshot of the file in our editor file list.

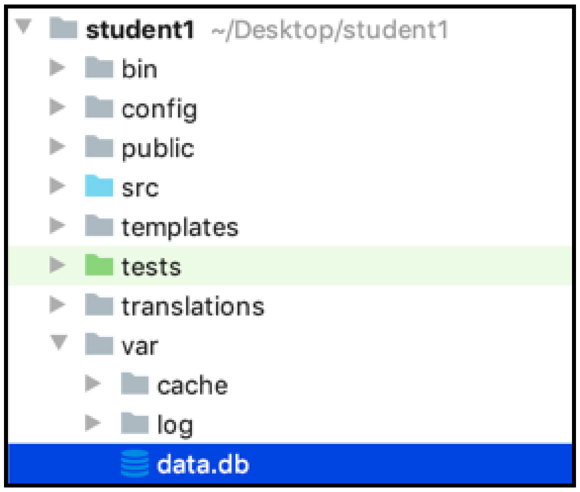


Figure 1.2: SQLite database file in project files.

1.10 Create Student class

Let's create a Student class and generate automatic CRUD web pages. Do the following:

1. At the command line type:

   ```
   php bin/console make:entity Student
   ```

   You should see the following:

   ```
   > php bin/console make:entity Student

   created: src/Entity/Student.php
   created: src/Repository/StudentRepository.php

   Entity generated! Now let's add some fields!
   You can always add more fields later manually or by re-running this command.
   ```

   - notice that it tells us that is has created 2 new classes `src/Entity/Student.php` and `src/Repository/StudentRepository.php`
     - `Student.php` is a simple class with private propetries and getters/setters, with special 'annotation' comments so these objects can map directly to rows in a database table...

2. Now we need to ask this console 'make' tool to add an integer `age` property for us:

   - we need to enter the property name `age`
   - we need to specify its data type `integer`
   - (to keep things simple) we don't mind if our properies start as null (just press `<RETURN>` for this question)
   - you should see the following when answering these questions at the `make` command tool prompt:

     ```
     New property name (press <return> to stop adding fields):
     > age

     Field type (enter ? to see all types) [string]:
     > integer

     Can this field be null in the database (nullable) (yes/no) [no]:
     >

     updated: src/Entity/Student.php
     ```

3. Now add a string `name` property:

- we need to enter the property name `name`
- we need to specify its data type `string` (since default just press `<RETURN>`)
- accept default string length of 255 (since default just press `<RETURN>`)
- can be nullable (since default just press `<RETURN>`)
- you should see the following when answering these questions at the `make` command tool prompt:

```
Add another property? Enter the property name (or press <return> to stop adding fields
> name

Field type (enter ? to see all types) [string]:
>

Field length [255]:
>

Can this field be null in the database (nullable) (yes/no) [no]:
>

updated: src/Entity/Student.php
```

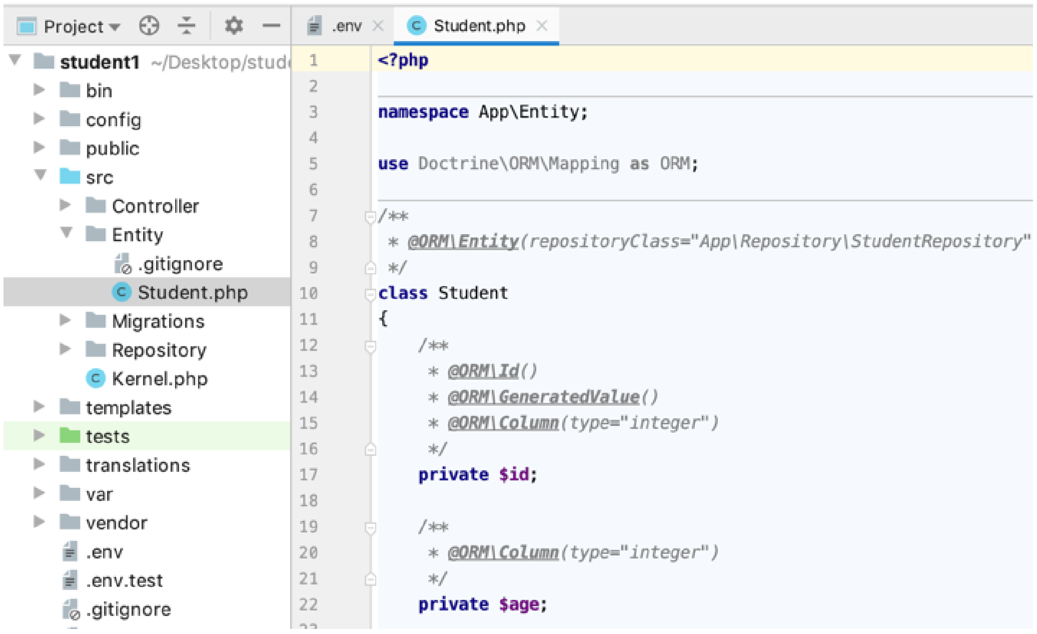
4. That's all our fields created, so just press `<RETURN>` to complete creation of our entity:

```
Add another property? Enter the property name (or press <return> to stop adding fiel
>

 Success!

Next: When you're ready, create a migration with make:migration
```

1.11 Take a look at the created entity class

Take a look at what's been created for us: `src/Entity/Student.php`. If you ignore the comments, mostly this is a class

See Figure 1.3 shows a screenshot of PHPStorm and our new class PHP code.



```php
<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\StudentRepository"
 */
class Student
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="integer")
     */
    private $age;
```

Figure 1.3: New Student.php entity.

1.12 Make 'migration' SQL to create database to correspond to our entity class

We can also use the 'make' command line tool to look at our classes and create the SQL to update our database create/update tables for storing the object data in tables and rows.

Enter the following at the command line `php bin/console make:migration`:

```
> php bin/console make:migration


  Success!


 Next: Review the new migration "src/Migrations/Version20190927055812.php"
 Then: Run the migration with php bin/console doctrine:migrations:migrate
 See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
```

If you look inside the newly created file you'll see a line like this showing the SQL generated to create a database table to match our `Student.php` class:

```php
$this->addSql(
    'CREATE TABLE student (
        id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        age INTEGER NOT NULL,
        name VARCHAR(255) NOT NULL
        )'
    );
```

1.13 Execute our 'migration' SQL to create/update database

Now let's tell Symfony to connect to the database and execute the migration SQL - to actually create the new `Student` table.

We need to enter the terminal command `php bin/console doctrine:migrations:migrate`:

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

                    Application Migrations


WARNING! You are about to execute a database migration that could result in schema changes a
Are you sure you wish to continue? (y/n)
```

At this point we must enter `y` to go ahead - saying we are happy for our database structure to be chagned by exectuing our migration SQL:

```
WARNING! You are about to execute a database migration that could result in schema changes a
Are you sure you wish to continue? (y/n)y

Migrating up to 20190927055812 from 0

  ++ migrating 20190927055812
     -> CREATE TABLE student (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    age INTEGER NOT NULL, name VARCHAR(255) NOT NULL)

  ++ migrated (took 64.4ms, used 18M memory)
  ------------------------
  ++ finished in 70.8ms
  ++ used 18M memory
  ++ 1 migrations executed
  ++ 1 sql queries
```

That's it - we have now created a table in out databasde to match our PHP entity class.

1.14 Generate the CRUD web form for class `Student`

Let's generate some HTML and PHP code for a web form to list and create-read-update-delete data from our database.

We need to execute this command to create that code `php bin/console make:crud Student`:

```
> php bin/console make:crud Student

 created: src/Controller/StudentController.php
 created: src/Form/StudentType.php
 created: templates/student/_delete_form.html.twig
 created: templates/student/_form.html.twig
 created: templates/student/edit.html.twig
 created: templates/student/index.html.twig
 created: templates/student/new.html.twig
 created: templates/student/show.html.twig

  Success!

 Next: Check your new CRUD by going to /student/
```

1.15 Run the Symfony web sever

Let's run the web server on our machine (`localhost:8000`) by entering terminal command `symfony serve --no-tls`.

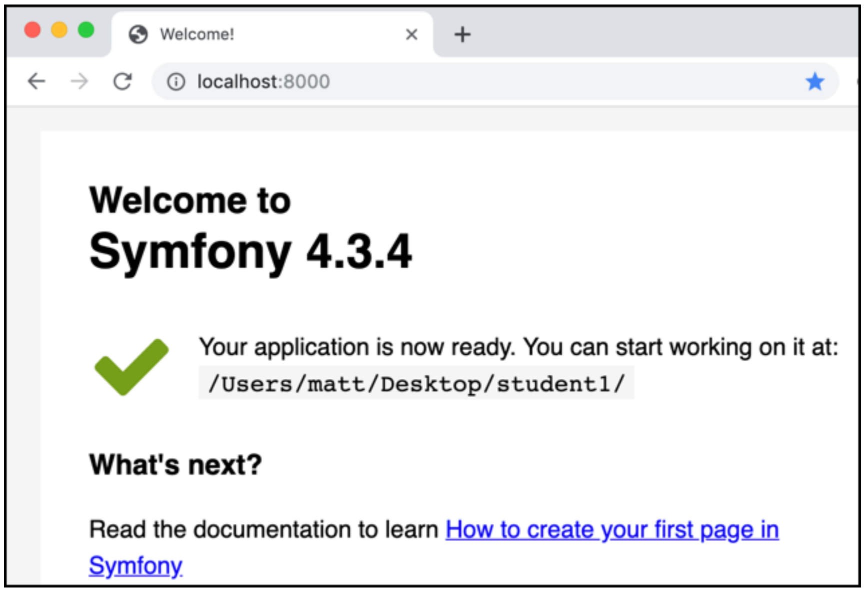
Note, you might get some warnings/info messages about version of PHP etc. - just ignore them!

```
> symfony serve --no-tls
Sep 27 07:22:19 |DEBUG| PHP    Using PHP version 7.3.8 (from default version in $PATH)
Sep 27 07:22:19 |INFO | PHP    listening path="/usr/local/php5-7.3.8-20190811-205217/sbin/php-fpm"
Sep 27 07:22:19 |DEBUG| PHP    started
Sep 27 07:22:19 |INFO | PHP    'user' directive is ignored when FPM is not running as root
Sep 27 07:22:19 |INFO | PHP    'group' directive is ignored when FPM is not running as root
Sep 27 07:22:19 |INFO | PHP    fpm is running, pid 98665
Sep 27 07:22:19 |INFO | PHP    ready to handle connections

 [OK] Web server listening on http://127.0.0.1:8000 (PHP FPM 7.3.8)
```

1.16 Visit the home page `localhost:8000`

Open a web browser and visit our website home page at `http://localhost:8000`.

Since we didn't create a home page, we'll see a default Symfony home page. See Figure 1.4 shows a screenshot of PHPStorm and our new class PHP code.



Welcome to
Symfony 4.3.4

Your application is now ready. You can start working on it at:
`/Users/matt/Desktop/student1/`

What's next?

Read the documentation to learn How to create your first page in Symfony

Figure 1.4: Default Symfony home page.

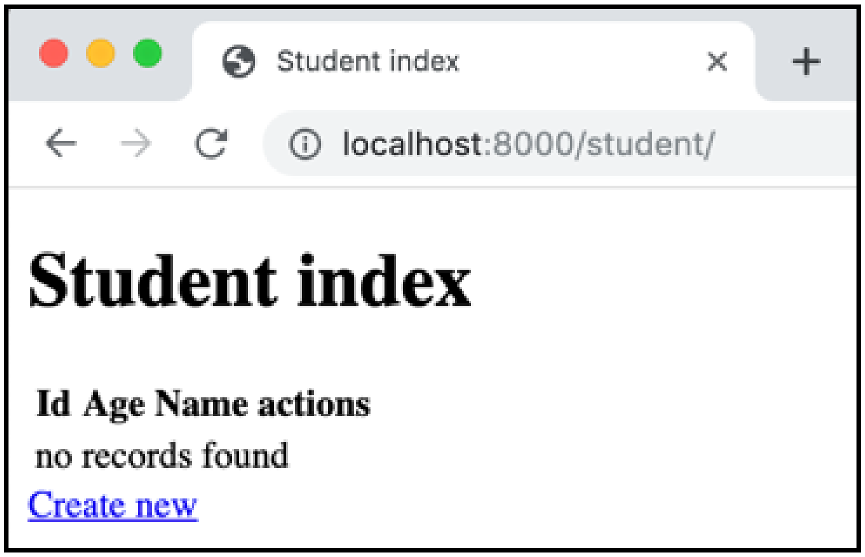
1.17 Visit our generated `Student` crud pages at `/student`

Let's visit our generated CRUD pages, these can be found by adding `/student` at the end of the URL. See Figure 1.5 shows a screenshot of PHPStorm and our new class PHP code.

Click `Create new` and add a student. Then try clicking edit, and change some values or delete it and create it again.

You should find you have a fully working web-based CRUD interface to your database.

See Figure 1.6 shows a screenshot of several students having been created (and yes, my grandmother did live to 96!).



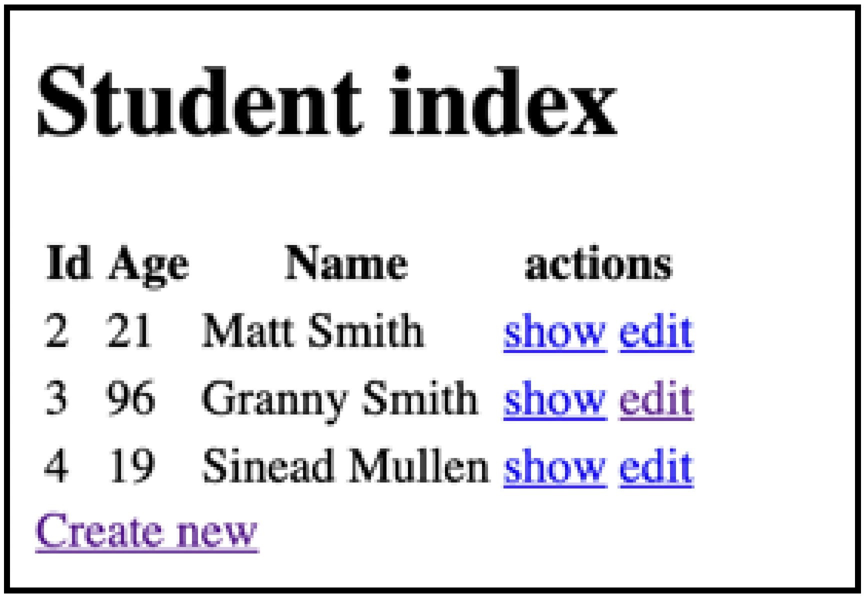
Student index

Id	Age	Name	actions

no records found

Create new

Figure 1.5: Student CRUD list page.



Student index

Id	Age	Name	actions
2	21	Matt Smith	show edit
3	96	Granny Smith	show edit
4	19	Sinead Mullen	show edit

Create new

Figure 1.6: Several students created with web CRUD.

1.18 Databases are persistent

Kill the Symfony web server at the command line by pressing `<CTRL>-C`. Then quit the PHPStorm IDE application.

You could also go have a cup of coffee, or perhaps shut down and restart your computer.

Then restart the PHPStorm editor, and restart the web server with `symfony serve --no-tls`.

Now open a web browser to URL `http://localhost:8000/student` and you should see that the students you created in your database are still there.

Any changes we make are remembered (persisted) as part of our `var/data.db` database file.

1.19 How to make it look nice with Bootstrap CSS

Let's add the Bootsrap CSS to our project - so it looks more professional.

Learn more about the Bootstrap 4 theme on the Symfony documentation pages:

- https://symfony.com/doc/current/form/bootstrap4.html

Do the following steps...

1.19.1 Configure Twig to use the Bootstrap theme

Well Symfony to generate forms using the Bootstrap theme by adding:

```
form_themes: ['bootstrap_4_layout.html.twig']
```

to `/config/packages/twig.yml` file. So this file should now look as follows;

```
twig:
    paths: ['%kernel.project_dir%/templates']
    debug: '%kernel.debug%'
    strict_variables: '%kernel.debug%'
    form_themes: ['bootstrap_4_layout.html.twig']
```

1.19.2 Add the Bootstrap CSS import into our base Twig template

The Bootstrap QuickStart tells us to copy the CSS `<link>` tag from here:

- https://getbootstrap.com/docs/4.1/getting-started/introduction/#css

into the CSS part of our `/templates/base.html.twig` Twig template. Add this `<link>` tag just before the `stylesheets` block:

```
    <!DOCTYPE html> <html>
    <head>
        <meta charset="UTF-8" />
        <title>MGW - {% block pageTitle %}{% endblock %}</title>

        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.m
        {% block stylesheets %}{% endblock %}

    </head>
   ...
```

1.19.3 Add the Bootstrap JavaScript import into our base Twig template.

The Bootstrap QuickStart tells us to copy the JS `<script>` tags from here:

- https://getbootstrap.com/docs/4.1/getting-started/introduction/#js

into the last part of the `<body>` element in `templates/base.html.twig` Twig template. Add these `<script>` tags just after the `javascripts` block:

```
    ...

    <body>
        {% block body %}{% endblock %}

        {% block javascripts %}{% endblock %}

    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7ab
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js" integrity="
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js" integrity="sha

    </body>
</html>
```

1.19.4 Adding elements for navigation and page content

Let's ensure main `body` content of every page is inside a Bootstrap element. We need to wrap a Bootstrap container and row divs around the `body` Twig block:

In file `templates/base.html.twig` replace line:

```
    {% block body %}{% endblock %}
```

with the following (i.e. the same linem, but with Boostrap `divs` wrapped around it):

```
    <div class="container">
        <div class="row">
            <div class="col-sm-12">

            {% block body %}{% endblock %}

            </div>
        </div>
    </div>
```

1.19.5 Full listing of updated `templates/base.html.twig`

Your websites **basic template** file (`templates/base.html.twig`) should now look like this:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Welcome!{% endblock %}</title>
        {% block stylesheets %}{% endblock %}
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/boot
    </head>
    <body>

        <div class="container">
            <div class="row">
                <div class="col-sm-12">

                    {% block body %}{% endblock %}

                </div>
            </div>
        </div>
        {% block javascripts %}{% endblock %}

        <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+9
        <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js" int
    </body>
```

```
</html>
```

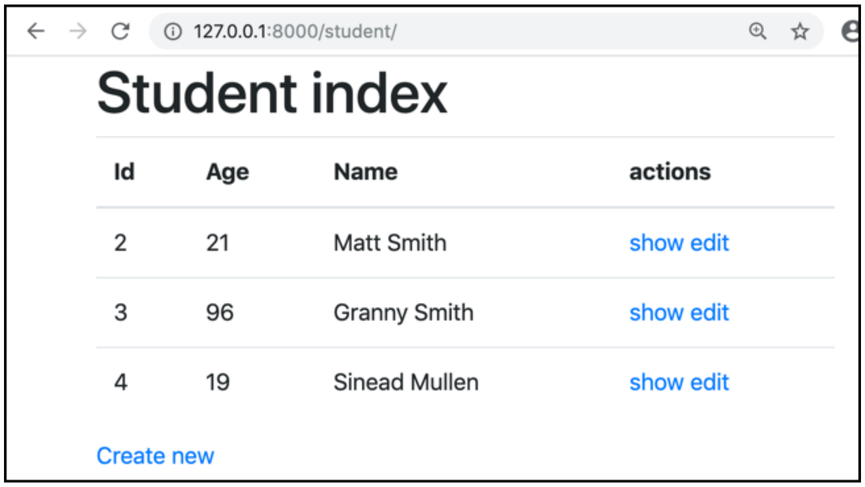
1.19.6 Remove CRUD and regenerate with Bootstrap

We now need to delete our generated CRUD, and re-genereate it. The generation logic will see the Bootrap theme, and style our web forms to look nice with appropriate Bootstap CSS.

1. Delete the old CRUD files:
    - Delete the `student` folder in `templates`
    - Delete the `StudentController.php` class file in `src/Controller`
    - Delete the `StudentType.php` class file in `src/Form`
2. Now re-generate the crud
    - run terminal command `php bin/console make:crud Student`

Run the web server again, visit `/student` and voila! we have Bootstrap themed web-forms!

See Figure 1.7 shows a screenshot of several students having been created (and yes, my grandmother did live to 96!).



Student index

Id	Age	Name	actions
2	21	Matt Smith	show edit
3	96	Granny Smith	show edit
4	19	Sinead Mullen	show edit

Create new

Figure 1.7: CRUD with Bootstrap theme.