

# Photo gallery

---

Nume: Badea Ștefania-Andreea

## Contents

Photo Gallery - Photo album management application .....	3
1 Database.....	4
2 Features.....	6
3 Application presentation .....	9
3.1 SignUp.....	9
3.2 SignIn .....	10
3.3 MainPage .....	11
3.4 AddImage .....	13
3.5 Albums .....	13
3.6 AlbumView .....	15
4 Bibliography.....	17

## Photo Gallery - Photo album management application

Photo Gallery is an application that manages photo albums in which users can upload and view their photos.

To make the application I used the MEAN stack. MEAN is a full-stack JavaScript solution that helps us create robust, flexible, powerful, fast, extensible, and easy-to-maintain web applications. MEAN is a set of Open Source components that together provide solutions to build dynamic web applications; starting from the top (front-end) to the bottom (database)[1].



Figure 1. Logo used for MEAN stacks

MEAN is composed of:

- **MongoDB:** MongoDB is a popular, open source NoSQL database that provides a way to store data in JSON format. With MongoDB, we get flexibility and scalability. MongoDB uses collections and documents instead of tables and tuples.
- **Express:** Express is an open source web application framework for Node that builds fast, flexible, robust and secure web applications and APIs.
- **Angular:** Angular is referred to as the framework for the client side and is used in the development of front-end web with high efficiency and potential to develop high performance web applications.
- **Node:** Node.js is a JavaScript execution environment that runs the back-end application. Node is the backbone of the MEAN stack.



MEAN development benefits and services:

1. JavaScript language: No need to learn a separate language.
2. Open source framework: We can create dynamic websites and applications.
3. Flexibility and scalability: MEAN-based applications can be programmed more easily using code reuse
4. Better and faster performance: Scalable and event-driven architecture delivers the best performance.
5. Security architecture: High security applications based on security guidelines and practices.
6. Cloud integration: Cloud solutions save time, money and space.
7. NoSQL: Migration management becomes easier with NoSQL.

## 1 Database

Within the application I used MongoDB as a database server. The Photo Gallery database contains the following collections:

- **users:** users are permanently stored here. The collection contains a unique id (`_id`), username (`firstName`), last name (`lastName`), email address (`email`), password (`password`), a salt (`saltSecret`). A salt is a string of random data used in the password encryption function.

- **tempusers:** This collection temporarily stores users who have not yet verified their email address. The collection contains a unique id (`_id`), username (`firstName`), first name (`lastName`), email address (`email`), password (`password`), a salt (`saltSecret`), lifetime (`expireAt`, set to 30 minutes ), verification code (`verificationToken`).
- **uploads.chunks:** stores binary pieces of the image. This collection is generated by GridFS. The connection of this collection with the `uploads.files` collection is made through the image id (`files_id` in `uploads.chunks` and `_id` in `uploads.files`).
- **uploads.files:** stores image metadata. This collection is generated by GridFS. The collection stores the date the image was uploaded (`uploadDate`), the image name (`filename`), the image size (`length`), the size of each piece (`chunkSize`) and the image type (`contentType`).
- **images:** this collection connects the image to the user. The collection contains an unique id (`_id`), image name (`filename`) and user id (`user_id`).
- **allalbums:** all user albums are stored in this collection. The collection consists of: an unique id (`_id`), the description of the album (`description`), the name of the album (`name`) and the id of the user who owns it (`user_id`).
- **albums:** this collection links an image to an album. The collection stores the image name (`filename`), the album name (`albumname`), a unique id (`_id`) and the user id (`user_id`).

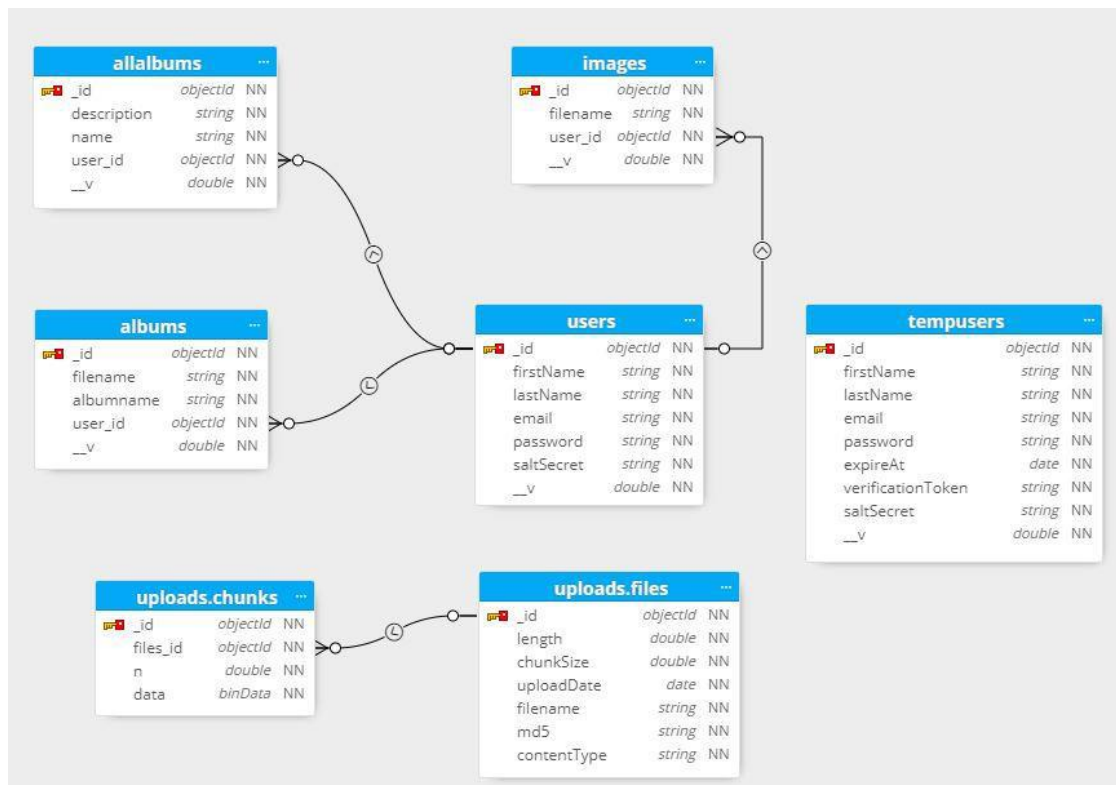


Diagram 1. Database

## 2 Features

Diagram 2 shows the functionalities of the application.

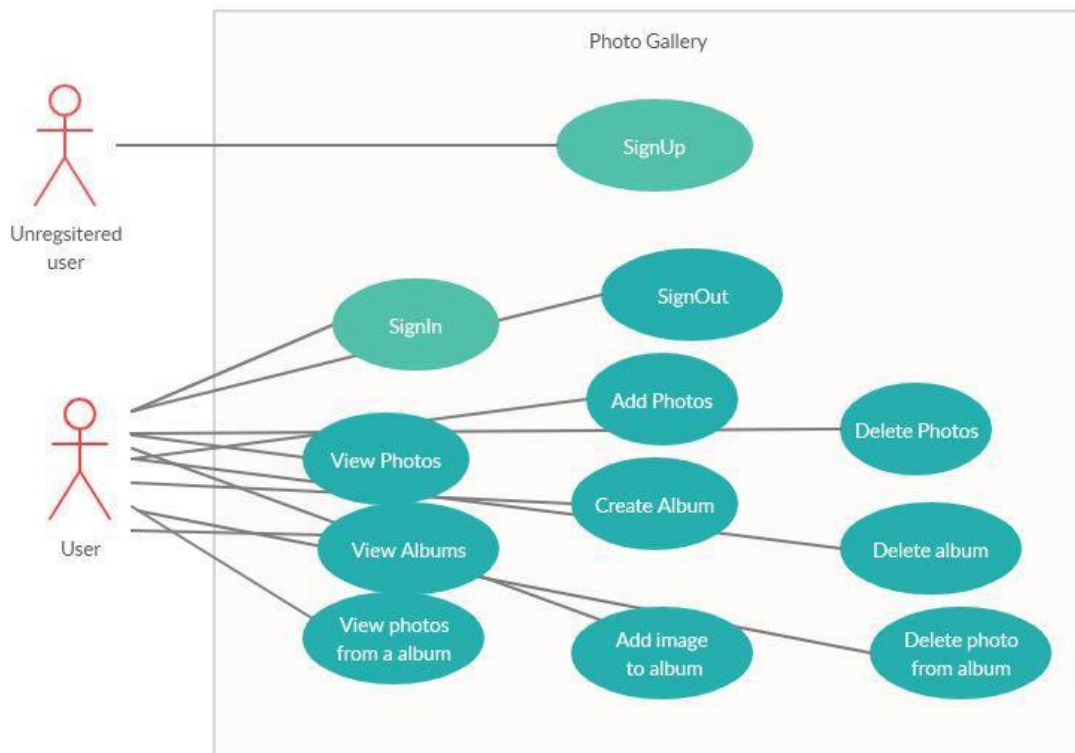


Diagram 2. Use Case Diagram

*User registration.* After completing the form on the SignUp page, the data will be temporarily stored in the tempusers collection. Before being saved, the password is encrypted using the bcrypt function. Bcrypt is a password encryption feature based on the Blowfish cipher. This is an adaptive function that uses a random data string called a salt.

```

tempUserSchema.pre('save', function(next) {
  bcrypt.genSalt(10, (err, salt) => {
    bcrypt.hash(this.password, salt, (err, hash) => {
      this.password = hash;
      this.saltSecret = salt;
      next();
    })
  })
});

```

Before the data is saved, it is checked if the entered email address exists in the users collection. If it exists, the message "*Published email address found.*" will be sent as a reply. if it does not exist, a verification code will be generated which will be sent to the user by email and the data will be saved in the tempusers collection.

```

newUser.verificationToken = randomstring.generate(8);

```

To complete the registration, the new user will need to confirm that the email address belongs to them. This is done by entering in the form the verification code received by email. If it matches the one in the database, then its data will be moved from the `tempusers` collection to the `users` collection. Otherwise, the message “*Code not found!*” will be sent.

*User authentication.* For authentication, I used `passport`, a means of authentication for Node. Passport uses strategies for authentication requests. The strategy used in the application is the `passport-local` strategy that allows authentication using a username and password. Because the email address is unique, I used it instead of the username. When a user wants to log in, `passport-local` checks if the email address appears in the database and if the password is correct. If one of the two conditions is not met, a message will be sent specifying this. Otherwise, the authentication process continues.

The next step is to create a JWT (JSON Web Token) token, with which the user proves to be authenticated. The token is generated using the following function:

```
JWT_SECRET = "sEcReT#@123";
JWT_EXP = "1440m";
.....
userSchema.methods.generateJwt = function() {
  return jwt.sign({ _id: this._id }, JWT_SECRET, {
    expiresIn: JWT_EXP
  });
}
```

*Add a new photo.* To add images to the database, I used GridFS. GridFS is a MongoDB specification for storing and retrieving files whose BSON document exceeds the size of 16MB. Instead of storing the file in a single document, GridFS splits the file into 255kB pieces and stores each piece as a separate document. GridFS uses two collections to store files. One collection stores pieces of files, and the other stores file metadata.

When the user uploads an image, it will be stored in the `uploads.files` and `uploads.chunks` collections and the image name and user id will be stored in the `images` collection. Before an image is uploaded, its name will be encoded so that it is not repeated.



*View all photos.* We have described this functionality in two functions. One of the functions searches the `images` collection for the documents that contain the user id and returns them in pages of 10. The second function is used to display an image. I used two functions because the display function can display a single image.

*Delete a photo.* When a user wants to delete a photo, it will be deleted from the `uploads.files`, `uploads.chunks` and `images` collections, but also from `albums`.

*Add a photo to an album.* This action is performed by a function that creates a new document in the `albums` collection, taking the name of an image and the name of an album. If the image has already been added, the function will send the message "*Duplicate file found.*"

*Viewing albums.* A user can list their created albums. The system calls a function that returns the name and description of all albums owned by the user.

*Creating a new album.* To create a new album, the user provides a name and description for the album that will be saved in the `allalbums` collection.

*Deleting an album.* If a user chooses to delete an album, both the album in the `allalbums` collection and the documents in the `albums` that contain the album name will be deleted.

*View photos from an album.* For this functionality, a function is used that returns all documents in `albums` that contain the album name. This function returns 10 page images.

*Delete a photo from an album.* Deleting an image from an album consists of deleting the document from the `albums` that contains the image name and the album name.

### 3 Application presentation

#### 3.1 SignUp

The SignUp page (Figure 2) is the page used to register new users. This page contains a form where users will enter their first name, last name, email and password. The `First name` and `Last name` fields may remain blank, and the `Email` and `Password` fields are required. The password cannot be less than 6 characters long.

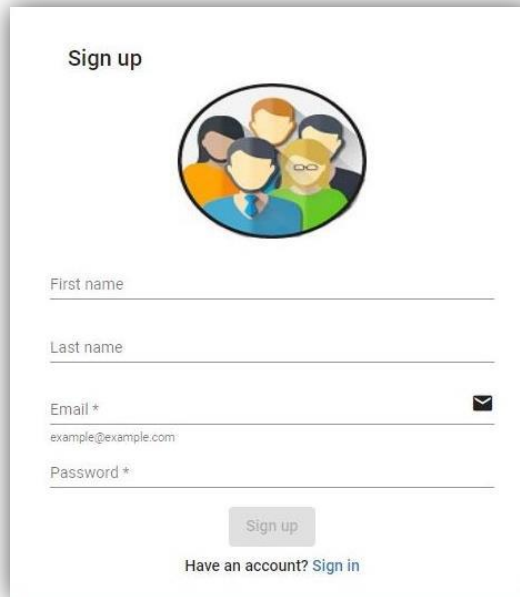
A sign-up form titled "Sign up" with a circular icon of four people. The form contains four input fields: "First name", "Last name", "Email \*" (with an email icon and the placeholder "example@example.com"), and "Password \*". A "Sign up" button is at the bottom, and a link "Have an account? Sign in" is below it.

Figure 2. SignUp page

The Sign Up button is only valid after the correct fields or the entire form have been filled in correctly. If the email you entered already exists in the database, a message specifying this will be displayed and the form will be reset. If the email is not in the database then an email will be sent to that address with a code. After pressing the Sign Up button, the dialog box in Figure 3 will open. The code received by email will be entered in this window. If it is not entered correctly the message *'Code not found'* will be displayed and the window will remain open so that a correct code can be entered. When the code is entered correctly, the message *'Saved successfully'* will appear and the SignIn page will load.

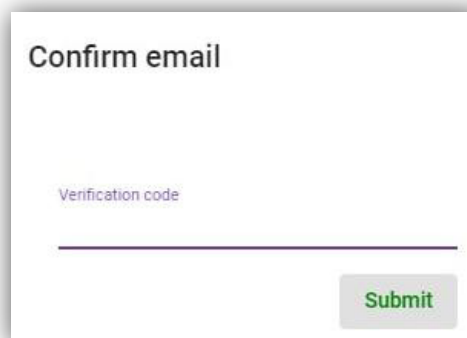
A "Confirm email" window with a "Verification code" label and a text input field. A "Submit" button is at the bottom right.

Figure 3. Confirm email window

### 3.2 SignIn

The SignIn page (Figure 4) is the page through which a user logs in. This page contains a form in which the user enters his email address and password. If the email address is not found in the

database, a message will appear specifying this and the form will be reset. If the password is incorrect, an appropriate message will be displayed and the form will remain unchanged.

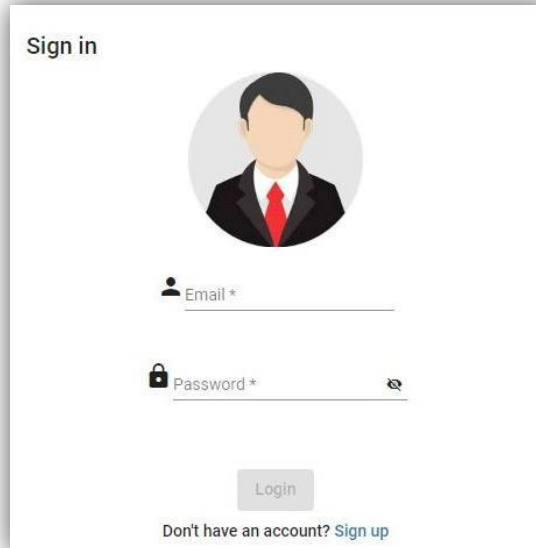
A sign-in form with a white background. At the top left, it says "Sign in". In the center is a circular placeholder for a user profile picture, showing a generic person in a suit. Below the profile picture are two input fields: "Email \*" with a person icon and "Password \*" with a lock icon. To the right of the password field is a small eye icon for toggling visibility. Below the input fields is a grey "Login" button. At the bottom, it says "Don't have an account? Sign up" with "Sign up" as a blue link.

Figure 4. SignIn page

From this page, you can navigate to the SignUp page using the link under the Login button.

### 3.3 MainPage

The MainPage page is the page where all the user's photos are displayed. This page can only be accessed by a logged in user.

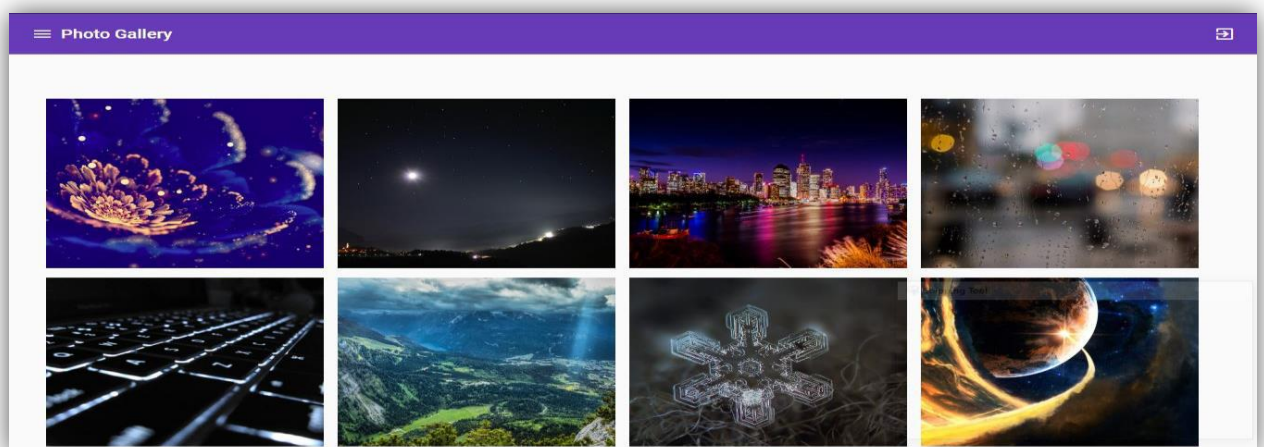


Figure 5. MainPage page

The user's photos are displayed in the gallery, 10 pages each, the oldest on the first page. The page contains a toolbar, photo gallery and two navigation buttons.

The toolbar is created using the following code sequence:

```
<mat-toolbar color="primary">
  <button mat-icon-button (click)="sidenav.toggle()">
    <mat-icon>menu</mat-icon>
  </button>
  <span>Photo Gallery</span>
  <span class="spacer" style="flex:1 1 auto"></span>
  <span class="toolbar_icon">
    <button (click)="logout()" mat-icon-button>
      <mat-icon>exit_to_app</mat-icon>
    </button>
  </span>
</mat-toolbar>
```

When an image is clicked, a menu will open. It contains two buttons: one for deleting an image and one for adding the image to an album. The following code sequence is used to generate the menu:

```
<mat-menu #menu="matMenu">
  <button mat-menu-
item (click)="openAddToAlbumDialog(images[i].filename)">
    <mat-icon>add</mat-icon>
    <span>Add to album</span>
  </button>

  <button mat-menu-item (click)="removeImage(images[i].filename)">
    <mat-icon>delete</mat-icon>
    <span>Delete image</span>
  </button>
</mat-menu>
```

The Add to album button will open a window of a selectable album.

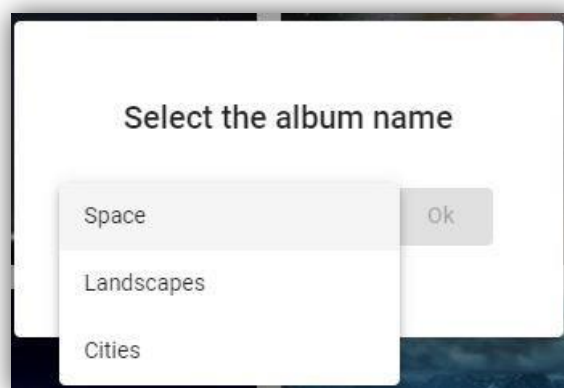


Figure 6.Window for album choosing

The toolbar contains a navigation menu and a SignOut button. All application pages, except SignIn and SignUp pages, have the same toolbar. The navigation menu consists of 3 buttons: New Image leading to the AddImage page, My photo leading to the MainPage (inactive for this page) and My albums leading to the album page. This menu remains the same for the MainPage and AddImage pages. The My Photos button has bold text which means the user is on that page.

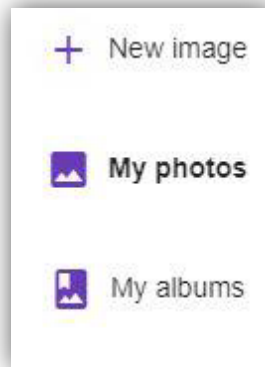


Figure 7. Navigation menu

### 3.4 AddImage

Through the AddImage page, the user adds new photos to the gallery. The page consists of a toolbar and a form for adding images, although it can upload only one image at a time.

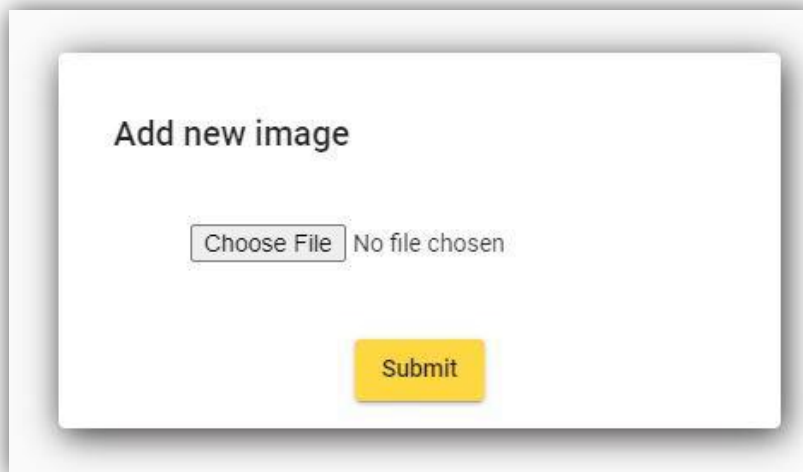
A form titled 'Add new image' with a white background and a subtle shadow. It contains a 'Choose File' button, a text label 'No file chosen', and a yellow 'Submit' button.

Figure 8. Form for adding a new image

### 3.5 Albums

This page displays all of a user's albums, displayed as a table. The table contains 3 columns: one for the button that opens the album menu, one with the album name and one with the album description, if any.



	Name	Description
⋮	Space	Photos of the space
⋮	Cities	
 Delete album  See album	Lanscapes	

Figure 9. Table with albums

The album menu consists of the See Album button that navigates to the album photos page and the Delete Album button that deletes the album.

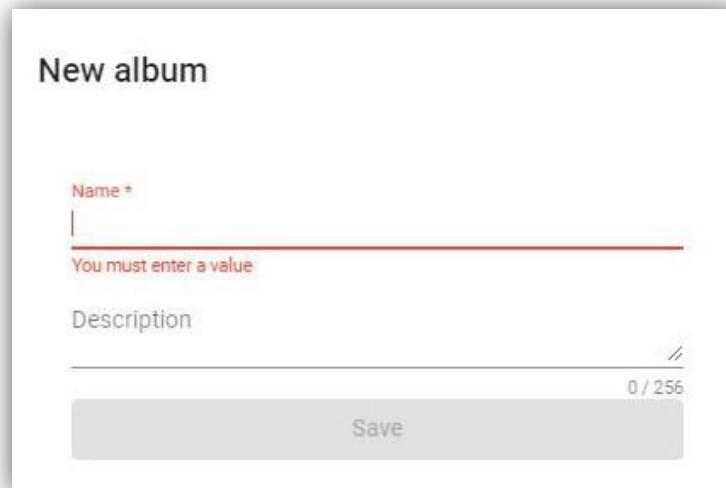
In the navigation menu of this page, the Add Image button is replaced by the New Album button. This button opens a window for creating a new album. The following code sequence corresponds to this window.

```
<mat-dialog-content>
  <form [formGroup]="myform" fxLayout="column" (ngSubmit)="newAlbum(myform.controls['name'].value, myform.controls['description'].value)">
    <mat-form-field>
      <input formControlName="name" matInput placeholder="Name" required>
      <mat-error *ngIf="myform.controls['name'].errors?.required">You must enter a value</mat-error>
    </mat-form-field>

    <mat-form-field>
      <textarea #message formControlName="description" matInput placeholder="Description"></textarea>
      <mat-error *ngIf="myform.controls['description'].errors?.maxlength"></mat-error>
      <mat-hint align="end">{{message.value.length}} / 256</mat-hint>
    </mat-form-field>
    <button align="end" mat-raised-button color="primary" type="submit" [disabled]="myform.invalid">Save</button>
  </form>
</mat-dialog-content>
```

```
</form>  
</mat-dialog-content>
```

The window contains a form in which the Name field is a required element and the Description field is optional. The Save button cannot be pressed until the Name field is filled.



New album

Name \*

You must enter a value

Description

0 / 256

Save

Figure 10.Window for creating a new album

### 3.6 AlbumView

Photos from an album can be viewed on this page. This page consists of a toolbar and a card (Figure 4.11). The card has the title of the album name. The body of the card consists of image cards, arranged in 4 columns for large screens, the number of columns decreasing with the screen size.

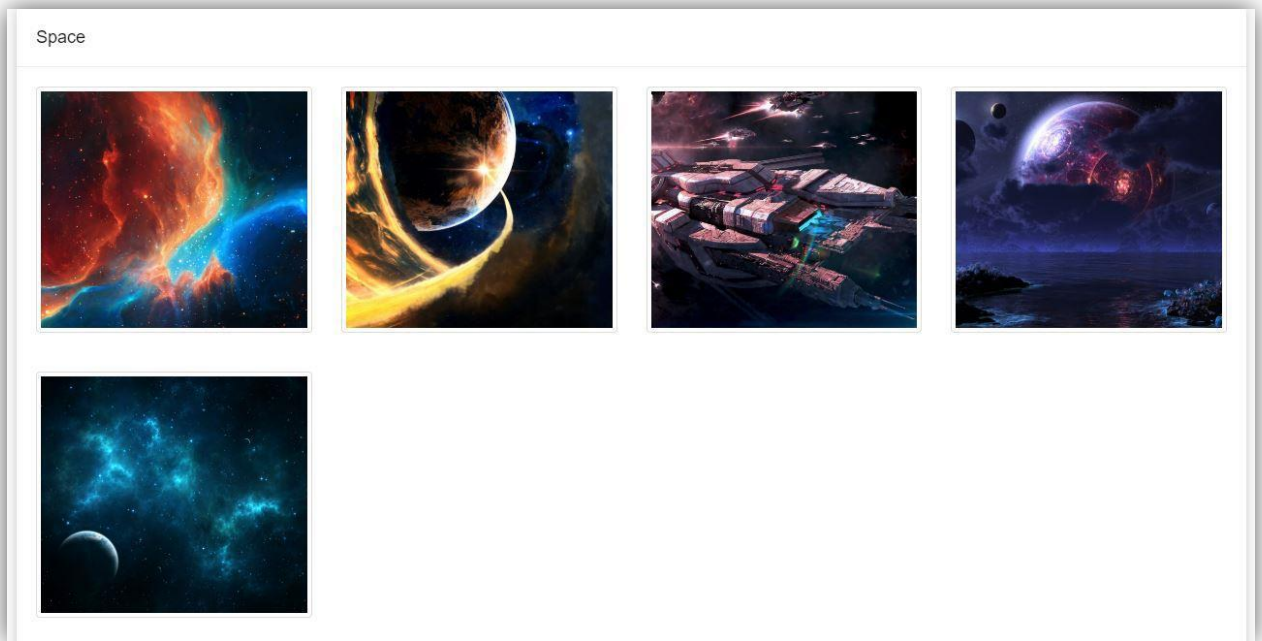


Figure 11.AlbumView page

When you press an image, a menu will appear (Figure 11) containing a button to remove that image from the album.

```
<mat-menu #menu="matMenu">
  <button mat-menu-item (click)="removeImage(images[i].filename)">
    <mat-icon>delete_outline</mat-icon>
    <span>Remove from album</span>
  </button>
</mat-menu>
```

For this page, the toolbar remains the same as in the other pages, but the navigation menu has only two buttons that lead to the MainPage and Albums pages.



## 4 Bibliography

- [1] K.Asp, "Medium," 2019 May 29. [Online]. Available: <https://medium.com/@kiran.asp12/mean-stack-development-d773e53a6d76>.