

Projektabschlussbericht MMS - Meme-Portal

Thema

Das Thema dieses Projekts ist primär die Bearbeitung von Bildern in einem Setting eines kleinen Portals zur Verteilung von selbst erstellten Bildern und Memes. Dabei kann im Editor aus bereits vorhandenen Meme-Templates ausgewählt, aber auch auf Basis von eigenen Bildern am lokalen Rechner gestartet werden.

Dieses Projekt wurde mit modernsten Technologien, wie einerseits dem plattformunabhängigen Framework .NET Core 5.0 C# für das Backend und andererseits mit Angular 2 Version 11.2 für das Frontend als Single-Page-Application, entwickelt.

Die im Frontend verwendeten Komponenten zur visuellen Darstellung des Contents basieren auf den Frameworks Devextreme und Angular Material, beziehungsweise wurde für den Bildeditor der TOAST UI Image-editor als Grundlage verwendet.

Die Sourcecodeverwaltung erfolgte in einem öffentlichen Github-Repository und die Aufgabenformulierung/-zuteilung mittels Github-Issues.

Team

Das Team besteht aus folgenden Personen mit den jeweiligen Kompetenzbereichen:

- Kaan Baylan
 - Frontend
 - Bildeditor
 - Upload bearbeiteter Bilder in das Portal
 - Abschlussbericht
 - Präsentation
- Julian Garn
 - Backend und Frontend
 - Userverwaltung im Backend
 - Requestseite
 - Profileinstellungen im Frontend bearbeiten
 - Bildeditor
- Jakob Rumpelsberger
 - Backend und Frontend
 - Verwalten der eigenen Bilder
 - Suchfunktion
 - Adminbereich
- Elias Schächl
 - Backend und Frontend
 - Registrierungsseite für neue Benutzer
 - Login-Seite zur Authentifizierung bei geschützten Bereichen
 - Adminbereich
- Simon Primetzhofner
 - Backend, Frontend und Projektleitung
 - Layout der Applikation
 - Verwalten der Bilder
 - Suchfunktion
 - Portalkomponente zur Darstellung der Bilder

Projektbeschreibung

Das beinhaltet folgende **Komponenten**:

- .NET Core Backend
 - Usermanagement für das Portal
 - JWT-Tokens
 - Verwalten der erstellten Bilder (CRUD)
 - Bereitstellung der Bilder und deren Metainformationen
 - Abspeichern der Daten in einer SQL-Server Datenbank
- Angular Webanwendung
 - Darstellen aller vorhandenen Bilder
 - Suchen von Bildern durch deren Tag
 - Verwalten der eigenen Bilder von eingeloggten Benutzern
 - Login bzw. Registrierung von Benutzern
 - Admin-Funktionalität:
 - Inhalte entfernen, Tag ändern, ...
 - Bildbearbeitung:
 - Eigene Sektion im Portal
 - Bearbeiten der Bilder (diverse Funktionen)
 - Direkter Upload ins Portal möglich
 - Einbinden vorhandener Templates

Funktionalitätsdefinition:

Unangemeldete Benutzer

- Anzeigen aller Bilder im Portal
- Suchen von Bildern

Angemeldete Benutzer

- Funktionalität von unangemeldeten Benutzern
- Bilder im Bildbearbeitungstool erstellen und bearbeiten
- Upload ins Portal
- Verwalten der eigenen Bilder
- Verwalten des eigenen Accounts
- Anfragen an Admins richten

Admin

- Inhalte entfernen
- Tag von Bildern ändern
- Anfragen von Benutzern beantworten

Architektur

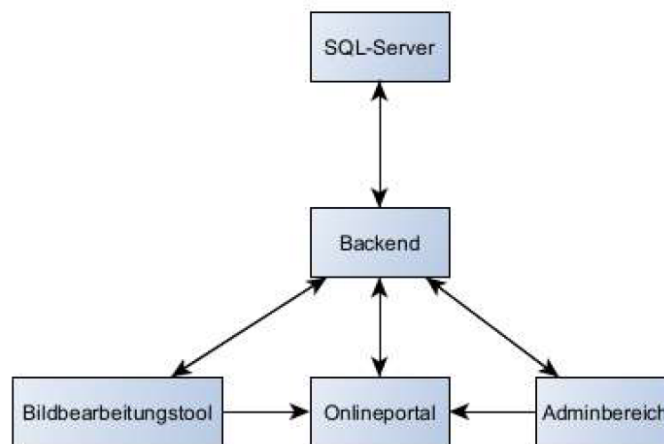


Abbildung 1 Architektur

SQL-Server:

wird als Datenbank unseres Portales verwendet. Hier werden in verschiedenen Tabellen die erstellten Bilder, Benutzer unserer Seite, Metadaten der Bilder und die Requests an den Admin (siehe Datenbankmodell) abgespeichert.

Backend:

ist eine .Net-Core Anwendung und bietet eine REST-API an welche die Kommunikation zwischen SQL-Server und Frontend übernimmt.

Frontend:

- **Online-Portal:**

ist unsere Angular-Anwendung, welche für

- den Login und Registrierung,
- das Erstellen, Anzeigen und Verwalten der Memes,
- den Admin-Bereich

zuständig ist.

- **Bildbearbeitungstool:**

basiert auf dem TOAST UI Image-Editor und wird für die Erstellung der Memes verwendet. Hier können angemeldete Benutzer

- ein von uns bereitgestelltes Meme-Template verwenden
- oder selbst ein eigenes Bild hochladen

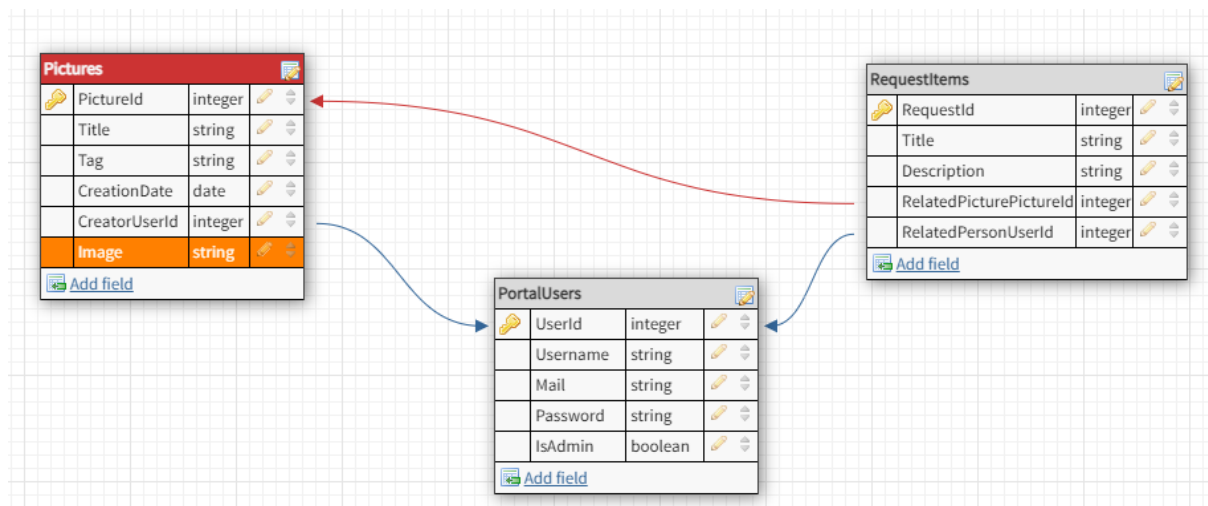
Das ausgewählte Bild kann dann noch durch Einfügen von Text, Bildern, Stickern usw. bearbeitet werden.

Nachdem der Benutzer zufrieden ist mit seinem erstellten Meme, kann er den direkt aus dem Image-Editor hochladen.

- **Adminbereich:**

wird von bestimmten Benutzern verwendet, die als Admin gekennzeichnet sind. Diese können nun Bilder, die von den Benutzern gemeldet worden sind, ansehen, gegebenenfalls löschen oder eine Änderung am Namen oder Sonstigem vornehmen.

Datenbankmodell:



Pictures:

- In dieser Tabelle befinden sich die Metadaten des Bildes (Title, Tag, CreationDate, CreationUserId)
- Die CreationUserId ist ein Fremdschlüssel und referenziert hier den Benutzer, der dieses Meme erstellt hat.
- Das Bild an sich welches angezeigt werden soll, ist in der base64-Codierung im Feld Image vorhanden. (im Bild orange markiert)

PortalUsers:

- Diese Tabelle ist für die Speicherung der registrierten User zuständig.
- Hier wird auch das Feld IsAdmin abgespeichert, welches besagt ob dieser Benutzer die Admin-Bereiche sehen darf oder nicht.

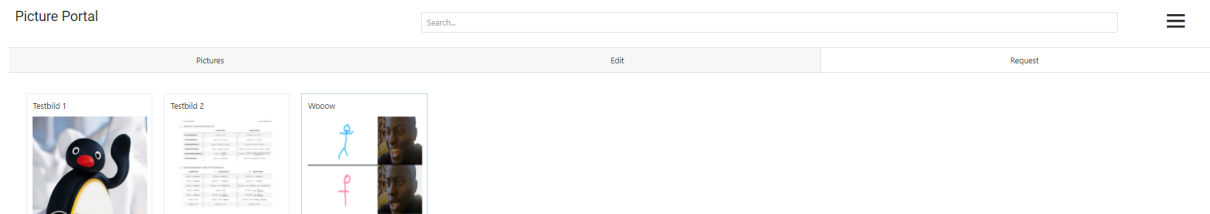
RequestItems:

- Hier werden die Meldungen von Usern gespeichert.
- Es wird der Titel und eine Beschreibung mit gespeichert.
- Und durch Fremdschlüssel werden hier das Bild und der die Meldung erstellte User referenziert.

Realisierte Funktionen (technische Details)

Im Folgenden werden die einzelnen Module bzw. Komponenten von unserem Angular-Projekt (Online-Portal) einzeln erklärt und deren Anbindung ans Backend wird jeweils erklärt.

Pictures Ansicht:



Als Erste Ansicht wenn man die Picture-Portal Seite öffnet, kommt man auf die Pictures-Ansicht.

Hier sieht man die von den anderen Benutzern erstellten Memes.

Diese kann man per Klick aufs Bild vergrößern, um die Metadaten einzusehen und auch gegebenenfalls diesen Beitrag zu reporten:



Um die Bilder aus der DB zu holen wird hier folgender REST-Aufruf ausgeführt "GET: api/Pictures":

```
// GET: api/Pictures
[HttpGet]
// Verweise
public async Task<ActionResult<IEnumerable<PictureEntry>>> GetPictures()
{
    // Include for joining foreign key relation-objects
    return await _context.Pictures
        .Include(p => p.Creator)
        .ToListAsync();
}
```

Dieser Aufruf liefert uns eine json-Datei, welche alle Daten von der Tabelle Pictures beinhaltet.

Die wichtigste Stelle in der Tabelle ist hierbei der base64-codierte String.

Dieser wird von uns dann clientseitig per JavaScript decodiert und als normales Bild angezeigt:

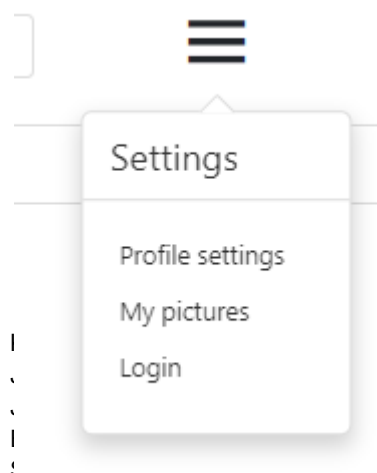
```
constructor(private store: Store,
    private router: Router,
    private sanitizer: DomSanitizer,
    private activatedRoute: ActivatedRoute) { }

convertBase64ToImage(base64Image: string) {
    return this.sanitizer.bypassSecurityTrustResourceUrl('data:image/jpg;base64,' + base64Image);
}

<div *dxTemplate="let tile of 'pictureViewTemplate'">
    <dx-item [widthRatio]="1" [heightRatio]="1" class="pointer">
        <div style="padding: 10px">
            {{ tile.title }}
            <img style="padding-top: 10px" [width]="200" [height]="180" [src]="convertBase64ToImage(tile.image)" />
        </div>
    </dx-item>
</div>
```

Settings PopUp:

Um auf die weitere Funktionalität unseres Portales zuzugreifen muss man sich anmelden, dies erfolgt in dem man rechts oben auf das Menü klickt:



In diesem Menü kann man:

- Profileinstellungen ändern
 - E-Mail, Username, Passwort ändern
- Seine selbst erstellten Bilder anzeigen
 - wie die Portal-Pictures Ansicht nur mit Filter, dass man nur seine eigenen Memes sieht
 - Und als letztes noch ein Login.

Login-Page

Log in

Username

Password

✓ Log in

OR

Sign up

Auf dieser Seite kann man sich nun entweder anmelden, falls ein Konto vorhanden oder sich registrieren:

Become a member

Username

Email

Password

✓ Sign up

In beiden Fällen wird ein POST-Anruf an das Backend gesendet, dieser

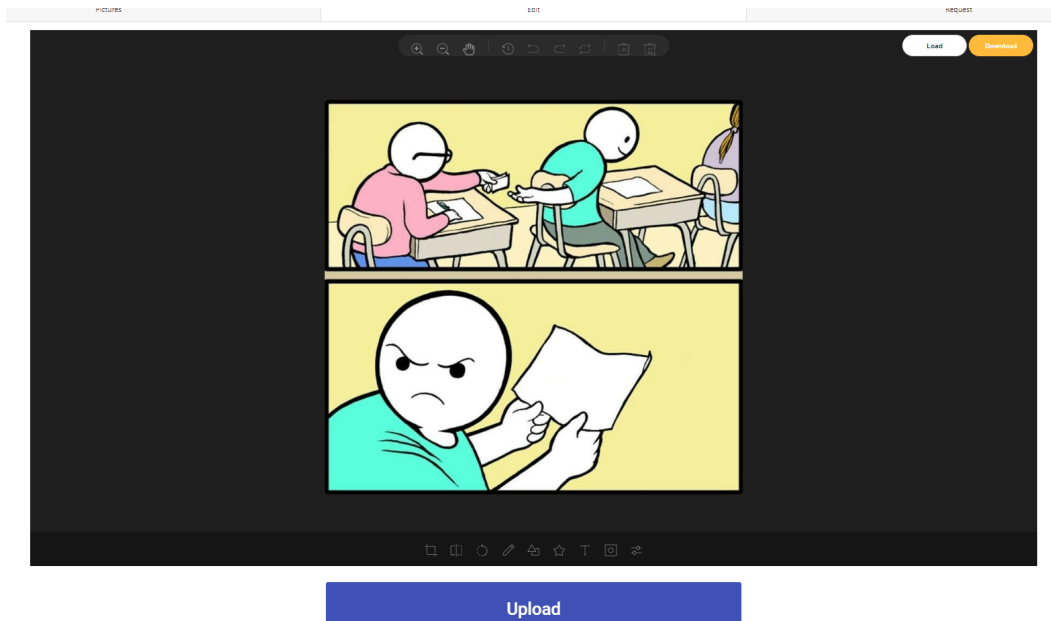
- schaut im Falle einer Anmeldung in der Tabelle nach ob die Anmeldung möglich ist
- speichert im Falle einer Registrierung den Benutzer in die Tabelle.

Als Antwort auf eine erfolgreiche Anmeldung/Registrierung einen JWT-Token. Dieser wird bei den weiteren Aufrufen mitgesendet, um zu zeigen das der Benutzer authentifiziert ist.

Im Backend wird sich in einer weiteren kleinen Tabelle gemerkt welcher User hinter diesem Token steckt.

Edit-Page:

Auf dieser Seite unseres Portales, können nun Memes erstellt werden:



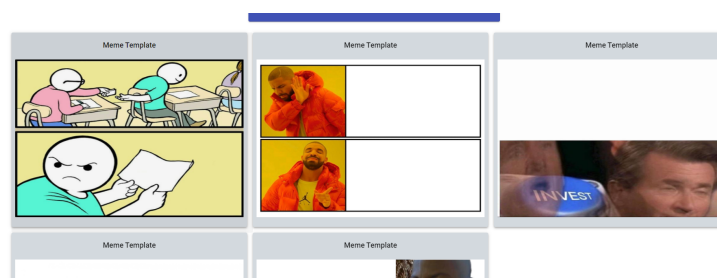
Zum Editieren der Bilder ist ein Image-Editor implementiert, der folgendes kann:

- Zusammenschneiden des Bildes
- Spiegeln des Bildes
- Drehen des Bildes
- Zeichnen auf Bild
- Formen/Sticker/Eigene Bilder hinzufügen auf Bild
- Text auf Bild hinzufügen
- Verschiedene Filter anwenden.

Dieser Editor benutzt reines Javascript um die Bildmanipulation zu gewährleisten. Hierbei wird das Bild in ein HTML Canvas geladen und in diesem werden dann die gewünschten Objekte hinzugefügt.

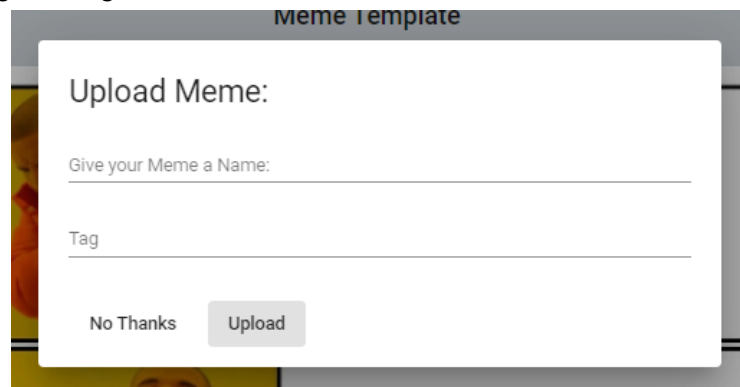
Um ein Bild in den Editor zu bekommen, kann

- rechts oben auf den Load-Button gedrückt werden.
- oder runtergescrollt und eines der vorhandenen Meme-Templates ausgewählt werden:



Wenn man nun zufrieden ist mit seinem neu erstellten Meme, kann man dieses hochladen:

- per Klick auf Upload erscheint ein Dialogfenster, wo man den Namen und den jeweiligen Tag mit angeben soll:



- Nachdem Ausfüllen und auf Upload klicken, wird ein POST-Aufruf gesendet, der den Namen, das Tag und das erstellte Bild als base64String beinhaltet.

Weiters wird noch der JWT-Token angehängt, damit das Backend weiß, von wem dieser Aufruf kommt.

```
// POST: api/Pictures
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see https://aka.ms/RazorPagesCRUD.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult<PictureEntry>> PostPictureEntry(PictureEntryDTO picture)
{
    PictureEntry pictureEntry = new ();
    pictureEntry.Title = picture.Title;
    pictureEntry.Tag = picture.Tag;
    pictureEntry.CreationDate = picture.CreationDate;

    //Get managed object from context, otherwise EF wants to insert a new customer
    var creator = await _context.PortalUsers.FindAsync(picture.CreatorId);
    pictureEntry.CreatorUserId = creator.UserId;

    pictureEntry.Image = Convert.FromBase64String(picture.Image);

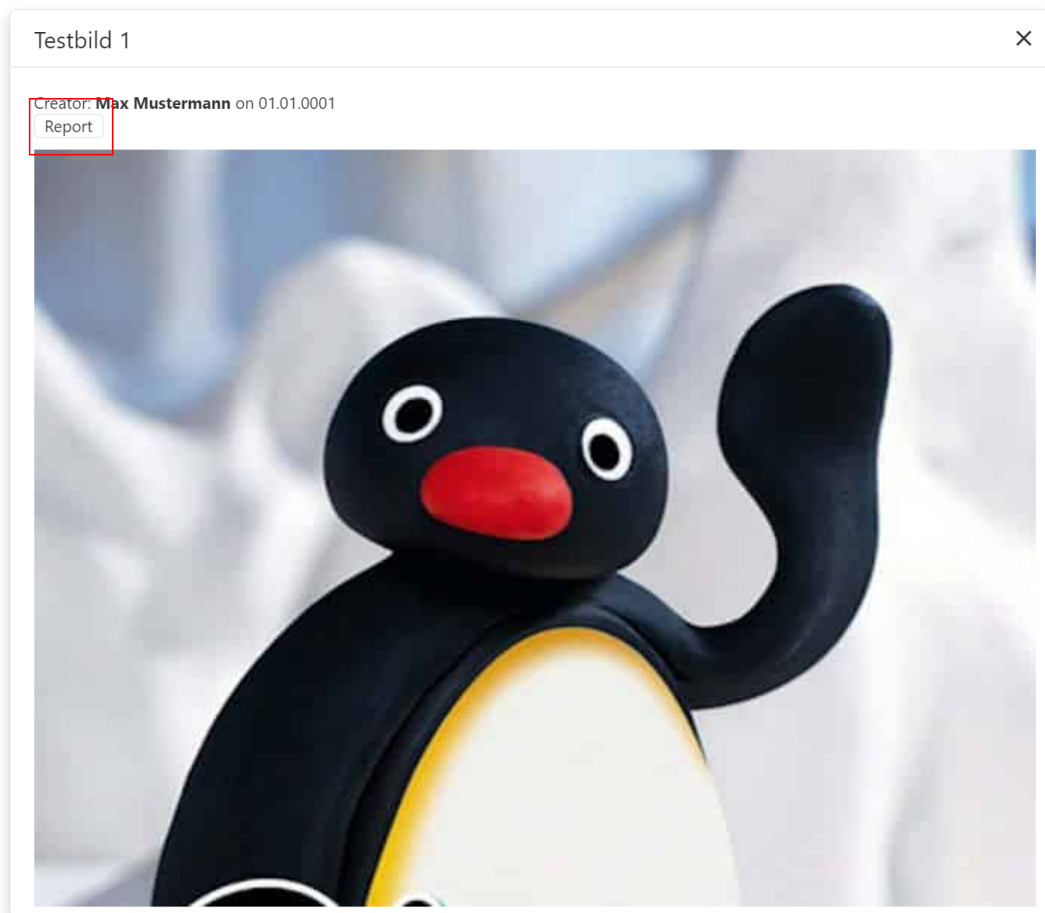
    _context.Pictures.Add(pictureEntry);

    await _context.SaveChangesAsync();

    return CreatedAtAction("PostPictureEntry", new { id = pictureEntry.PictureId }, pictureEntry);
}
```

Request-Page

Falls bei einem Bild ein Problem vorliegt, kann jeder eingeloggte Benutzer den Report Knopf benutzen, um die Administratoren unseres Portals zu kontaktieren.



Dieser Knopf leitet den Benutzer auf eine Eingabemaske weiter, welche es erlaubt eine beliebige Nachricht einzugeben. Die dazugehörige Bildnummer wird dabei automatisch Ausgefüllt.

Title: *

Description: *

Image: *

Admin-Page

Benutzer die als Admin angemeldet sind haben die Möglichkeit, von Benutzern erstellte Requests einzusehen und Bilder dementsprechend zu bearbeiten oder zu entfernen.

Wos is mid du?

Testticket

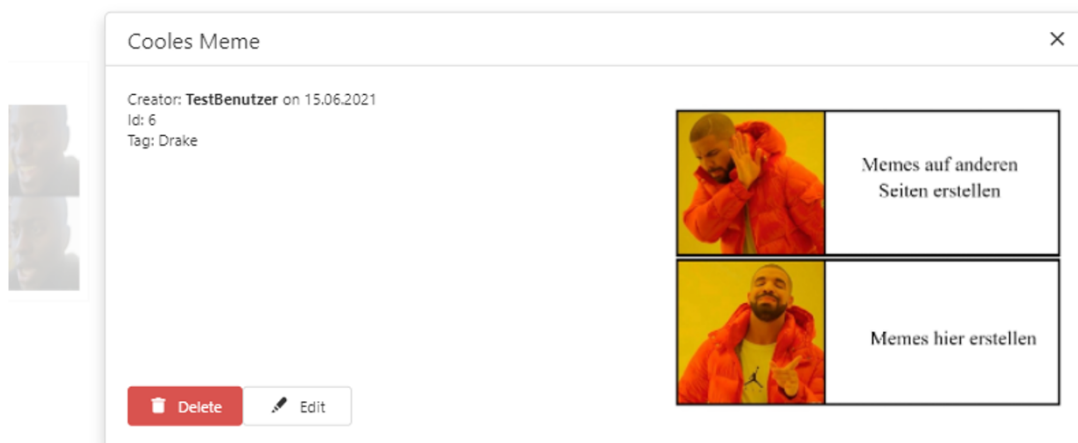
Sofort löschen!

Der hod mei Copyright verletzt mimimi!

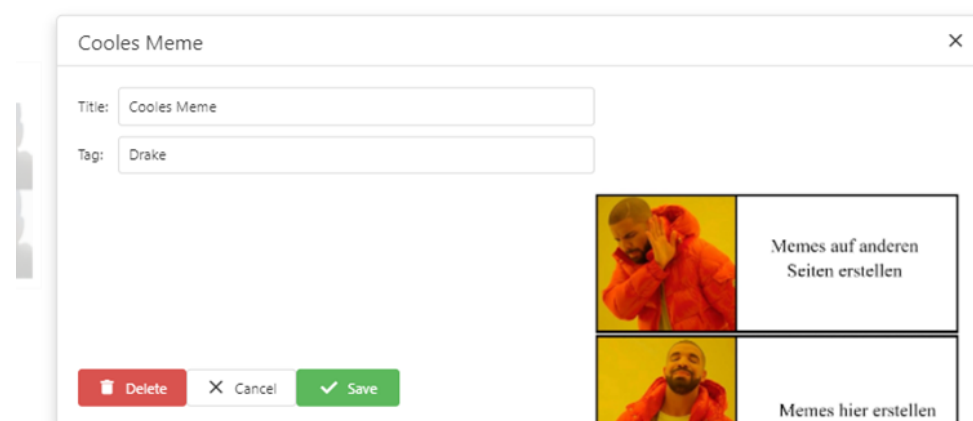
Datenschutz!

Ich bin in dem Bild drinnen und möchte das man mich verpixxelt!

Durch klicken auf einen Request wird eine Detailansicht des zugehörigen Bildes geöffnet, über welche die Daten des Bildes eingesehen und gegebenenfalls editiert werden können.

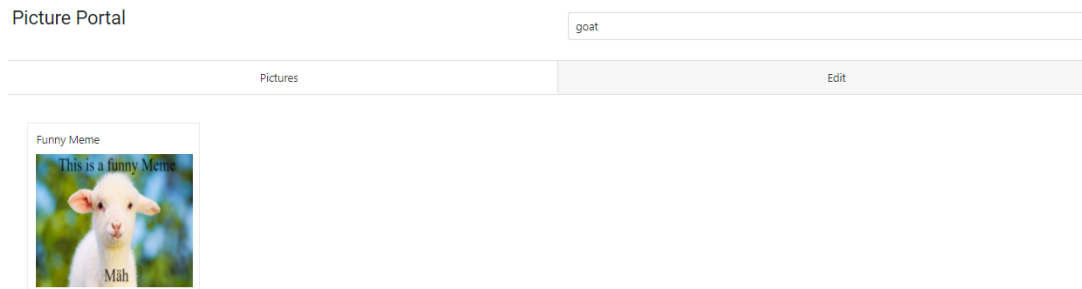


Ein Betätigen des Edit Buttons ermöglicht dem Benutzer Titel und Tag des Bildes über Textfelder anzupassen. Durch klicken auf Save werden die Änderungen übernommen, bei einem Klick auf Cancel wird zurück zur Detailansicht gewechselt.



Search:

Da die verschiedenen Bilder alle mit einem Tag ausgestattet sind kann man, durch die Suche die Bilder nach den Tags oder nach dem Ersteller filtern: Hier ein Beispiel mit dem Suchbegriff "goat".



Hierfür erfolgt im Backend eine Filterung, welche uns dann die Memes anzeigt, welche den Suchbegriff in ihrem Tag oder als ihren Ersteller haben:

```
// GET: api/Pictures/ByTag/:tag
[HttpGet("ByTag/{tag}")]
// Verweise
public async Task<ActionResult<IEnumerable<PictureEntry>>> GetPicturesByLabel(string tag)
{
    return await _context.Pictures
        .Where(p => p.Tag.Contains(tag))
        .Include(p => p.Creator)
        .ToListAsync();
}
```

JWT

JSON Web Tokens sind eine State of the Art Technologie, welche für das Benutzerberechtigungen eingesetzt wird. Im Gegensatz zu traditionellen Methoden wie Session Ids benötigen JWTs keinen Datenbankzugriff um die zu prüfen, ob der Benutzer die von ihm angegebenen Rechte besitzt. Dies wird dadurch erreicht dass die Rechte des Nutzers im Klartext übertragen werden und durch eine Signatur das verändern dieser Informationen verhindert wird. Dies hat auch zur Folge, dass der User seine Session selbst verwaltet, für einen Logout wird keine Internetverbindung benötigt, es wird lediglich der JWT aus dem Speicher geleert.

JWT werden von verschiedensten Programmiersprachen und Frameworks unterstützt was die Entwicklung von Applikationen stark vereinfacht. Auch Tools wie <https://jwt.io/>, welche den Inhalt eines JWT visualisiert, helfen bei der Entwicklung.

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDAxLzA1L2lkZW50aXR5L2NsYWltcy9uYW1laWR1bnRpZmllciI6IjMiLCJleHAiOjE2MjM3ODY0OTIsIm1zcyI6Imh0dHBzOi8vbG9jYXRob3N00jQ0Mzg0LyIsImF1ZCI6Imh0dHBzOi8vbG9jYXRob3N00jQ0Mzg0LyJ9.bGmcB16k9_uYwhwTsKPUheZM6D_VN9vDi55optFMQlg
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier": "3",
  "exp": 1623786492,
  "iss": "https://localhost:44384/",
  "aud": "https://localhost:44384/"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  m`Hp7rh17Rt66LRaxBhkHP
)
```

☐ secret
 ☒ base64
 ☐ encoded

Signature Verified

SHARE JWT

Die Visualisierung eines von unserem Portal generierten JWT.

Kaan Baylan
 Julian Garn
 Jakob Rumpelsberger
 Elias Schäch
 Simon Primetzhofner

14

Quellen für ähnliche Projekte

Unser Github-Repo:

<https://github.com/SimonPrimetzhofer/mms>

Image-Editor Javascript-Library

<https://ui.toast.com/tui-image-editor>

Devextreme Angular Komponenten (kostenpflichtig für kommerziellen Gebrauch)

<https://js.devexpress.com/>

Angular (Frontendframework)

<https://angular.io/>