Pens Website

- *Project idea / users / general information*
- *Data required for your users*
- *Database design (complete)*
- *Website overall layout/design*
- *Website individual page designs and layout incl. fonts / colours / etc*
- *Routes (@app.route) required for each page, and related function signatures*
- *SQL queries for each required route to provide the information from the database that you think you'll need*

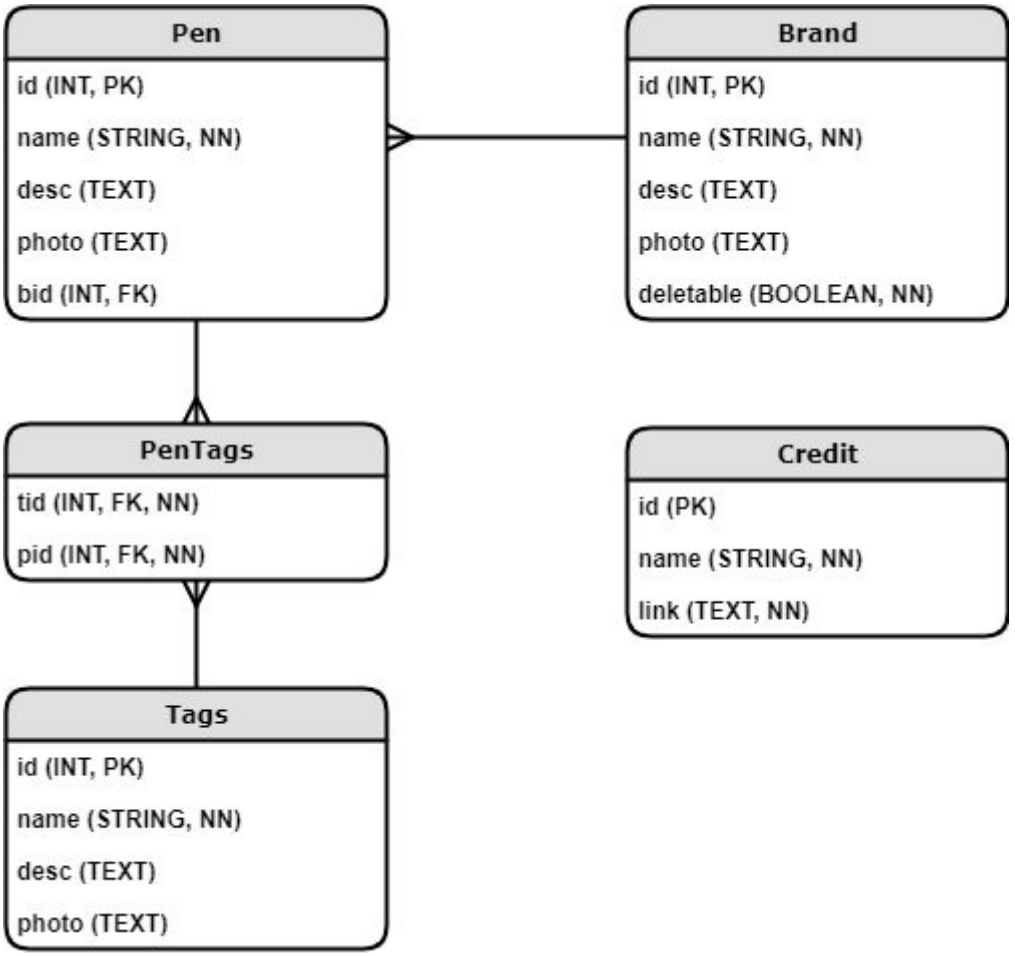
Idea

I am going to make a flask website about luxury pens. My website will present different types of pens, features, and brands of pen. The purpose of each pen and what they are best used for. The goal of this website is to inform the viewer about different pens and help them make a decision when wanting to buy a new pen. My target audience is luxury pen users and pen enthusiasts.

Some factors that would go into making the decision of which pen to buy would be: pen shape, comfort, material, ink, type, style, thickness, brand, cost, durability, etc.

Possible target audience: office, home, school, artistic, luxury, historic

Database



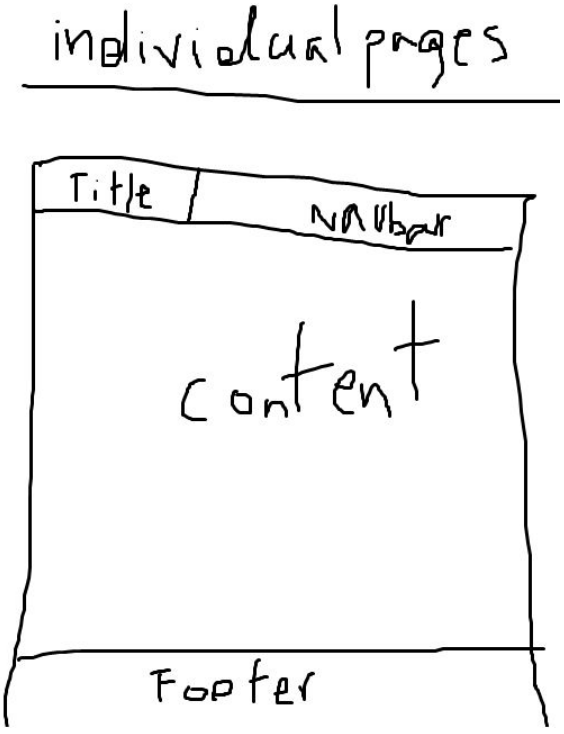
Tags

- Ballpoint
- Permanent
- Gel
- 0.5mm thickness
- 0.7mm thickness
- 1.0mm thickness
- Water-based
- Oil-based

Layout/Design

Initially, I was going to make a classic website with a general homepage which would then navigate to other parts of the website (eg. tags, brand) using links. But I realised, this would make the information on my homepage too similar to the information on the individual pages. So instead, I am going to make an interactive one-page website that contains information on all sections of the website, where then clicking on any item on page the takes the user directly to the specific item (eg. a brand, a pen, or a feature/tag).

The title and navbar will be inline next to each other. The page will have a background image on the landing page. The main content is sectioned off into brands, types and thicknesses. And then followed by the footer. The main content will have images for items which are doubled for flip over info cards. Clicking on the links in the navbar will jump to a location on the website. The types section will have a slideshow of different pen types. And the thickness will just be clickable images of different pen thicknesses.



Routes

- Home
- brand/id

- ➢ tag/id
- ➢ pen/id
- ➢ add-brand
- ➢ edit-brand/id
- ➢ delete-brand/id

SQL queries

Navbar:

Clicking on the links in the navbar will direct the user to the location on the page. If the user is not on the homepage, this link will direct them to the homepage and then the specific location.

All Brands:

These are presented in square info cards that have the image on the front and click to flip them over to view the information. The user can then click read more to view the brand's individual page with complete information. This page contains all the pens in the database that are created by this brand.

All Tags:

Tags of pens are presented in a slideshow format where there are next and previous arrow keys and the user can click on the content of the slideshow itself to direct them to the individual type of pen page. This page includes all the pens with this type of pen tag.

Credits:

This page includes credits and disclaimers.

Search Pens / All Pens:

This page includes a list of all pens displayed in a square grid similar to the all brands page. The page also includes a search function that can filter the pen grid by brand and tag. Pressing search will refine the pens grid to match the new search.

Pen:

This page includes more specific information for a pen model. The page will have the pen's brand, it's tags, description and photo, etc.

Brand:

This page includes more specific information for a brand. The page will have the brand name, it's pens, description and photo, etc.

Tag:

This page includes more specific information for a pen tag. The page will have the tag's name, it's pens, cost, description and photo, etc.

Tools

VS Code

I will be using Visual Studio Code as my main code editor/IDE. I made this choice because I am the most familiar with this application and know how to use it's features effectively. I have used VS Community in my game design class for three years now and am experienced with the software. VS Code is similar to that because they are created by the same company, therefore, I can quickly familiarize myself with the application and use my previous knowledge here.

VS Code has some useful features that will help me work efficiently and achieve my goals. There is a source control section which allows me to review changes, commit, edit, revise, and more directly from within the application. This makes the job of version control much easier. All edits made are auto saved and changes instantly shown on the website. This speeds up the testing process as I can make small changes and quickly see the outcome.

SQLiteStudio

SQLiteStudio is my chosen database management system. This is mainly because it is free and easy to use and serverless. Other database management systems require some sort of database server, however SQLite is serverless which means the information is stored in a file that can then be imported into the project. This makes SQLite simple to set up and with zero configuration is required. It also makes it very portable which comes in handy for taking backups of the project. Some cons of this app are that as it is serverless, it doesn't provide network access through another device, and it is not the best for large-scale applications. This is

suitable as we will not be needing either of those as we are creating a small database on a local machine.

Google Chrome Inspector

I regularly used Chrome Inspector for debugging purposes. The inspector provides a visual representation of the website layout in code and helps to easier understand the code. This helps often when I am confused about what a certain feature does or how it interacts with my website. The inspector allows for live changes to the website html and css which becomes useful for testing and previewing changes without having to go back to the code editor.

Other

Some helpful websites that I frequently used were CSS Tricks, Stack Overflow and W3 Schools.

Development

Legal, ethical and moral requirements

I added a credits table to the database. The purpose of this table is to hold information about all the content used that is copyrighted or requires crediting of the original author. This includes data such as images used and brand/pen descriptions.

At the current state, the website is at risk of breaching copyright laws. This is because I have only credited the original creators of the content without actually personally consulting with them whether it's okay to use the material. This means that if the website was to go public, these companies have the ability to sue the website. In order for using their content to be legal, I need to individually ask permission for each piece of content I use. But since this is just for a school project and the website will never actually go public, the images and text I have used are just a placeholder and may not be the final outcome. If I wanted to publish this website, I would need to get permission for using all of the text and images shown.

Moral: There is a disclaimer on the credits page about validity of the information provided. As the website is open to public access, anybody can edit the information in the database through the website. The disclaimer states that information is intended as accurate and true. However we do

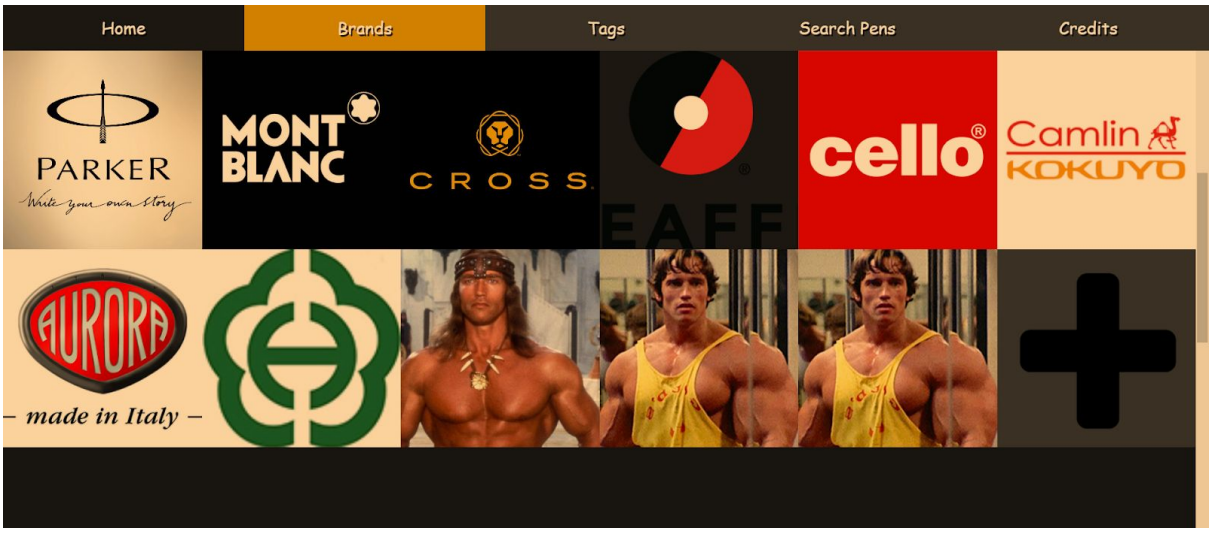
not hold responsibility for false information as anybody could have edited the website; and users should check their own facts. This is because we hold a moral responsibility to provide correct information and the users should be informed in the case that information may not be accurate.

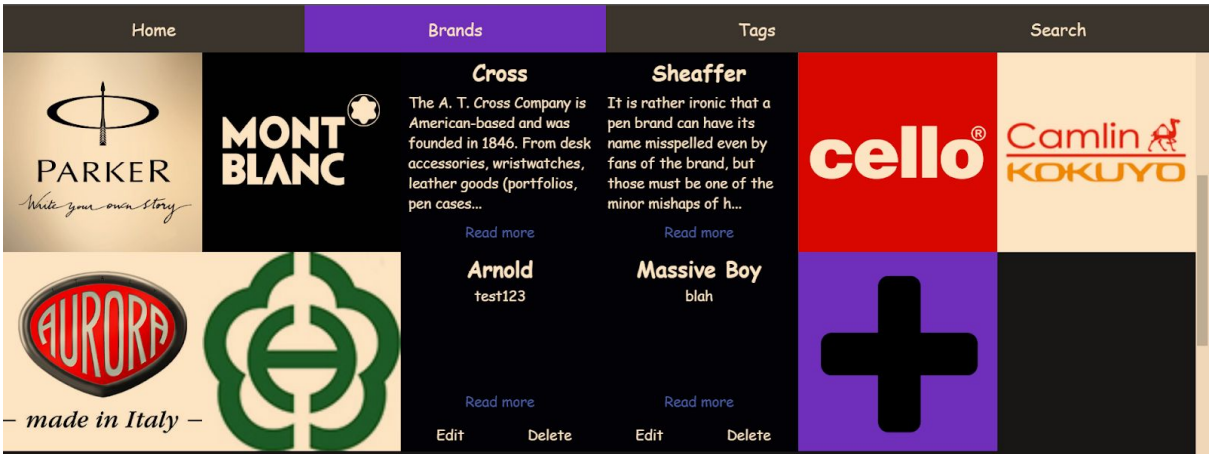
PenTag table modification

I changed my joining table from a complex one to a simple. This is because my joining table does not need to hold any information about the link being made. For example, with a website about movies and actors, the joining table of MovieActor would need to hold information about the role of the actor in the movies, like their character name, so a complex joining table is needed. With my website, the joining table is only there to link pen to tag and no other information is needed to be held in the table. This is why I chose to switch from a complex to a simple joining table. I tested this works by...

Colour Choice

For my color scheme, I decided to go with a mix of gold and black. Gold because my website is about luxury pens and gold is a suiting color as it represents Gold, richness and prosperity. Another choice would have been purple as it represents mystery and royalty but it looked a little too vibrant on the website. Black is a nice complementary because it is easy on the eye to look at on device screens and goes well with almost any other color. Most modern websites nowadays have a dark theme to ease the eye-strain for late night use and for eliminating blue-light. So I chose this color scheme from www.schemecolor.com which combines black, gray and gold colour to create a nice palette. I asked a few classmates for feedback on this color choice. They all agreed that golden suits better over purple.





https://colorpalettes.net/color-palette-2088/

CSS Grid

Css Grid is a system for making two dimensional website layouts. Every other option in css until now has been a one dimensional work-around such as float, display flex, etc. CSS Grid is different because it contains both rows and columns which gives for so much more freedom in designing layouts. It makes creating layout and designing websites much easier with its large amount of customization. I used this in my website in the brands grid, the slideshow, the pen pages and etc.

Access permissions

Who can edit the database? Who can access the database file location?

My website works similarly to Wikipedia where anybody can access and edit the information posted. This includes brands, tags, pens, etc. This is

only done through the client side interface which means the user cannot directly access the database which would be a security breach. There is a disclaimer on the credits page which states that any information on the website may be inaccurate and we do not hold responsibility.

The database files and images are saved to the server computer. Users cannot access the database. Users can only access the information that is displayed on the website and the javascript executed on the client side.

If I was to improve the website in the future, I would develop a login system. This would make the website more secure as only authorised users can edit the data and everyone can see who edited what information.

Testing

Navbar

The navigation bar contains all the important links to assist the user in navigating around the website.

The buttons are arranged in a grid. The grid uses a variable "--navbar-items" to determine the number of columns needed. To find the number of buttons, I have used a javascript function. This function finds the nav div and counts the number of children, then sets the value. This is future proofing the website as more navbar links may be added. The code is able to adapt to the new number of buttons.

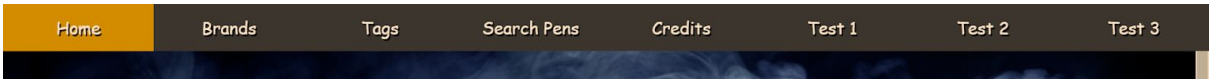
```
/* gets value of navbar items from a javascript function */
--navbar-items: 4;
/* navbar height variable | so I can modify the value in only one place */
--navbar-height: 60px;
```

```
/* navigation bar */
nav{
    overflow: hidden;
    background-color: var(--gray-color);
    position: fixed;
    width: 100%;
    height: var(--navbar-height);
    z-index: 1000;
    transition: all 0.4s ease;
    display: grid;
    grid-template-columns: repeat(var(--navbar-items), 1fr);
    place-items: center;
}
```

```
// Calculates the number of navbar links in the header
navbar = () =>{
    const navItems = $('nav a').length;
    const body = $('body');
    body.css({'--navbar-items': navItems});
}
```

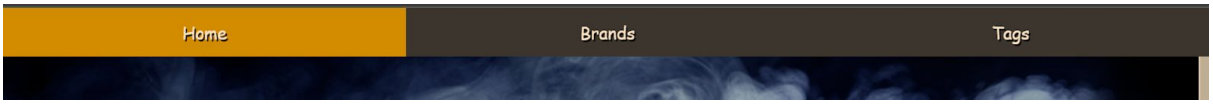
Example: Increased number of links from 5 to 8.

```
<nav>
    <a href="/#home" class="navlink" id
    <a href="/#brands" class="navlink">
    <a href="/#tags" class="navlink"><p
    <a href="/search" id="searchlink"><
    <a href="/credits" id="creditslink"
    <a href="/test1"><p>Test 1</p></a>
    <a href="/test2"><p>Test 2</p></a>
    <a href="/test3"><p>Test 3</p></a>
</nav>
```



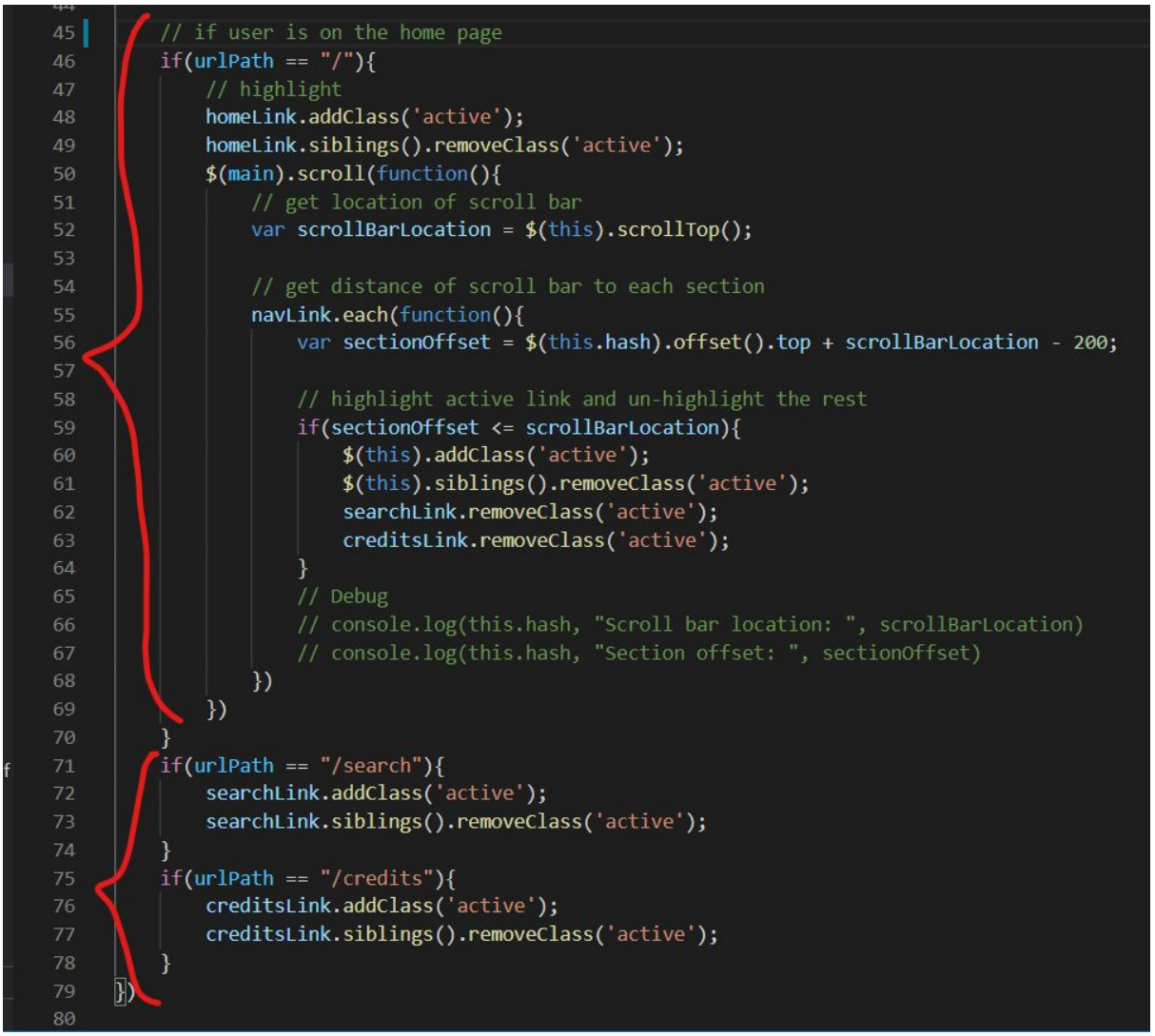
Example: Decreased number of links from 5 to 3.

```
<nav>
    <a href="/#home" class="navlink" id="homelink"><p
    <a href="/#brands" class="navlink"><p>Brands</p><
    <a href="/#tags" class="navlink"><p>Tags</p></a><
</nav>
```



Navlink highlighting

```
// Highlight anchor link
$(document).ready(function(){

    var navLink = $('.navlink');
    const main = $('#main');
    var homeLink = $('#homelink');
    var searchLink = $('#searchlink');
    var creditsLink = $('#creditslink')


    const urlPath = window.location.pathname;
```



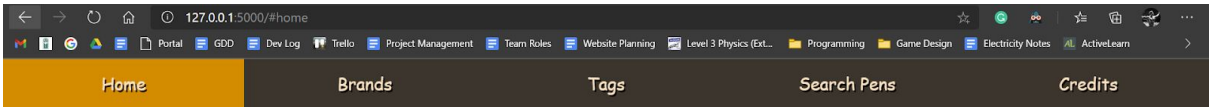
```
// if user is on the home page
if(urlPath == "/"){
    // highlight
    homeLink.addClass('active');
    homeLink.siblings().removeClass('active');
    $(main).scroll(function(){
        // get location of scroll bar
        var scrollBarLocation = $(this).scrollTop();

        // get distance of scroll bar to each section
        navLink.each(function(){
            var sectionOffset = $(this.hash).offset().top + scrollBarLocation - 200;

            // highlight active link and un-highlight the rest
            if(sectionOffset <= scrollBarLocation){
                $(this).addClass('active');
                $(this).siblings().removeClass('active');
                searchLink.removeClass('active');
                creditsLink.removeClass('active');
            }
            // Debug
            // console.log(this.hash, "Scroll bar location: ", scrollBarLocation)
            // console.log(this.hash, "Section offset: ", sectionOffset)
        })
    })
}
if(urlPath == "/search"){
    searchLink.addClass('active');
    searchLink.siblings().removeClass('active');
}
if(urlPath == "/credits"){
    creditsLink.addClass('active');
    creditsLink.siblings().removeClass('active');
}
})
```

The navlink highlighting is achieved using javascript. The first part of the code calculates the offset from the top of the home page to each section to determine which section of the page the user is on. This is used to highlight the appropriate anchorlink and un-highlight the rest. This method for highlighting links ensures that the highlights are always up to date both when scrolling and when directly clicking the links.

The next section is for the exceptions which are the non anchor links.

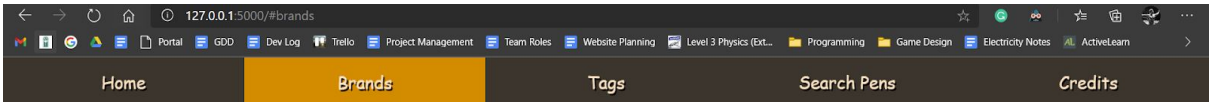
The first three links (Home, Brands, Tags) in the navbar are anchor links to different sections on the home route.

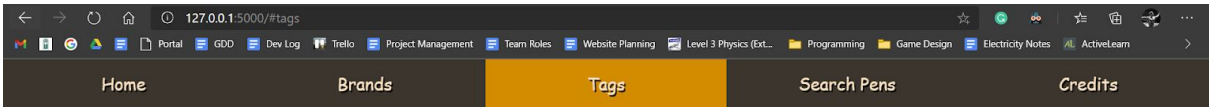
```
style.css | forms.py | main.py | search.html | script.js | nav.html | models.py
templates > nav.html > nav > a#searchlink > p
1  <nav>
2      <a href="/#home" class="navlink" id="homelink"><p>Home</p></a><!-- scroll to the top -->
3      <a href="/#brands" class="navlink"><p>Brands</p></a><!-- directs user to the all brands page -->
4      <a href="/#tags" class="navlink"><p>Tags</p></a><!-- directs user to the all tags page -->
5      <a href="/search" id="searchlink"><p>Search Pens</p></a><!-- directs user to the search page -->
6      <a href="/credits" id="creditslink"><p>Credits</p></a><!-- directs user to the credits page -->
7  </nav>
```



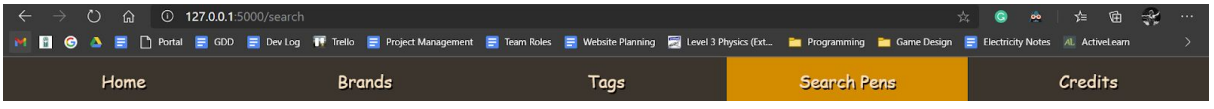
Clicking on the anchor links takes the user to the div with the matching id. For example, the brands button has link “/#brands”. This takes the user to the div on the page with an id of “brands”.

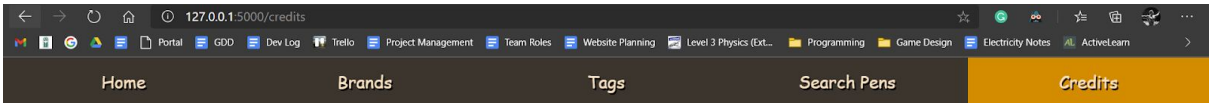
```
style.css | forms.py | main.py | search.html | all_brands.html
templates > all_brands.html > div#brands.section > div#brands-grid
1  <div class="section" id="brands">
2      <div id="brands-grid">
3          <!-- current brands -->
4          {% for brand in brands %}
5          <div class="card">
```





The next two (Search Pens, Credits) are normal links to other page routes on the website.





I used CSS to achieve the scrolling animation played when navigating

```
/* main div inside body that contains
#main{
    scroll-behavior: smooth;
    height: 100%;
    width: 100%;
    overflow-y: scroll;
    scroll-snap-type: y mandatory;
    scroll-padding-top: var(--navbar-
}
```

between the anchor links on the home page. This is done by setting a scroll snap type to the main div which contains all of the page content. Then adding scroll snap align to the section class. The section class is on every main section e.g landing page, all brands, all tags, etc. This achieves a smooth transition between anchor links. If the page is left half way through a section, the screen will snap to the nearest section.

Home Page Layout

Created an individual html file for landing page, all brands, and all tags.

Home page just includes these html files in the block content.

```
home.html ×
mplates > <> home.html
1   {% extends 'layout.html' %}
2   {% block content %}
3   {% include 'landing_page.html' %}
4   {% include 'all_brands.html' %}
5   {% include 'all_tags.html' %}
6   {% endblock %}
```

Landing Page

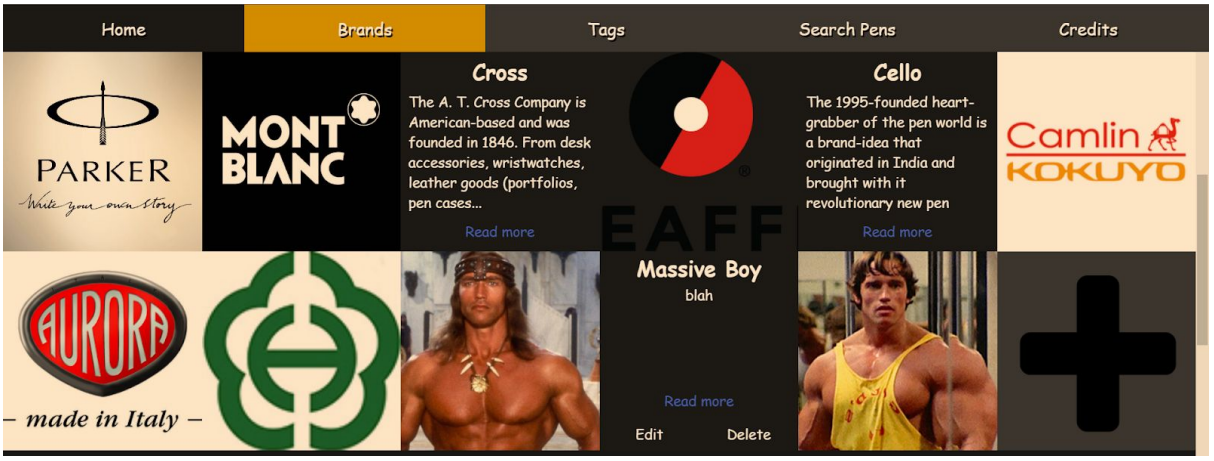
This page is a section of the home page. The landing page is the first thing the user sees upon entering the website. This is designed to be aesthetically pleasing and attract the viewer to stay. There are no queries required for this page.

```
1   <div class="section" id="home" style="grid-template-rows: 1fr;">
2       <div id="wallpaper">
3           <h1>Welcome to Pens</h1>
4       </div>
5   </div>
```

The image used is an image found on google images under creative commons license.

All Brands / Grid

This page is a section of the home page.



In the html, I used the jinja functions to access the brands table in the database and populate the brands page. The cards created by the admin have the not deletable boolean set as true and cards created by users are deletable. The jinja statement on line 16 covers for this to display the edit and delete buttons only on brands that are deletable.

main.py | all_brands.html | script.js | style.css

templates > all_brands.html > div#brands.section > div#brands-grid

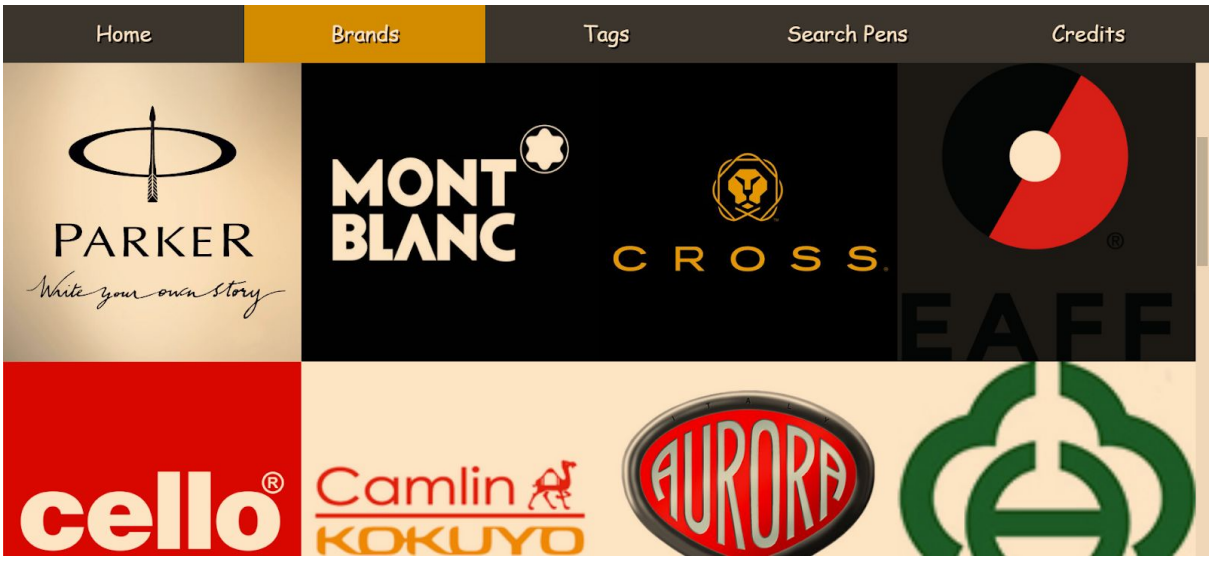
```
<div class="section" id="brands">
    <div id="brands-grid">
        <!-- current brands -->
        {% for brand in brands %}
        <div class="card">
            <div class="card-content" onclick="flip(this)">
                <!-- brand logo as a background image -->
                <div class="card-photo" style="background-image: url('/static/images/brands/{{ brand.photo }}')"></div>
                <!-- brand information -->
                <!-- jinja2 if statement to correct for card grid template -->
                <!-- This adjustment is because non-deletable cards have one less row in the grid -->
                <div class="card-info" {% if brand.deletable == true %}style="grid-template-rows: 1fr 3fr 1fr 1fr;"{% endif %}>
                    <h2 class="title">{{ brand.name }}</h2>
                    <p class="description">{{ brand.desc }}</p>
                    <a class="readmore" href="{{ url_for('brand', id = brand.id) }}">Read more</a>
                    {% if brand.deletable == true %}
                    <div class="edit-delete">
                        <a class="edit" href="{{ url_for('edit_brand', id = brand.id) }}">Edit</a>
                        <a class="delete" href="{{ url_for('delete_brand', id = brand.id) }}">Delete</a>
                    </div>
                    {% endif %}
                </div>
            </div>
        </div>
        {% endfor %}
        <!-- add new brand -->
        <div id="add-card">
            <a href="{{ url_for('add_brand') }}"></a>
        </div>
    </div>
    <!-- give the section some breathing room so that the screen doesnt jump to the next section on the first scroll -->
    <div style="margin-top: 400px"></div>
</div>
```

CSS:

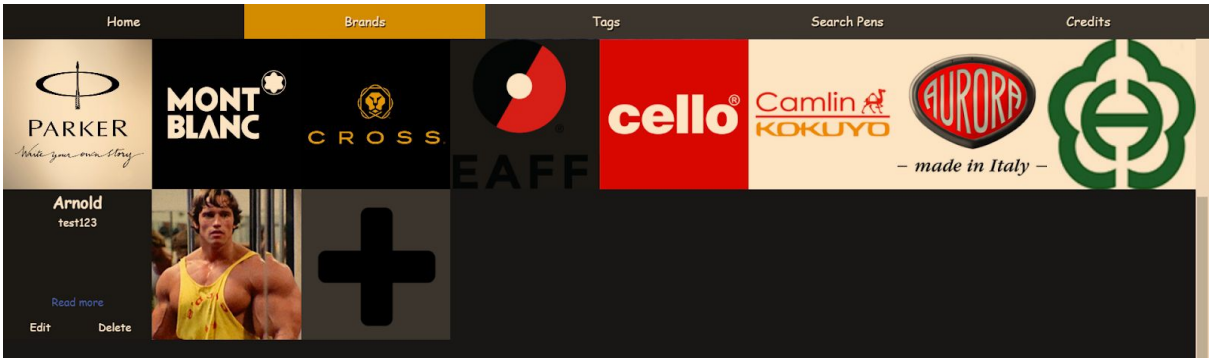
Here we can see the css for the main brands grid which describes a grid column template to populate the brand cards. The use of grid allows for the table to adapt to different situations such as resizing of the window.

```
/* grid of all brand info cards */
#brands-grid, #pens-grid{
    width: 100%;
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
}
```

Here the window has been zoomed in and you can see that the grid is able to adjust by lowering the number of columns from 6 to 4 to fit the page.



Zooming out will result in the opposite where the number of columns is increased as there is room for more cards in each row.



The CSS preserve-3d function is used to create the 3D card flip animation when the card is clicked. The front side of the card shows the brand logo image. When the card is flipped over, it reveals the brand name, descriptions and etc. The event listener for the click is in the javascript.

```css
/* card */
.card-content{
    height: 100%;
    width: 100%;
    position: absolute;
    transform-style: preserve-3d;
    transition: all 0.8s ease;
}

/* card photo */
.card-photo{
    width: 100%;
    height: 100%;
    position: absolute;
    backface-visibility: hidden;
    background-color: var(--black-color);
    background-repeat: no-repeat;
    background-position: center;
    background-size: cover;
    cursor: pointer;
}
```

This is the javascript which works with the CSS to provide the squared brand cards and the 3D card flipping animations.

```javascript
// Resize all brand cards into squares by setting the height to equal the width.
squarify = () =>{
    var cardList = document.getElementsByClassName('card');
    if(cardList != null){
        for(var i = 0; i < cardList.length; i++){
            var cardWidth = $(cardList[i]).width();
            $(cardList[i]).css({'height': cardWidth+'px'});
        }
    }
}

// Flip brand card on click
flip = (cardContent) =>{
    // Flip
    if(cardContent.style.transform == ''){
        cardContent.style.transform = 'rotateY(180deg)';
    }
    else{
        // Unflip
        if(cardContent.style.transform == 'rotateY(180deg)'){
            cardContent.style.transform = '';
        }
    }
}
```

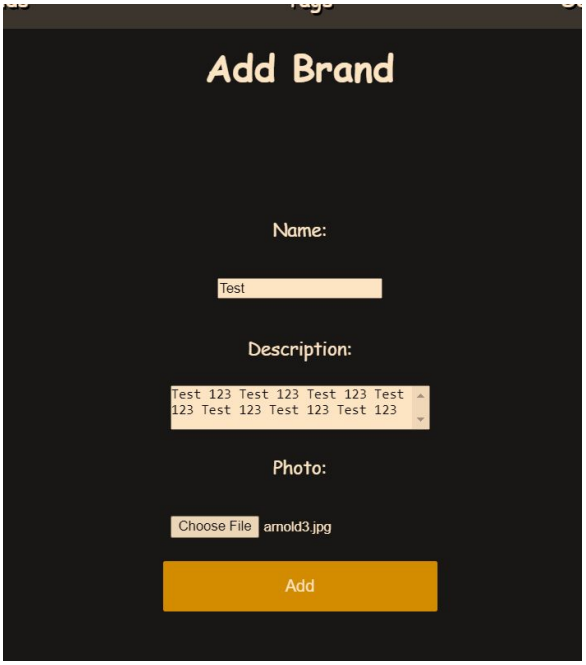
Add Brand

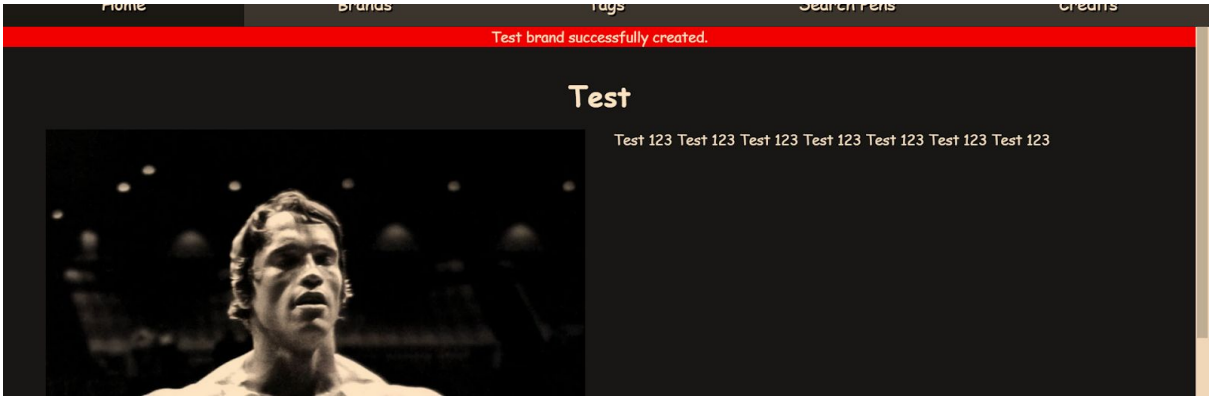
The add brand route creates a form. This form contains all the necessary data fields to create a new brand (name, description, photo). Upon submission, the selected photo file is saved to the server using a random hex name and a new brand is added to the database. The hex is used to ensure that no two files have the same name which would cause duplicate problems. The user is then redirected to the new brand page with a flash message that brand creation has been successful.

```
# save brand photo to computer files
def save_photo(form_photo):
    # give the file a random name to prevent errors with similar file names already in the database
    random_hex = secrets.token_hex(8)
    f_name, f_ext = os.path.splitext(form_photo.filename)
    photo_fn = random_hex + f_ext
    # save the uploaded photo to the database
    photo_path = os.path.join(app.root_path, 'static', 'images', 'brands', photo_fn)
    form_photo.save(photo_path)
    return photo_fn
```

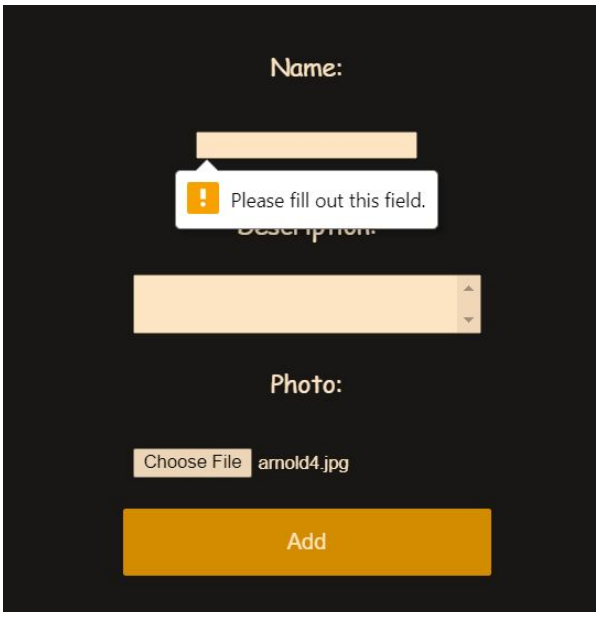
```
# form to add a brand to the database
@app.route('/add_brand', methods = ['GET', 'POST'])
def add_brand():
    form = Add_Brand()
    # form submitted to the server by a user
    if form.validate_on_submit():
        new_brand = models.Brand()
        # set brand name and description from form data
        new_brand.name = form.name.data
        new_brand.desc = form.desc.data
        if form.photo.data:
            photo_file = save_photo(form.photo.data)
            new_brand.photo = photo_file
        new_brand.deletable = True
        # flash message to let user know that the brand has been created
        flash('{} brand successfully created.'.format(new_brand.name))
        db.session.add(new_brand)
        db.session.commit()
        return redirect(url_for('brand', id=new_brand.id))
    # request to see the page
    return render_template('add_edit_brand.html', form = form, title = "Add Brand", legend = "Add")
```

Normal brand creation:





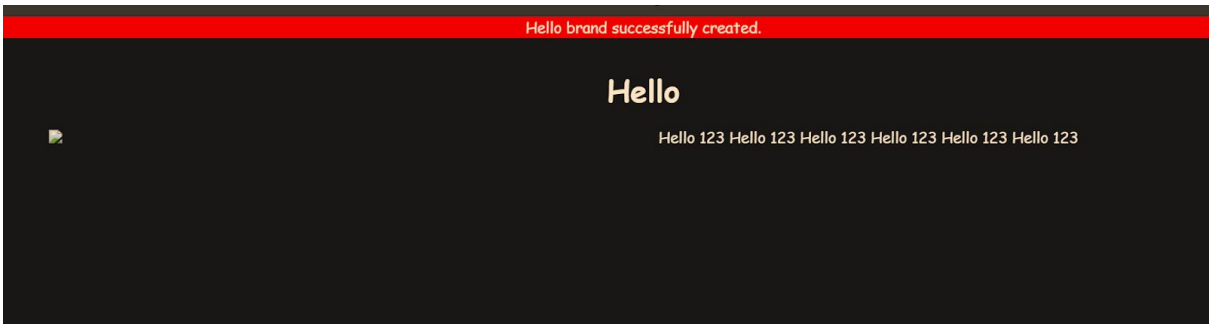
Brand creation with name and description left blank:




```
class Add_Brand(FlaskForm):
    name = TextField('name', validators=[DataRequired()])
    desc = TextAreaField('desc', validators=[Optional()])
    photo = FileField('photo', validators=[FileAllowed(['jpg', 'png']), Optional()])
```

The user receives a message that the name field needs to be filled. This is because in the forms.py file, we stated the name field as a required field whereas description and photo are optional.

Brand creation with no photo:



The user is allowed to create a brand with no photo.

This is acceptable because the photo can be added/changed later through the edit brand.

Edit Brand

The edit brand is similar to the add brand. The only differences are:

- The form is prefilled with the current brand data.
- Submitting will overwrite the current brand data with the new brand data.
- The old photo file is deleted and replaced by the new photo file.

If a field is left blank, that data remains unchanged.

```python
# form to edit a brand in the database
@app.route('/edit_brand/<int:id>', methods = ['GET', 'POST'])
def edit_brand(id):
    form = Add_Brand()
    # current brand data
    brand = db.session.query(models.Brand).filter(models.Brand.id==id).first_or_404()
    # form submitted to the server by a user
    if form.validate_on_submit():
        # set brand name and description from form data
        brand.name = form.name.data
        brand.desc = form.desc.data
        # check if user wants to update the brand photo
        if form.photo.data:
            # save new photo file
            photo_file = save_photo(form.photo.data)
            # delete old photo file
            delete_photo(brand.photo)
            # set new photo
            brand.photo = photo_file
        # flash message to let user know that the brand has been edited
        flash('{} brand successfully edited.'.format(brand.name))
        db.session.commit()
        return redirect(url_for('brand', id=brand.id))
    # populate fields with currently saved data
    form.name.data = brand.name
    form.desc.data = brand.desc
    # title for the edit page
    title = "Editing {} brand".format(brand.name)
    # request to see the page
    return render_template('add_edit_brand.html', form = form, title = title, legend = "Save")

```

Description of brand changed. Name and photo unchanged.

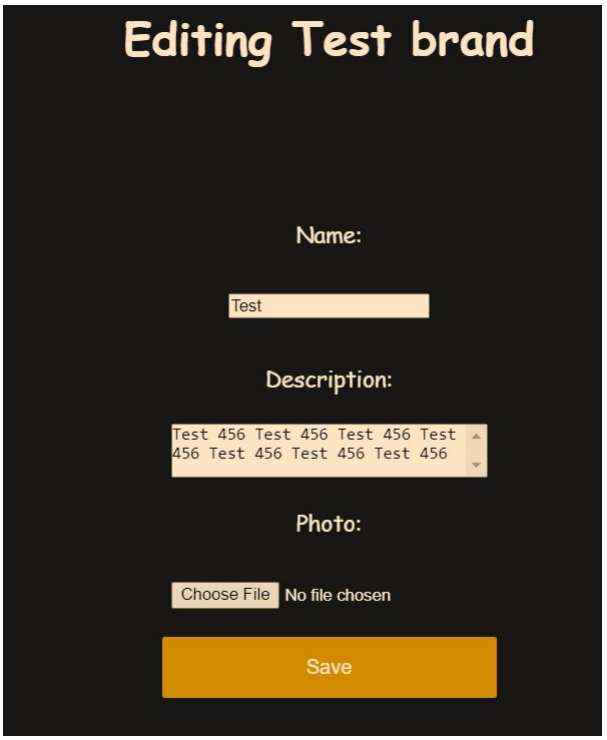
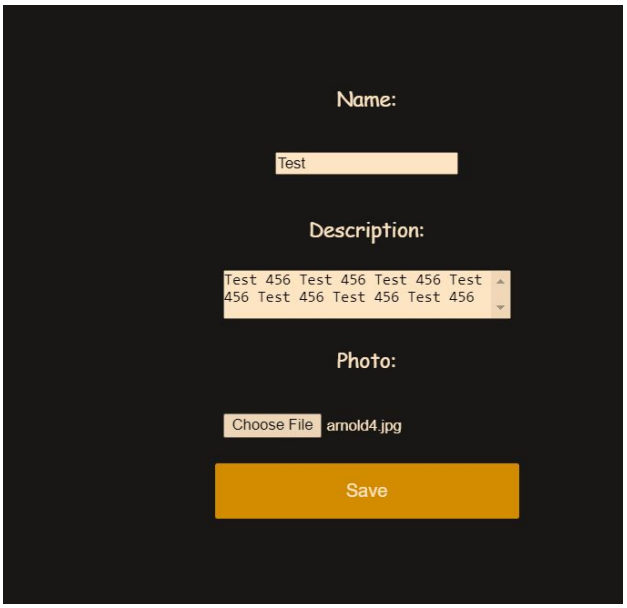
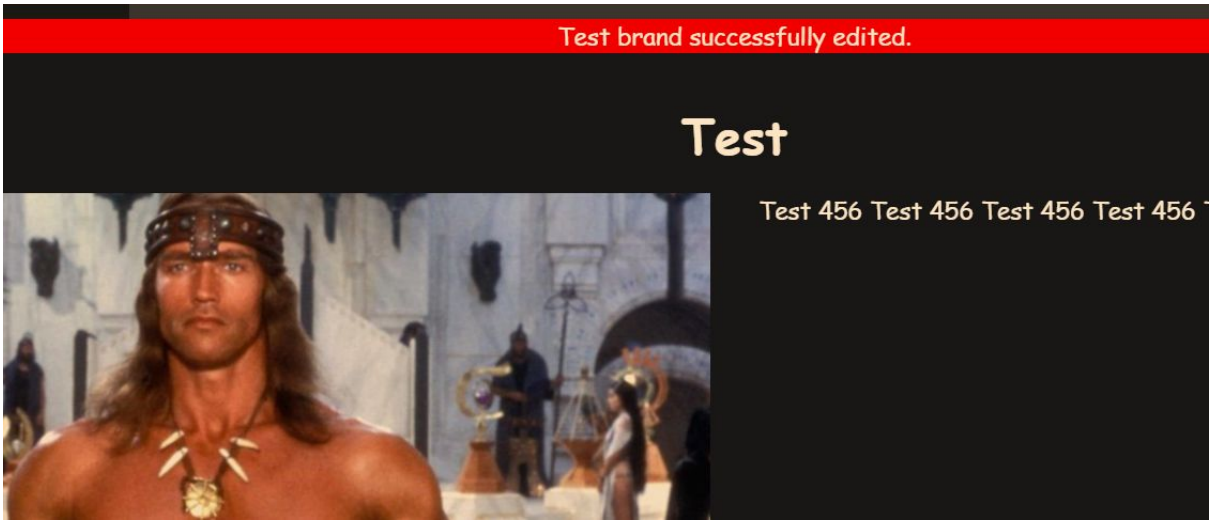


Photo changed. Name and description remain unchanged.



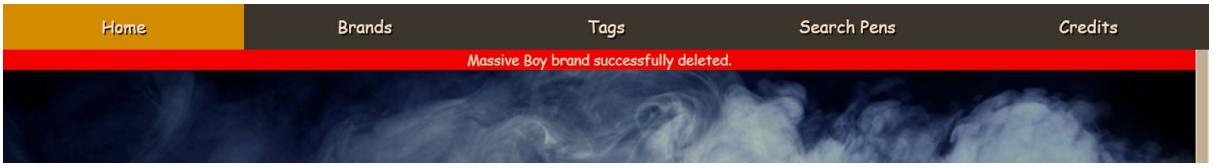


Delete Brand

```python
# form to delete a brand from the database
@app.route('/delete_brand/<int:id>')
def delete_brand(id):
    brand = db.session.query(models.Brand).filter(models.Brand.id==id).first_or_404()
    # delete saved photo from files
    if brand.photo:
        photo_file = delete_photo(brand.photo)
    # flash message to let user know that the brand has been deleted
    flash('{} brand successfully deleted.'.format(brand.name))
    # delete brand from database
    db.session.delete(brand)
    db.session.commit()
    return redirect(url_for('home'))
```

```python
# delete brand photo from computer files
def delete_photo(form_photo):
    photo_path = os.path.join(app.root_path, 'static', 'images', 'brands', form_photo)
    os.remove(photo_path)
    return
```

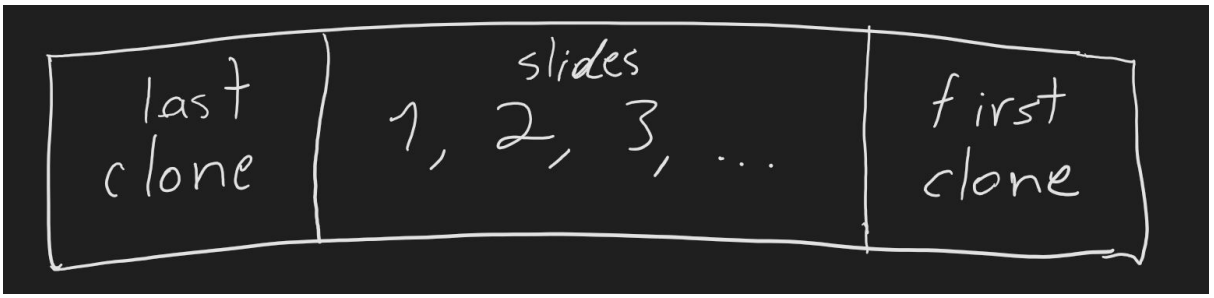
Pressing the delete button on a brand will delete the brand image file from the server and delete the brand from the brand table. The user is then redirected to the homepage with a flash message that informs the user which brand has been deleted. An improvement I would've made to this function would be to add a confirmation dialogue before deleting the brand. This is necessary for the HCI user experience considerations as accidental clicks can occur which may result in unwanted deletion of a brand.



All Tags / Slideshow

This page is a section of the home page. The all tags page is displayed in a carousel slideshow style. The left and right arrows are used to scroll through the slideshow.

The slideshow is looped, meaning that if the user is on the last frame and clicks next, it will automatically loop them back to the first frame in the slideshow; and vise-versa. This is achieved by creating a clone of the first frame and placing it at the end and a clone of the last frame placed at the start. When the user reaches the last slide (the clone of the first slide) it will instantly switch the original first slide without any transitions. I tested the robustness of the slideshow to make sure it works properly under unexpected situations such as the user spamming the next/previous buttons.



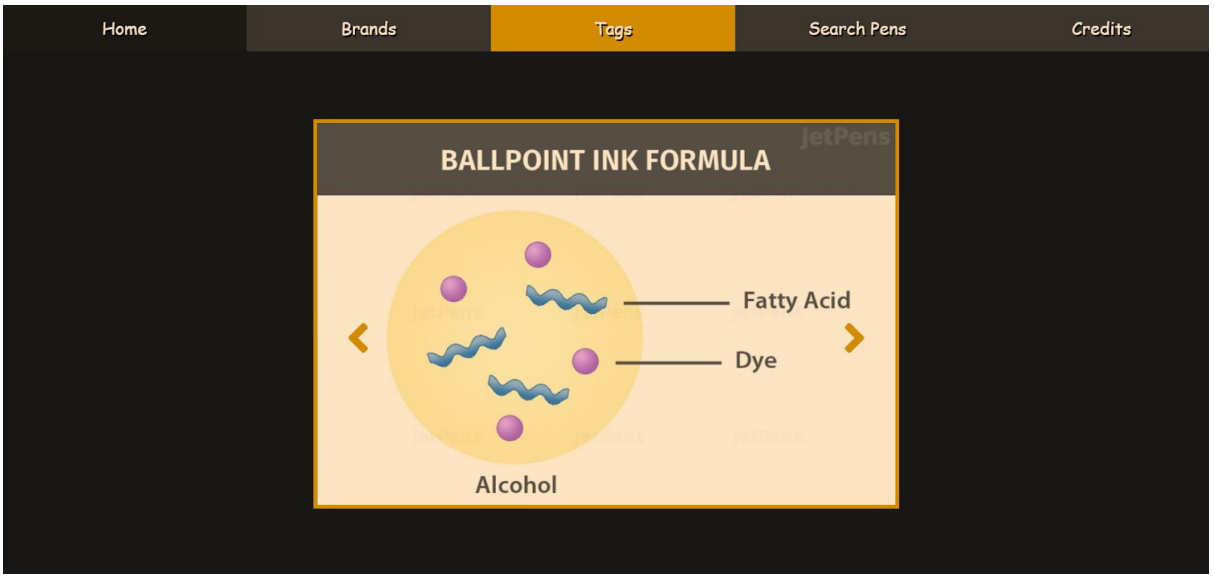
HTML:

```html
<div class="section" id="tags">
    <div id="tags-grid">
        <i id="prev-button" class="fas fa-angle-left"></i>
        <i id="next-button" class="fas fa-angle-right"></i>
        <!-- get length of tags list and add two to make up for lastClone and firstClone -->
        <div id="slideshow" style="grid-template-columns: repeat({{ tags|length + 2 }}, 100%);">
            <!-- last tag repeats at the start to create seamless loop -->
            {% set lastTag = tags|last %}
            <div class="slide" id="lastClone">
                <img src="/static/images/tags/{{ lastTag.photo }}">
                <a href="{{ url_for('tag', id = lastTag.id) }}"><p>{{ lastTag.name }}</p></a>
            </div>
            {% for tag in tags %}
            <div class="slide">
                <img src="/static/images/tags/{{ tag.photo }}">
                <a href="{{ url_for('tag', id = tag.id) }}"><p>{{ tag.name }}</p></a>
            </div>
            {% endfor %}
            <!-- first tag repeats at the end -->
            {% set firstTag = tags|first %}
            <div class="slide" id="firstClone">
                <img src="/static/images/tags/{{ firstTag.photo }}">
                <a href="{{ url_for('tag', id = firstTag.id) }}"><p>{{ firstTag.name }}</p></a>
            </div>
        </div>
    </div>
</div>
```

Javascript:

```
// carousel slideshow
var slideshow;
var slides;
var counter;
var size;

slideshow = () =>{
    // parents with grid property
    slideshow = document.querySelector('#slideshow');
    if(slideshow == null){
        return;
    }
    // get list of all slide divs
    slides = document.querySelectorAll('.slide');
    // console.log("Slide length: " + slides.length);

    // Buttons
    const prevButton = document.querySelector('#prev-button');
    const nextButton = document.querySelector('#next-button');

    // Counter
    counter = 1;
    size = slideshow.clientWidth;

    slideshow.style.transform = 'translateX(' + (-size * counter) + 'px)';

    // Button Listeners
    nextButton.addEventListener('click', ()=>{
        if(counter >= slides.length - 1) return;
        slideshow.style.transition = "transform 0.4s ease-in-out";
        counter++;
        slideshow.style.transform = 'translateX(' + (-size * counter) + 'px)';
        // console.log("User clicked next | Counter: " + counter);
    });
```

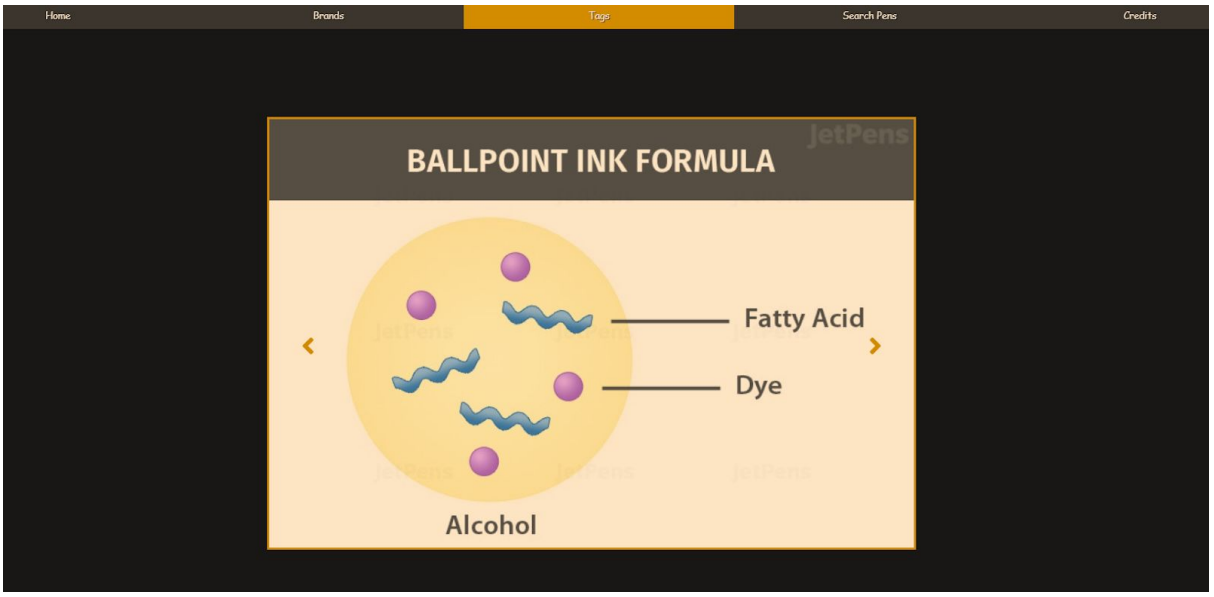
```
    prevButton.addEventListener('click', ()=>{
        if(counter <= 0) return;
        slideshow.style.transition = "transform 0.4s ease-in-out";
        counter--;
        slideshow.style.transform = 'translateX(' + (-size * counter) + 'px)';
        // console.log("User clicked prev | Counter: " + counter);
    });

    // Slideshow Listeners
    slideshow.addEventListener('transitionend', ()=>{
        // Create loop at the start that goes back to the last slide (clone of last slide at the start)
        if(slides[counter].id ===  'lastClone'){
            slideshow.style.transition = "none";
            counter = slides.length - 2;
            slideshow.style.transform = 'translateX(' + (-size * counter) + 'px)';
            // console.log("Landed on first slide, going back to lastClone | Counter: " + counter);
        }
        // Create loop at the end that goes back to the first slide (clone of first slide at the end)
        if(slides[counter].id ===  'firstClone'){
            slideshow.style.transition = "none";
            counter = slides.length - counter;
            slideshow.style.transform = 'translateX(' + (-size * counter) + 'px)';
            // console.log("Landed on last slide, going back to firstClone | Counter: " + counter);
        }
    });

}
```



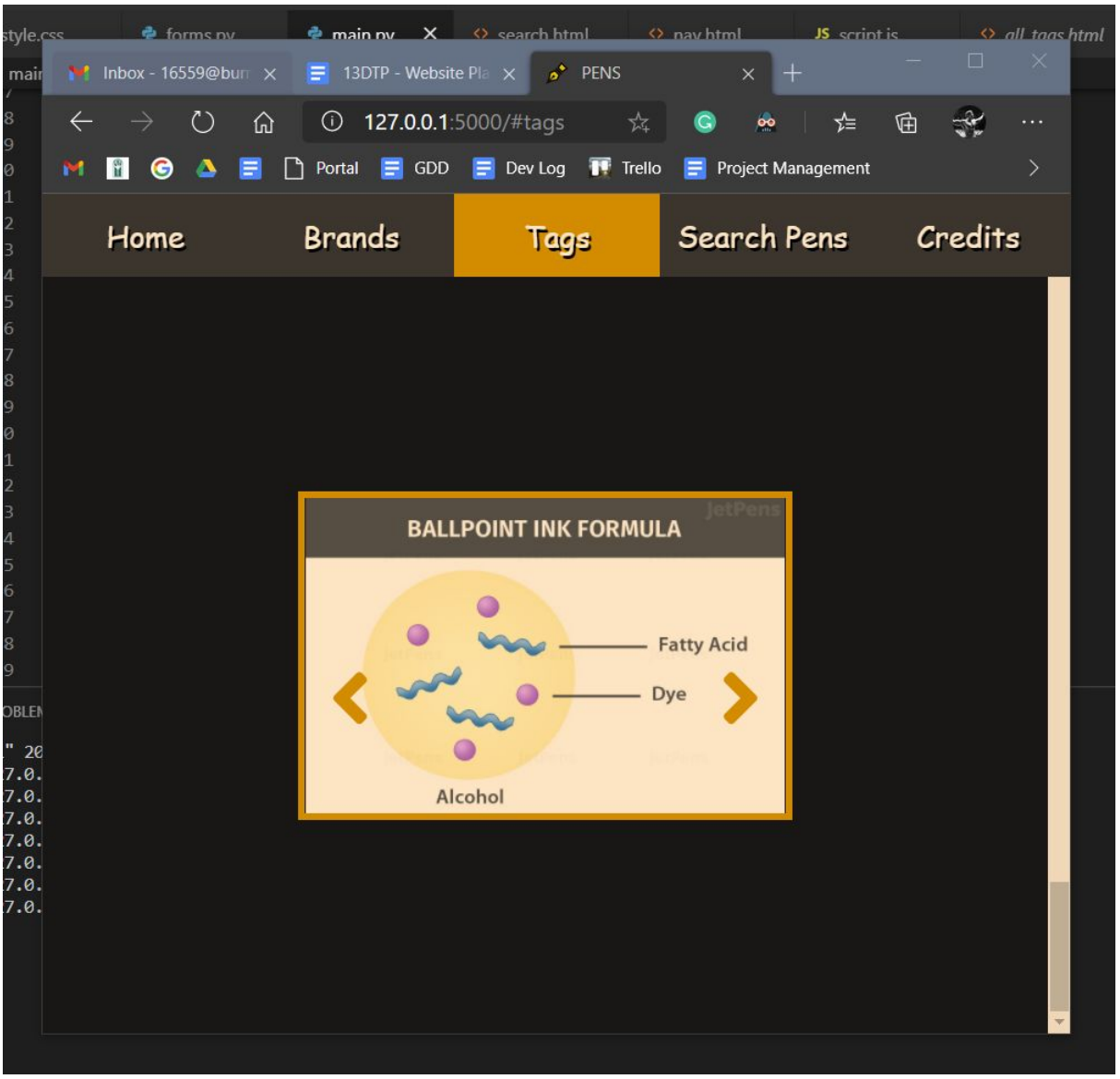
I tested the flexibility of the slideshow by resizing the window and zooming in and out. The code is able to adapt and update the page to the new viewing conditions.

Zoomed in to 200%:

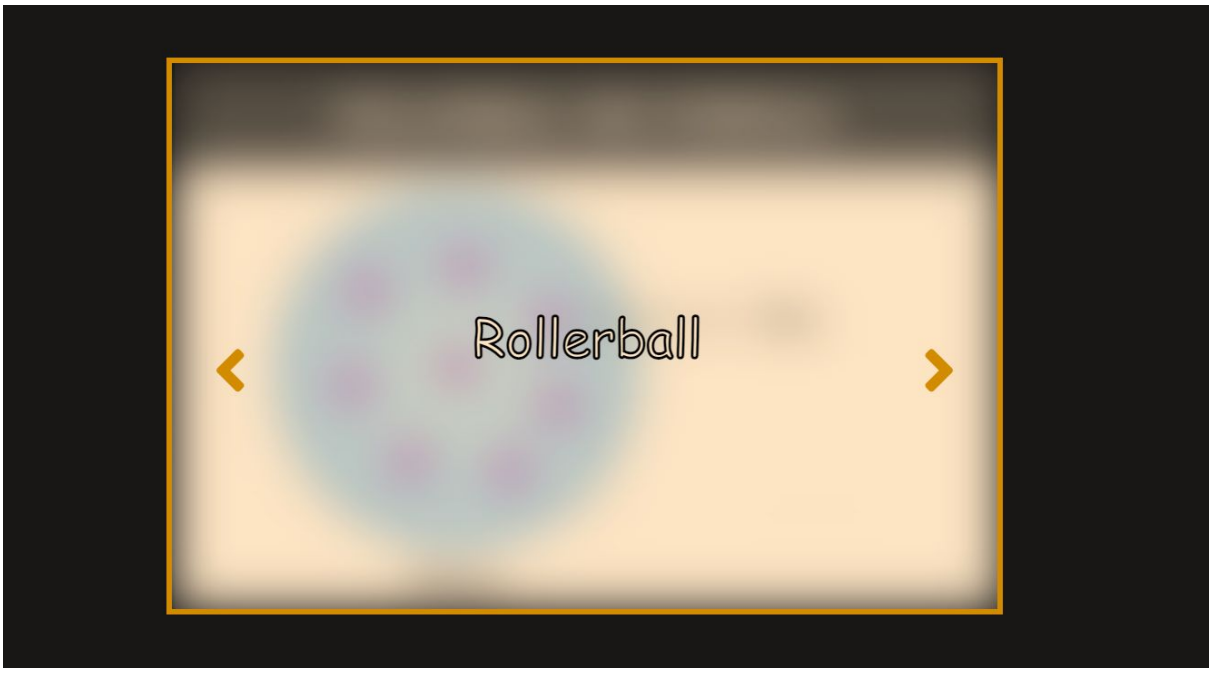
Zoomed out to 50%:



Window resized:

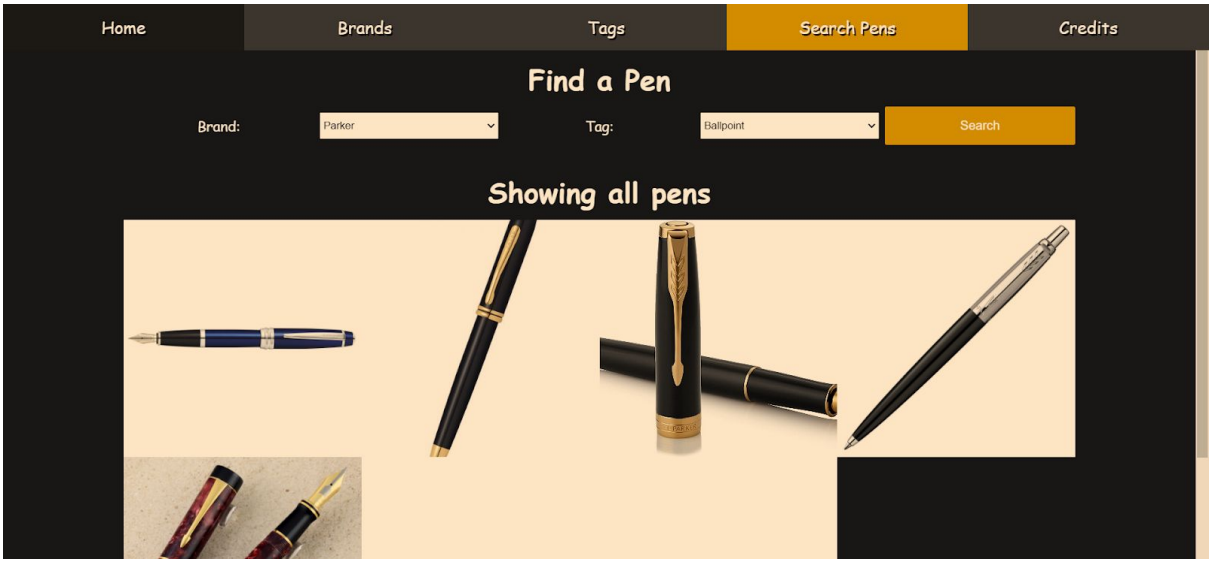


Hovering the image will play a blurring animation and display a link. The user can click this to be taken to the page for that tag.

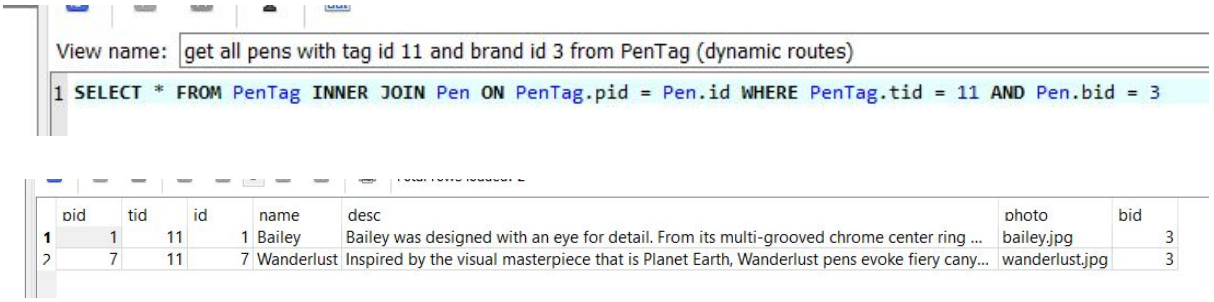


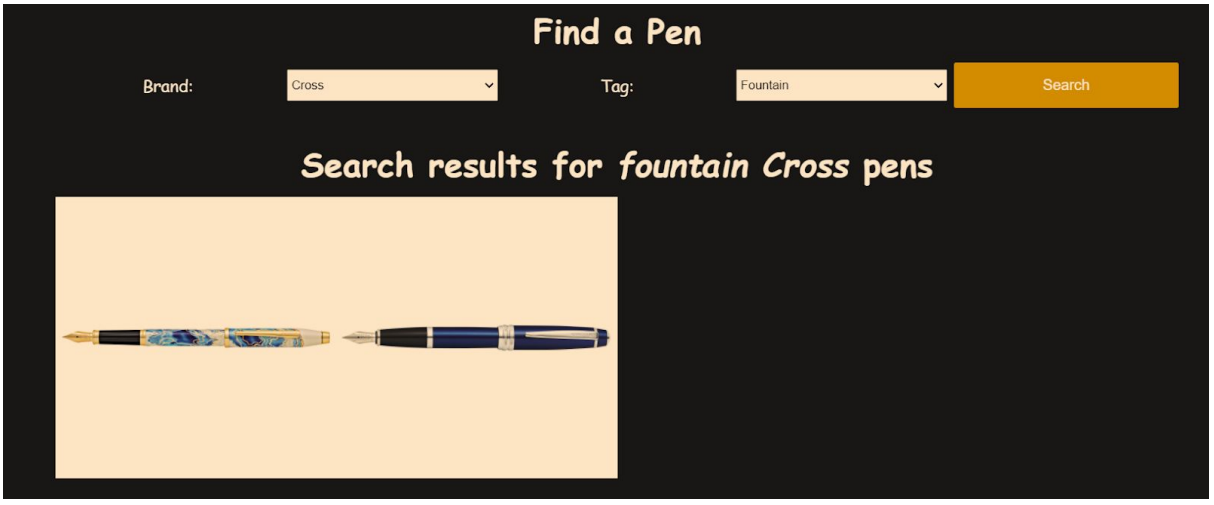
Search Pens / All Pens

The Search Pens page contains a grid of flip cards of all the pens similar to the all brands page. By default the grid is populated with all of the pens in the database. This is the page displayed by clicking search pens before any searching is performed.

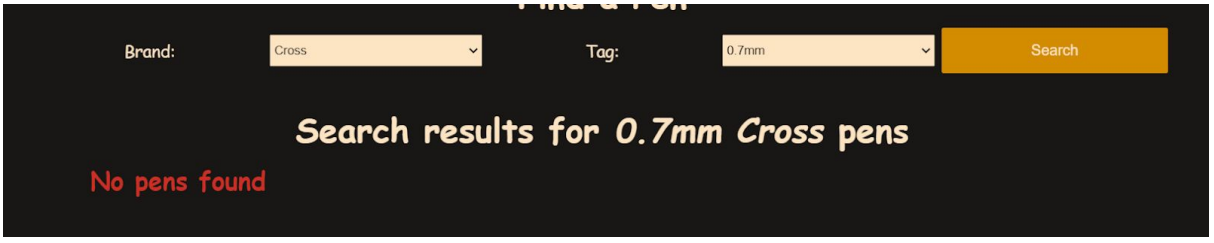


This is the page after a search for “fountain cross” pens is performed.





If no pens are found for the searched options, the page will display no pens found text.



More comprehensive testing is performed for the search function in the data integrity section at the bottom of this document.

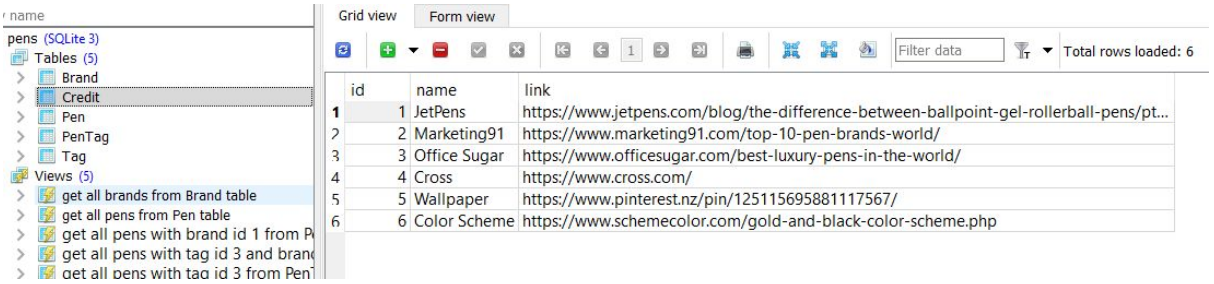
Credits Page

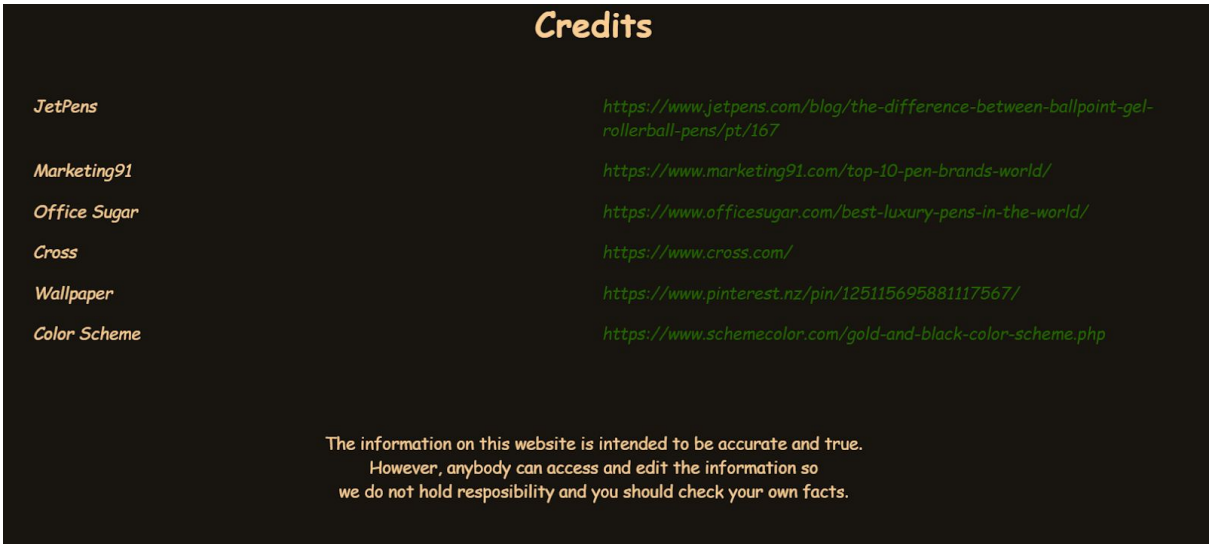
The credits page contains a table of websites and resources used (text and images) on the website which need to be credited for legal reasons.

At the bottom, there is a disclaimer that states that the information on this website is intended to be true but may not be 100% accurate. It is a moral obligation to provide genuine information to the best of our ability.

However, this is to ensure that we do not encounter any moral or legal issues from providing inaccurate information.

The credits route calls for the Credit table in the database. This is used to populate the credits on the page. I ensured this works correctly by checking the information in the database view matches the page displayed on the website.



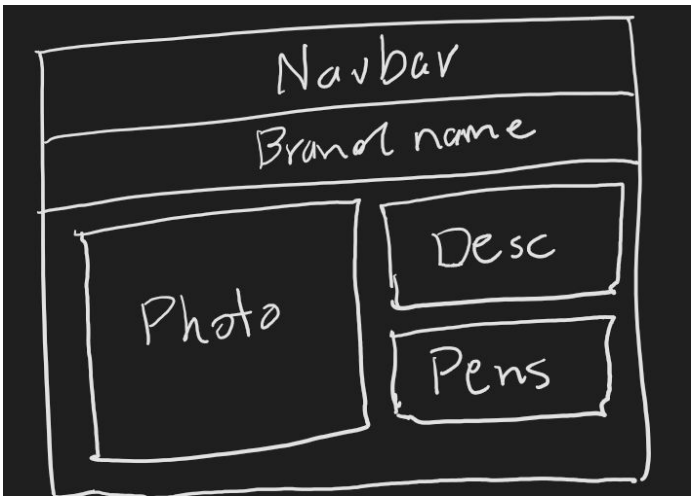


Brand Page

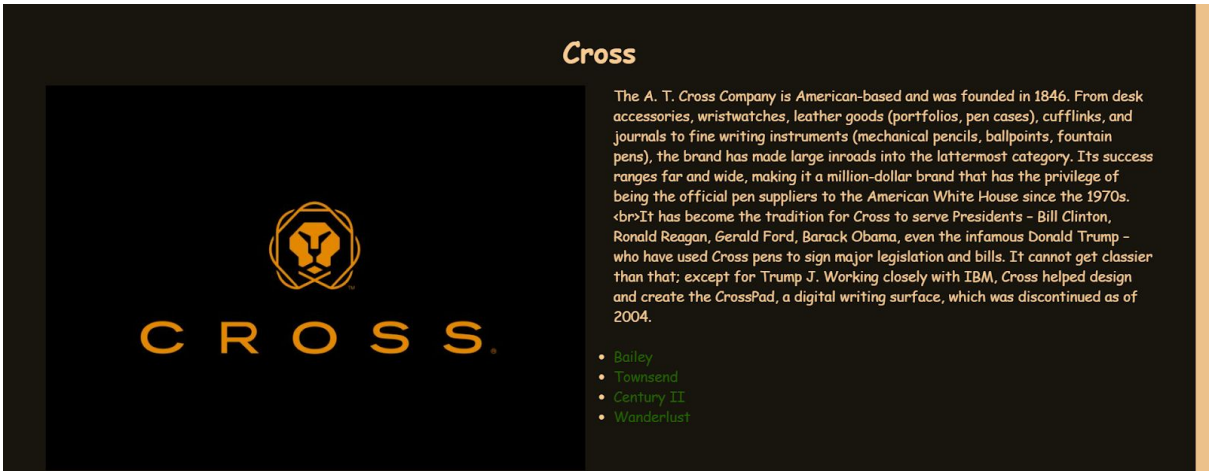
The brand page contains information about a certain brand. The route takes the brand id through the url path. The id is then used to gather the brand information from the Brand table in the database. I use sqlalchemy here to create a query of the Brands and then filter by the brand id. I used first_or_404 as a fallback option so if the id requested is not valid/found, the user will be redirected to the 404 page.

```python
# brand page route
@app.route('/brand/<int:id>')
def brand(id):
    brand = models.Brand.query.filter_by(id=id).first_or_404()
    return render_template('brand.html', page_title=' BRANDS', brand = brand)
```

The page will contain the brand name at the top. The brand photo will be on the left side of the page and the description and brand pens will be on the right side as shown.



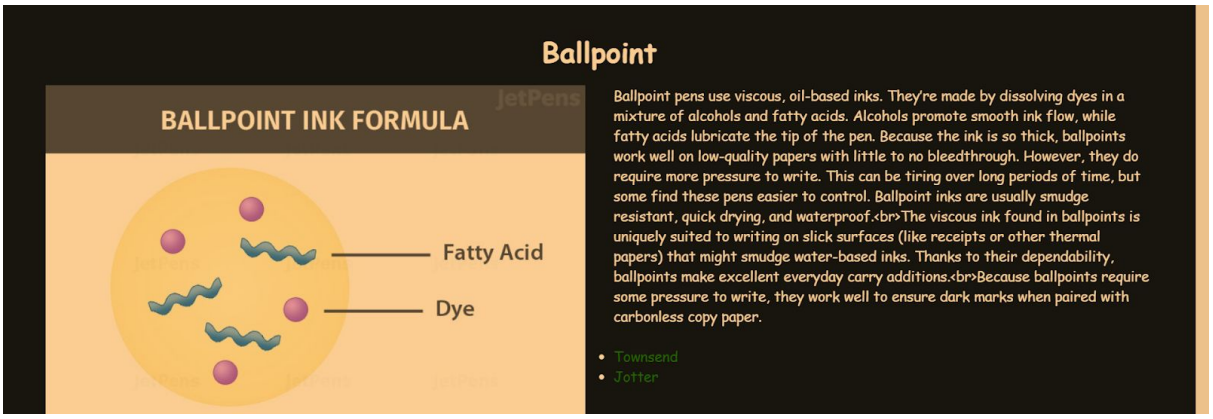
Brand Page route (brand id = 3)



Tag Page

The tag page is a duplicate of the brand page with the brand content replaced by tag content. Same testing has been performed on this route.

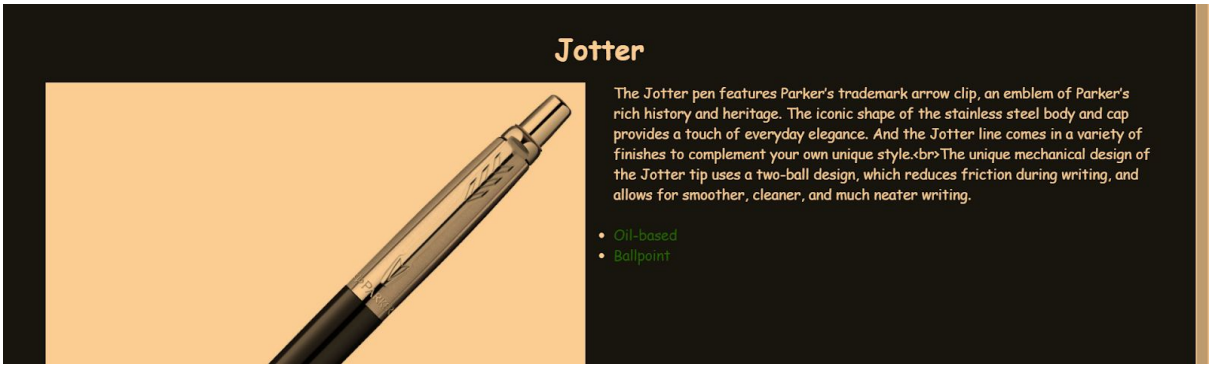
Tag Page route (tag id = 1)



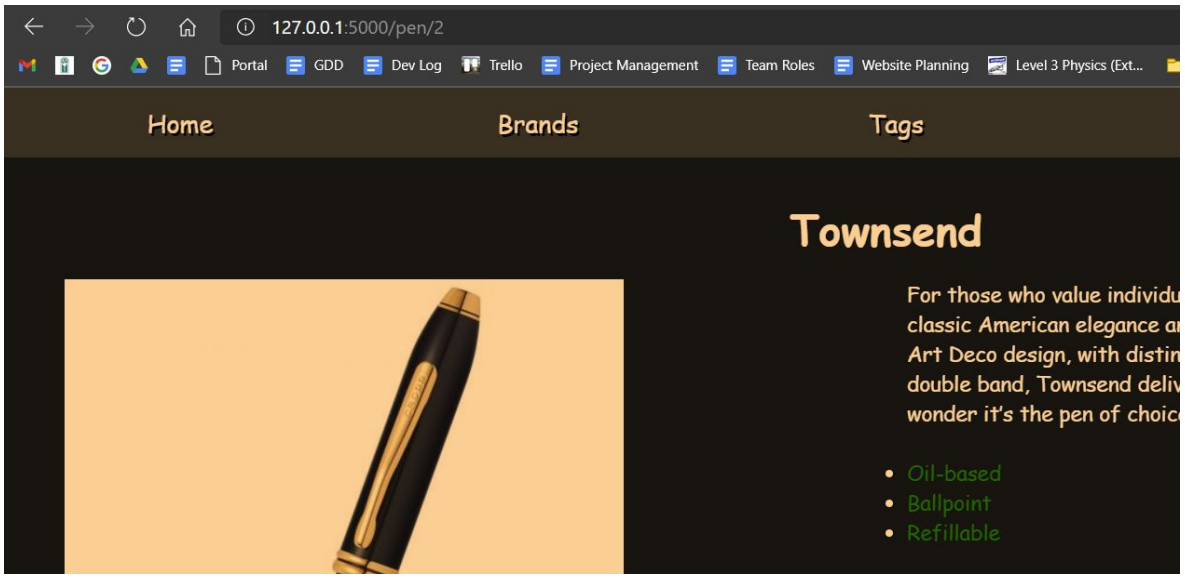
Pen Page

The penpage is a duplicate of the brand page with the brand content replaced by pen content. Same testing has been performed on this route.

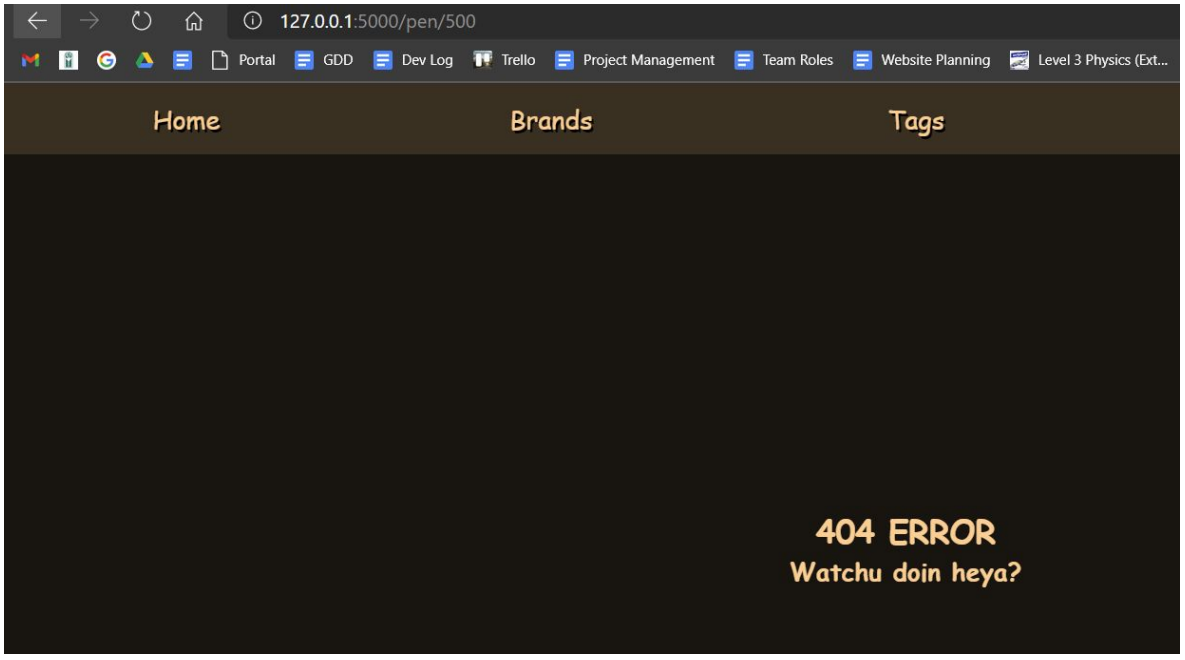
Pen Page route (pen id = 4)



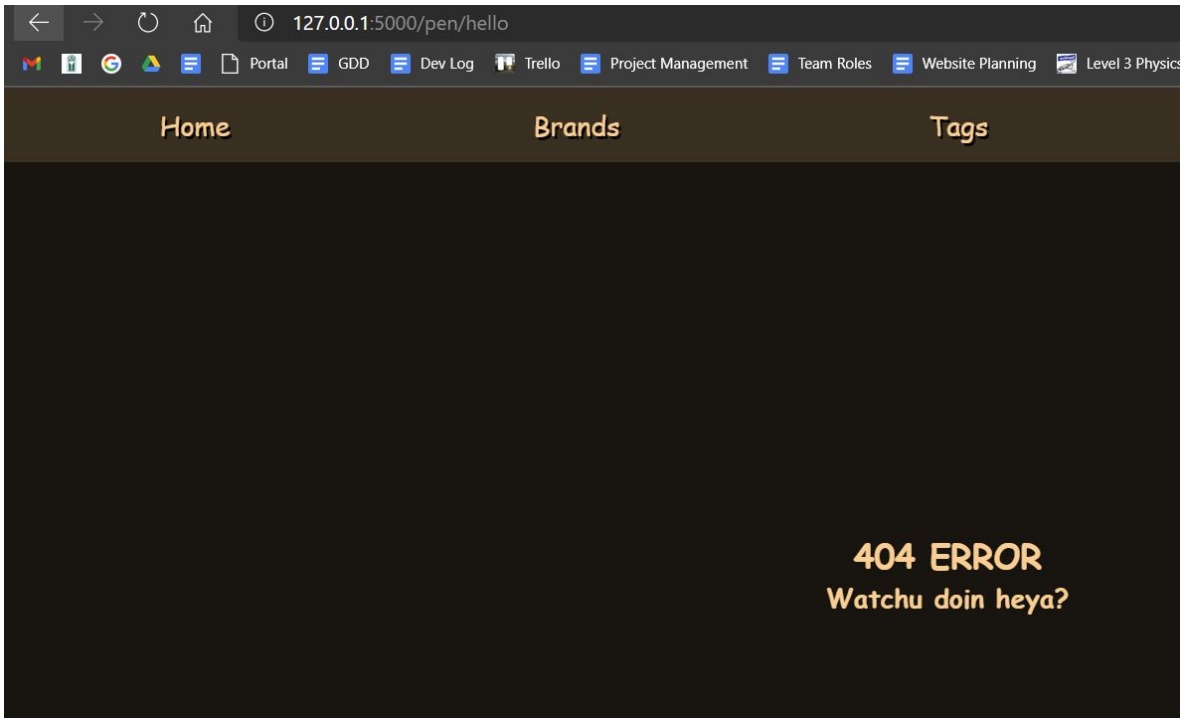
Boundary Testing for Pen Page route (repeated for Brand Page and Tag Page. As the code it the same, documentation of testing is not necessary)
Expected case: pen id = 2 (Townsend pen page)



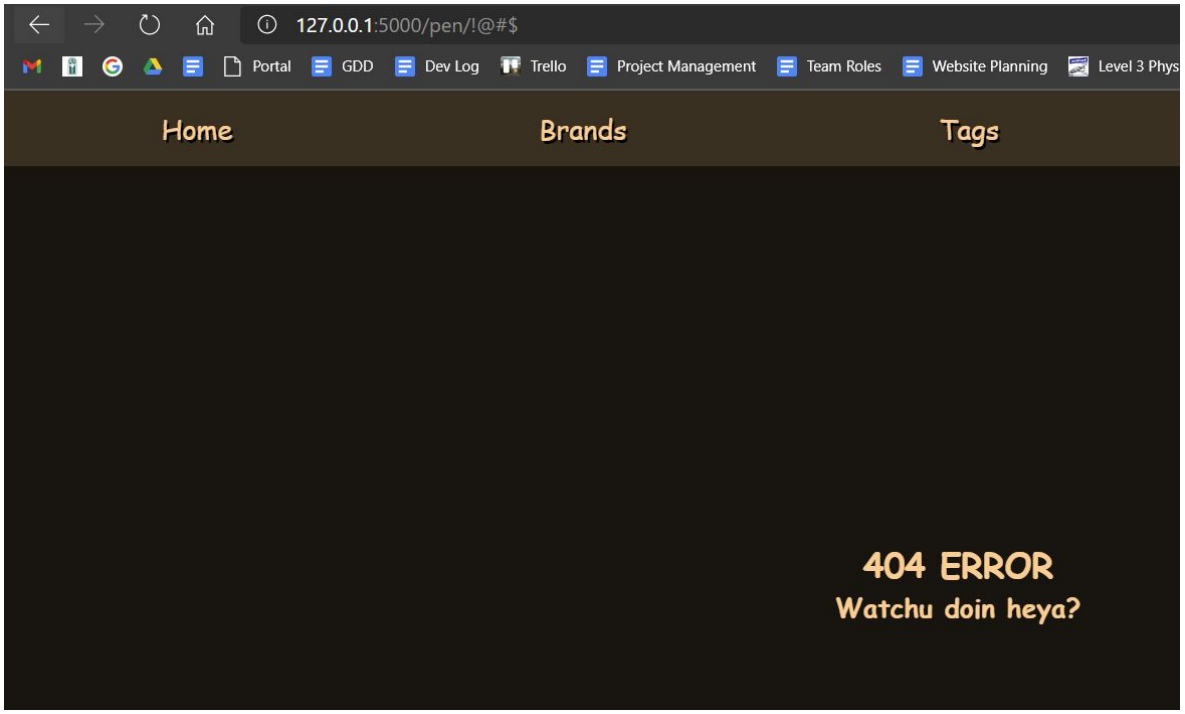
Boundary case: pen id = 500 (no pen with id 500 exists)



Boundary case: pen id = hello (no pen with id “hello” exists)



Boundary case: pen id = !@#$ (random letters used to make sure the website is able to handle unexpected situations)

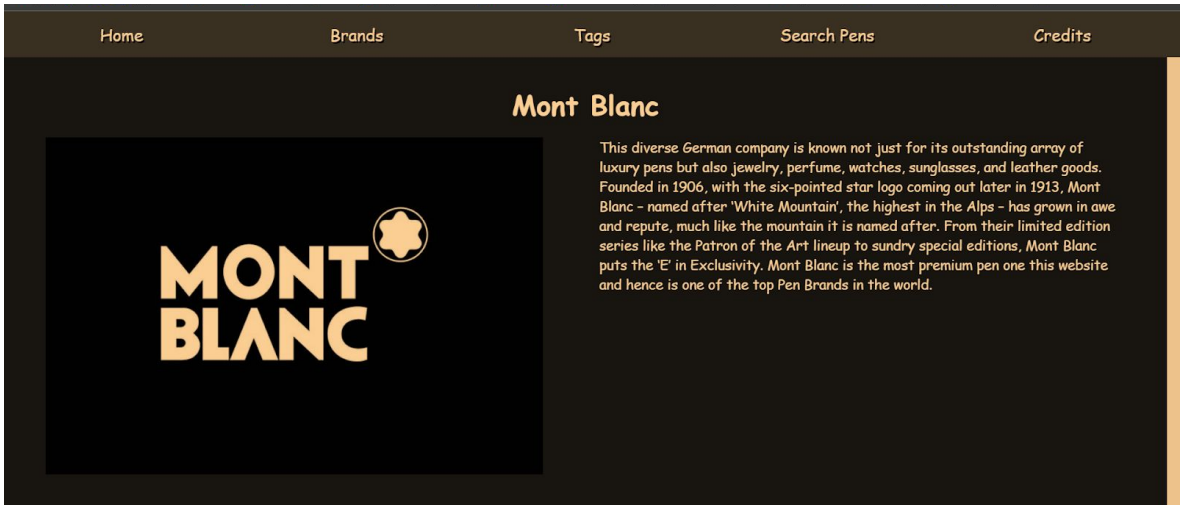


Dynamic screen

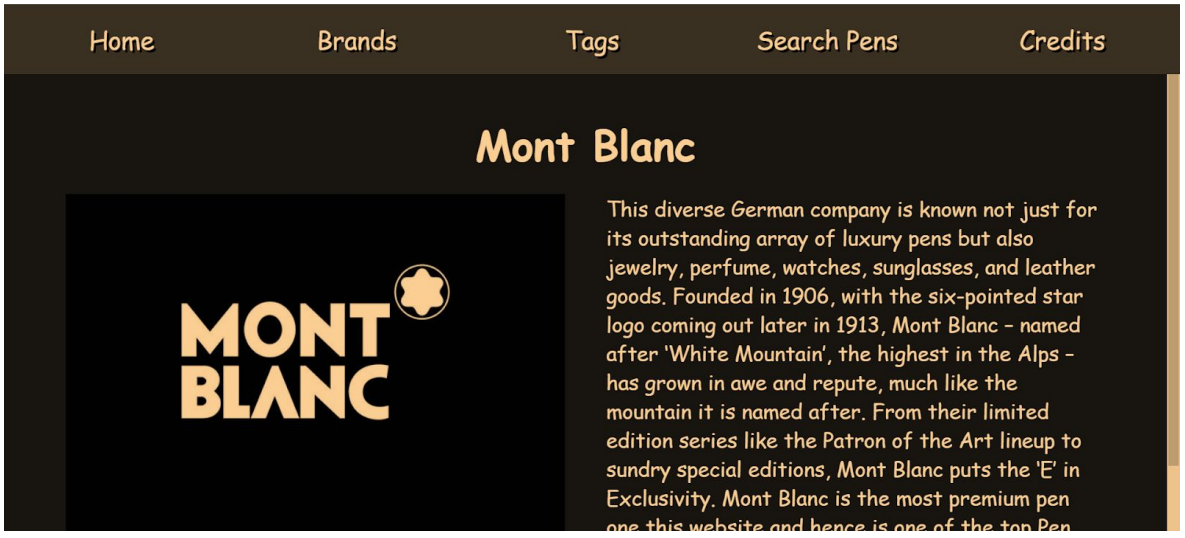
The website is fully optimized for all computer screens of different sizes. The pages have been tested on both small laptops and large desktops to ensure the content is still readable.

Brand Page

100% Normal

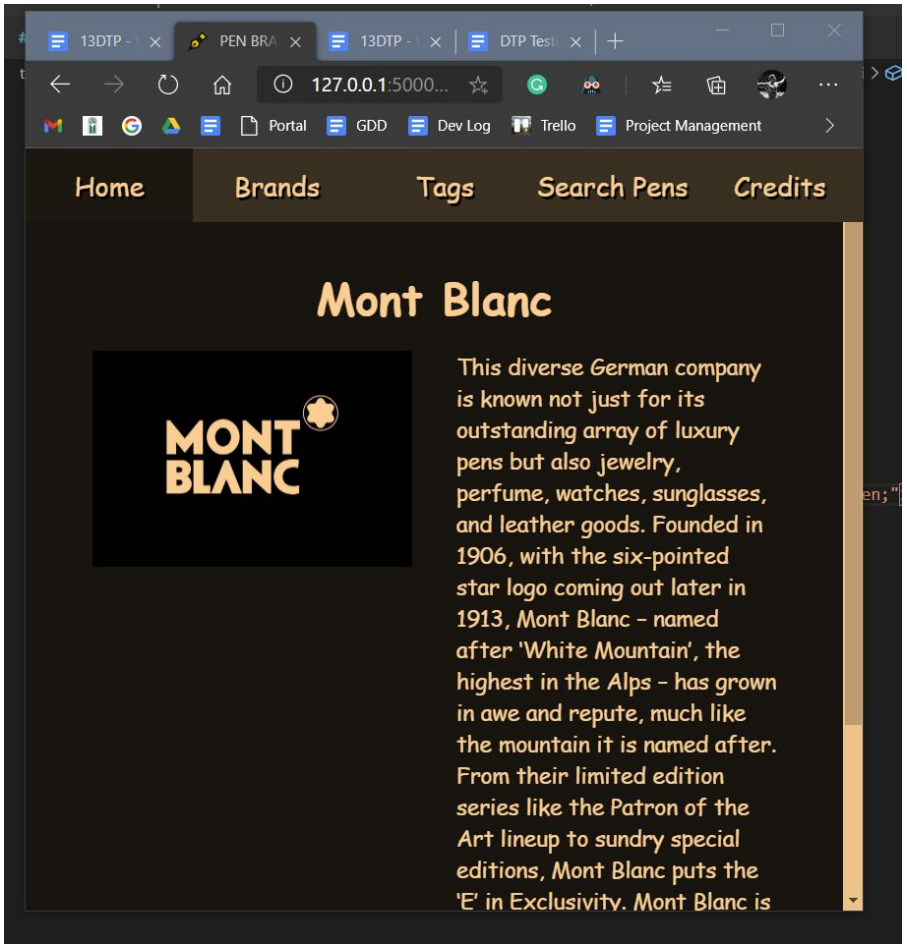


150% Zoomed in (similar view on laptop screens)



75% Zoomed out (similar view on ultrawide monitors)

Resized window



Data integrity

Here are some examples of testing performed to ensure all features are working properly.

Description	Action	Expected Result	Actual Result
Clicking on a brand will show the correct brand page	Click on Parker brand “Read more”	Show brand page for Parker where id = 1.	Shows brand page for Parker with id = 1.

Evidence

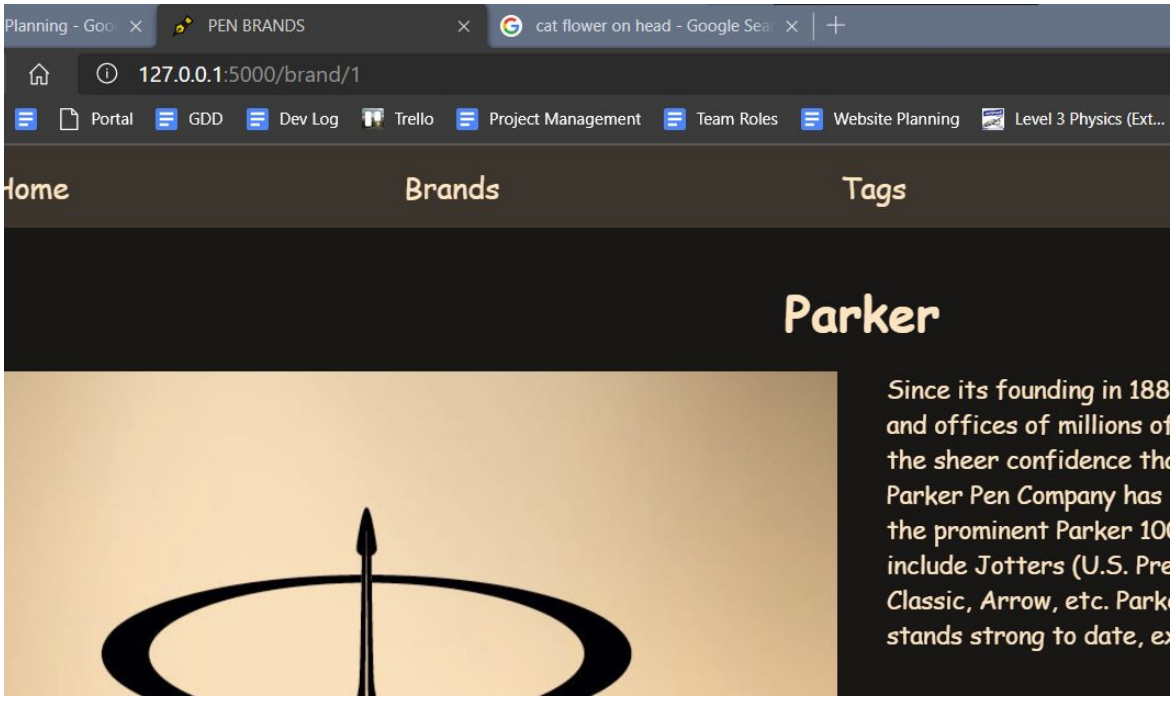
Brand table in SQLite:

	id	name	desc
1	1	Parker	Since its founding in 1888, Parker has written its way into the homes, classrooms, and off...

Brand query on main.py:

```
# pen page route
@app.route('/pen/<int:id>')
def pen(id):
    pen = models.Pen.query.filter_by(id=id).first_or_404()
    return render_template('pen.html', page_title=(' PENS'), pen = pen)
```

Brand page on website:



Description	Action	Expected Result	Actual Result
Brand page will	Viewing parker	The three pens	Correct pens

show the correct number of pens for the brand	brand page	with brand parker (bid = 1) are Sonnet, Jotter, and Duofold.	shown for bid = 1.

View name: get all pens with brand id 1 from Pen (dynamic routes)

```
1 SELECT * FROM Pen WHERE Pen.bid = 1
```

	id	name	desc	photo	bid
1	3	Sonnet	Handcrafted and individually checked for flawless quality, Parker Sonnet is full of meticu...	sonnet.jpg	1
2	4	Jotter	The Jotter pen features Parker's trademark arrow clip, an emblem of Parker's rich history ...	jotter.jpg	1
3	5	Duofold	A symbol of excellence since 1921, Duofold remains the most distinguished pen in Park...	duofold.jpg	1

```
<br>
<!-- list all pens for this brand -->
<ul>
    {% for pen in brand.pens %}
    <li><a href="{{ url_for('pen', id = pen.id) }}" style="color: green;">{{ pen.name }}</a></li>
    {% endfor %}
</ul>
```

Description	**Action**	**Expected Result**	**Actual Result**
Get a list of all the pens with a certain tag id using PenTag table (dynamic routes)	Viewing water-based tag page	The five pens with water-based tag (tid = 3) are Sonnet, Duofold, Bailey, Wanderlust, Century II.	Correct pens shown for tid = 3.

View name: get all pens with tag id 3 from PenTag (dynamic routes)

```
1 SELECT * FROM PenTag INNER JOIN Pen ON PenTag.pid = Pen.id WHERE PenTag.tid = 3
```

	pid	tid	id	name	desc	photo	bid
1	3	3	3	Sonnet	Handcrafted and individually checked for flawless quality, Parker Sonnet is full of meticu...	sonnet.jpg	1
2	5	3	5	Duofold	A symbol of excellence since 1921, Duofold remains the most distinguished pen in Park...	duofold.jpg	1
3	1	3	1	Bailey	Bailey was designed with an eye for detail. From its multi-grooved chrome center ring ...	bailey.jpg	3
4	7	3	7	Wanderlust	Inspired by the visual masterpiece that is Planet Earth, Wanderlust pens evoke fiery cany...	wanderlust.jpg	3
5	6	3	6	Century II	Embracing both tradition and evolution—the iconic Cross profile takes a broader point ...	century_ii.jpg	3

```python
# tag page route
@app.route('/tag/<int:id>')
def tag(id):
    tag = models.Tag.query.filter_by(id=id).first_or_404()
    return render_template('tag.html', page_title=' TAGS', tag = tag)
```

```
<!-- list all pens for this tag -->
<ul>
    {% for pen in tag.pens %}
    <li><a href="{{ url_for('pen', id = pen.id) }}" style="color: green;">{{ pen.name }}</a></li>
    {% endfor %}
</ul>
```

Description	Action	Expected Result	Actual Result
Get a list of all the pens with a certain tag id and brand id using PenTag table (dynamic routes with inner join)	Viewing search pens page and searching for water-based Parker pens.	The two pens with water-based tag (tid = 3) and Parker brand (bid = 1) are Sonnet, Duofold.	Correct pens shown for tid = 3 and bid = 1.

View name: get all pens with tag id 3 and brand id 1 from PenTag (dynamic routes)

```
SELECT * FROM PenTag INNER JOIN Pen ON PenTag.pid = Pen.id WHERE PenTag.tid = 3 AND Pen.bid = 1
```

Total rows loaded: 2

	pid	tid	id	name	desc	photo	bid
1	3	3	3	Sonnet	Handcrafted and individually checked for flawless quality, Parker Sonnet is full of meticu...	sonnet.jpg	1
2	5	3	5	Duofold	A symbol of excellence since 1921, Duofold remains the most distinguished pen in Park...	duofold.jpg	1

In the code below, we first gather all the necessary queries, then populate the search fields. We collect the user's selected search choices and store them in variables. Once search is submitted, we create a filter of the brand id and tag id pens and then combine the mutual pens between the two lists. Then we render the results to the user.

```python
# search form
@app.route('/search', methods = ['GET', 'POST'])
def search():
    form = Search()
    pens = models.Pen.query.all()
    brands = models.Brand.query.all()
    tags = models.Tag.query.all()
    # populate the dropdown selections
    form.brand.choices = [(brand.id, brand.name) for brand in brands]
    form.tag.choices = [(tag.id, tag.name) for tag in tags]

    selected_tag = form.tag.data
    selected_brand = form.brand.data

    # if no search done yet
    results = pens
    has_searched = False

    if request.method == 'POST':
        if form.validate_on_submit():
            has_searched = True
            brand = models.Brand.query.filter_by(id = selected_brand).first()
            tag = models.Tag.query.filter_by(id = selected_tag).first()
            # Get a list of the pens common between the brand and the tag chosen.
            results = list(set(brand.pens).intersection(tag.pens))
            # render the search results
            return render_template('search.html', title = "Find a Pen", form = form, results = results,
                                   brand = brand.name, tag = tag.name, has_searched = has_searched)
        else:
            abort(404)
    return render_template('search.html', title = "Find a Pen", form = form, results = results, has_searched = has_searched)
```

This code displays the html for the search page. The pens grid contains all of the filtered pens in a flip card style.

```
<!-- search results | display result of all pens if search not done yet -->
{% if has_searched == true %}
<h1 style="margin: auto;">Search results for <i>{{ tag|lower }} {{ brand }}</i> pens</h1>
<div id="pens-grid">
  <!-- display a list of all pens found -->
  {% if results|length != 0 %}
  {% for pen in results %}
  <div class="card">
      <div class="card-content" onclick="flip(this)">
          <!-- pen photo as a background image -->
          <div class="card-photo" style="background-image: url('/static/images/pens/{{ pen.photo }}')"></div>
          <!-- pen information -->
          <div class="card-info">
              <h2 class="title">{{ pen.name }}</h2>
              <p class="description">{{ pen.desc }}</p>
              <a class="readmore" href="{{ url_for('pen', id = pen.id) }}">Read more</a>
          </div>
      </div>
  </div>
  {% endfor %}
  <!-- if no pens are found for search options -->
  {% else %}
  <h2 style="color: #d83737;">No pens found</h2>
  {% endif %}
</div>
```

Cards flipped to show pen name.

Description	Action	Expected	Actual Result

		Result	
Get a list of all the pens with a certain tag id and brand id using PenTag table (dynamic routes with inner join)	Viewing search pens page and searching for permanent Parker pens.	Boundary case. No pens found as there are no “permanent parker” pens in the database.	No pens shown for permanent tag (tid = 8) and Parker brand (bid = 1).