README.md 19.2 KB

PWBOX

Pwbox is a custom file hosting service that offers a personal cloud storage with a simple and intuitive UI.

Prerequisits

In order to be able to follow with this exercise you are going to need a development environment with:

- A running server (nginx, apache...)
- A dedicated virtual host for this project
- PHP 7
- Composer

Note: we recommend the use of the **Homestead** virtual box developed by the Laravel team.

Introduction

This exercise has been divided in 13 blocks. Each one of them contains some mandatory and some optional functionalities.

If you just want to pass the exercise you only need to implement all the functionalities marked as *mandatory*. The functionalities marked as *optional* are for the groups that want to be able to apply for the interview and avoid the final exam.

The 13 blocks are explained in detail below.

BLOCK 1 - Structure

This first block describes how the project must be structured and some characteristics about the design and the layouts.

Mandatory

1. You need to define a base template **base.html.twig** that its going to be extended by all the views of the application. This template must define at least the following blocks:

- **Title:** The title of the current page (SEO)
- **Styles:** Where all the css files are going to be loaded
- **Header:** It need to include the main navigation menu
- **Content:** Includes the main content of every page

- **Footer**
- **Scripts:** Where all your `.js` files are going to be loaded

1. The base structure of the directories and files must be the same as the one used in the classes.

```
app
|__ settings.php
|__ routes.php
|__ dependencies.php
public
|__ assets
|__ index.php
src
|__ Controller
|__ Model
|__ View
var
vendor
```

1. All the routes must be defined in the `app/routes.php` file.
2. Every controller must only be associated with on route and vice versa (so its a 1-1 relation).
3. Every controller must be defined in a new class inside the `src/Controller` directory.
4. In order to implement the interfaces of your repositories you must use the package **doctrine/dbal**.
5. All the external libraries or services that you want to use in your project must be installed using composer and of course must be defined inside the `package.json` file.
6. The server name of the virtual host that associated to this project must be `pwbox.test` .
7. All the **css** and **js** code must be defined into external files inside the `public/assets` directory and be included in the **styels** and **scripts** blocks defined in the base layout.

Optional

1. Use one of the following css framework to stylize your application:
    1. Bootstrap (https://getbootstrap.com/)
    2. Bulma (https://bulma.io)
    3. Semantic UI (https://semantic-ui.com/)

Remember to put all the required files of the framework into the assets folder and to include them into the corresponding blocks of your templates.

BLOCK 2 - Landing page

This blocks describes some characteristics of the landing page of the application.

Acceptance criteria

As a non logged in user, when I access to the home page of the application I must be redirected to the landing page.

Mandatory

1. All the users without a valid session (not logged in) that access to the home page of the application `http://pwbox.test/` must be redirected to a cool landing page where all the characteristics of the product are well explained. As an example you can checkout the dropbox's landing page (https://www.dropbox.com/?landing=dbv2).

BLOCK 3 - Registration

This block describes the process of registering a new user into the system.

Acceptance criteria

As a non registered user I must be able to create a new account.

Mandatory

1. The form to register a new user must have the following inputs:

| Name | Description | Characteristics | | ---------- | ---------- | ---------- | | username | Identifies the user inside the application | Can only contain alphanumeric characters and a max length of 20 characters | | email | It will be used to sent all the notifications of the application | It must be a valid email | | birthdate | The birthadate of the user | It must be a valid date and must be well formatted | | password | | The length of the password must be between 6 and 12 characters and it must contain at least one number and one upper case letter | | confirm_password | Allow the users to double check the specified password | It must match with the value of the password field | | profile_image | The profile image of the user | This field is optional. The image size must be less than 500Kb. If the user does not upload any image, then a default placeholder must be used instead |

1. All the fields must be validated both in the frontend and in the backend. Of course the validations must be the same in both sides. The validation errors must be displayed below the corresponding invalid fields.

2. You must encrypt the user's password before storing it into the database. Feel free to use your preferred algorithm (explain a little bit your final choose in the readme).

3. In order to access to the register form you need to add a new link into the main navigation menu of the header. Once the user is logged in this link must be removed from the menu.

4. After finishing the registration the users must be redirected to the login page and a new folder must be created in the server and assigned to them. This root folder is where all the files and folders of the users are going to be stored.

Optional

1. If the registration has been successful an activation link must be sent via email to the user. Once a user clicks on the link then his account must be marked as active in the database and the user

must be logged in automatically.
2. If a user without an active account access to his profile page you must display a warning telling him to activate the account and display a link to resend the activation email.

Note: In order to sent the emails you can use the swiftmailer (https://packagist.org/packages/swiftmailer/swiftmailer) library and in order to get sample emails to use in the register forms you can check mailinator (https://mailinator.com/).

BLOCK 4 - Login

This block describes the process of logging a user into the system. After the use has been registered he must be able to start a new session using his specified credentials.

Acceptance criteria

As a registered user I must be able to start a new session using my credentials.

Mandatory

1. The users without an active session (not logged in) must see a **login** link in the main navigation menu that must redirect them to a cool login form.
2. The login form must contain a link to the register form.
3. The login form must contain two inputs, one for the **email** and one for the **password**. The validations of both fields must be the same as in the register form and must be done both in the frontend and in the backend side.
4. If both fields are valid then you need to check if the user exist in the database. If the user exists then he must be logged in and redirected to his dashboard. If the user does not exist then an error message muts be displayed in the login form warning the user about the problem.

Optional

1. The user must be able to use both the username or the email in the first input of the login form.

2. After the user has been logged in, his profile picture (or the default placeholder) must be shown in the navigation menu and it must be a link into the user's profile.

BLOCK 5 - Update profile

This block describes the process of updating the profile of an existent user.

Acceptance criteria

As a logged in user I must be able to check and update my information.

Mandatory

1. In order to access to this section of the application the users must be logged in. If a not logged in user try to manually access to this section the user must be redirected to an error page with a 403 status code.

2. If the user is logged in then a new link with the name **profile** must be displayed in the navigation menu of the header. If you have implemented the first optional point of the **BLOCK 3** you can use the image as the link to the user's profile page.
3. The users must be able to check all their information but they can only update the email, the password and the profile image. In order to update the password a new `confirm_password` field must be displayed below the current one.
4. Before updating any information of the user the same validations as in the register form must be applied. If something goes wrong then an error message must be displayed below the invalid fields. The validations must be done both in the frontend and in the backend side.
5. If the process is successful then the page must be reloaded in order to show the updated information.

Optional

1. If a user updates his profile image then the previous one must be removed from the server.
2. The process of updating the user information must be done using **AJAX** in order to avoid the second load of the page.

BLOCK 6 - Delete account

This block describes the process of removing the account of a registered user.

Acceptance criteria

As a registered user I must be able to remove my account and all my information must be successfully removed.

Mandatory

1. In the user's profile page you must display a button that allow the users to remove their accounts

Optional

1. Once the user clicks on the button to remove his account a popup alert must be displayed warning the user about the consequences of the action. This modal must contain two buttons, one for canceling the process and another one to continue it and remove the user's account.

BLOCK 7 - Roles

This block describes all the different roles that must be available in the application.

Acceptance criteria

As a registered user I must be able to take some actions in a file or a folder depending on my current role.

Mandatory

1. The system must have two different roles available: **admin** and **reader**.
2. All the roles of a user are associated to one root folder and to all its sub directories and files.
3. A user must be able to have multiple roles assigned.

4. The users with an *admin* role must be able to upload and remove files or folders from the root folder.
5. The users with a *reader* role must be able to check and download files from the root foler.

BLOCK 8 - User dashboard

This block decribes the characteristics and the structure of the user's dashboard. This dashboard is where all the *magic* happens. In this dashboard the user must be able to check t full list of all his files and folders and also add, rename and remove them.

Acceptance criteria

As a registered user I must be able to check all my files and folders and to take some actions with them.

Mandatory

1. Once the user access to his dashboard you must display a list containing only the first childs of the user's root folder.
2. All the item's of the list of type *folder* must contain a link to the the same dashboard that are going show the current content of the specified folder.
3. All the item's of the list of type *file* must contain a link to the current file location that will allow the users to download the current file.
4. The dashboard must contain two buttons, one to add a new folder and another one to uplodad a new file. The functionalities of these buttons are going to be explained in more detail in the blocks 9 and 10.

Optional

1. Detect a double click on the items of type *folder* instead of using the link to display its content.

BLOCK 9 - Folders

This block describes all the available actions related with the users folders.

Acceptance criteria

As a registered user with an *admin* role I must be able to add, rename and remove folders.

As a registered user with a *reader* role I must be able to check and download the content of the folders.

Mandatory

1. Every item of the dashboard's list of type *folder* must contain a button that allows the user to remove them. If the specified folder is not empty an error must be shown to the user and the action must fail.
2. Every item of the dashboard's list of type *folder* must contain a button that allows the user to rename the specified folder. Once the user click's on the button, a new modal must be shown. This modal must contain a little form with just one input containing the current folder's name and one button to submit the form.

3. Once the user clicks on the *new folder* button of the dashboard, a new modal must be shown. This modal must contain a little form with just one input where the users are going to introduce the name of the folder that they want to create and one button to submit the form. Once the user clicks on the submit button, the folder must be created inside the current selected folder of the dashbaord. So if the user is inside one of his folders the new folder must be created inside this folder.

Optional

1. If the user removes a folder that is not empty then all the content inside this folder must be removed as well.

BLOCK 10 - Files

This block describes all the available actions related with the users folders.

Acceptance criteria

As a registered user with an *admin* role I must be able to add, rename and remove files.

As a registered user with a *reader* role I must be able to check and download the files.

Mandatory

1. Once the user clicks on the *new file* button of the dashboard, a new modal must be shown. This modal must contain a little form with just one input of type *file* that are going to allow the users to select a file from their computers. Once the user clicks on the submit button, the file must be uploaded inside the current selected folder of the dashbaord.

2. The available file types are listed below:

    1. PDF (.pdf)
    2. JPG, PNG and GIF
    3. MARKDOWN (.md)
    4. TEXT (.txt)

If a user try to upload a file with a non valid extension an error must be shown warning the user about the available file types.

1. The maximum size allowed for a file is 2Mb. If a user try to upload a file with a larger size than 2Mb then an error must be shown warning the user.

2. After the file has been successfully uploaded into the system the current page should be reloaded in order to show the new file on the list.

Optional

1. The form to add a new file must allow the users to drop multiple files in it that are going to be uploaded concurrently. As a recommendation you can check the following js library (https://fineuploader.com/demos.html).

BLOCK 11 - Share

This block describes all the functionalities related with the process of sharing a folder with a user.

Acceptance criteria

As a logged in user with an *admin* role I must be able to share a folder with another user of the system.

Mandatory

1. You need to add another button in the users dashboards that allows them to share the current selected folder with another user of the system. Once the user clicks on the button a new modal must be shown. This modal must contain just one input to introduce the email of the user who is going to receive the invitation.
2. If the email introduced by the user is not from a registered user of the system, the process must fail and an error must be shown warning the user.
3. By default all the users that have been invited to a folder are going to have the *reader* role.
4. You need to add another link with the name *shared* into the main navigation menu. This link must redirect the users to another view. This view must display a list with all the shared folders where the user has been invited. Every item of the list must contain a link to the shared folder.

Optional

1. The modal to share a folder with another must contain an input of type *select* that allows the users to choose the role that its going to be assigned to the other user.

BLOCK 12 - Storage capacity

This block describes how the storage capacity should be handled.

Mandatory

1. Once a user is registered into the system 1Gb of storage space must be assigned to them
2. Every time a user upload a new file into one of his folders, his available remaining space must be recalculated
3. If the user try to upload a new file with a larger size than the remaining space then the operation must fail and a warning must be shown to the user

BLOCK 13 - Notifications -- Optional

This block describes all the different kind of notifications that are available in the system.

Note: This block is entirely **optional**.

Acceptance criteria

As a registered user that has shared a folder to another user I must be able to receive some notification when certain actions happens in the system.

Mandatory

1. Every time a user that has been invited to an specific folder with an *admin* role, renames, deletes or add a new file in it, a notification must be sent via email to the owner of the folder.

2. You need to add a new link with the name *notifications* into the main navigation menu that will redirect the users to a page where they will be able to see a list with all the received notifications.

REQUIREMENTS AND DELIVERY

In order to pass the exercise you need to implement all the functionalities marked as *mandatory* that have been distributed along the 13 blocks. By doing this you aim to obtain the maximum qualification but you will need to go to the final exam.

In order to avoid the final exam, you need to implement also all the functionalities marked as *optional*. Remember that the interviews are individual and mandatory to all the members of the groups.

The last day to deliver the exercise is in May 18 at 23:00pm. All the exercises delivered before this date are not going to be accepted. The requeriments of the delivery are listed below:

1. It need to be hosted in a private repository in **Github**, **Gitlab** or **Bitbucket**. Remember to share the URL of your project to the professors.
2. The name of the repository must follow the following pattern `PW_PWBOX_GROUP_X` where the X should be replaced with the group number.
3. Every member of the team needs to push at least 10 commits.
4. You need to push a named tag witht the name `final_delivery`. Remember that the last commit associated to this tag is the one that its going to be used to rate the exercise, so check your code before pushing it to the remote.
5. You need to add a brief **README** explaining all the things that you think that we the professors must know in order to run and test your application. At the end of the **README** specify the hours that every member of the group have invested in implementing the exercise. Be as honest as possible :)
6. Finally remember to apply to one of the scheduled interviews that you can find in the Estudy.

Note: Any copy of an exercise will involve that both groups (the one that copies and the one that allows the copy) will suspend the exercise and you won't be allowed to re-exam.