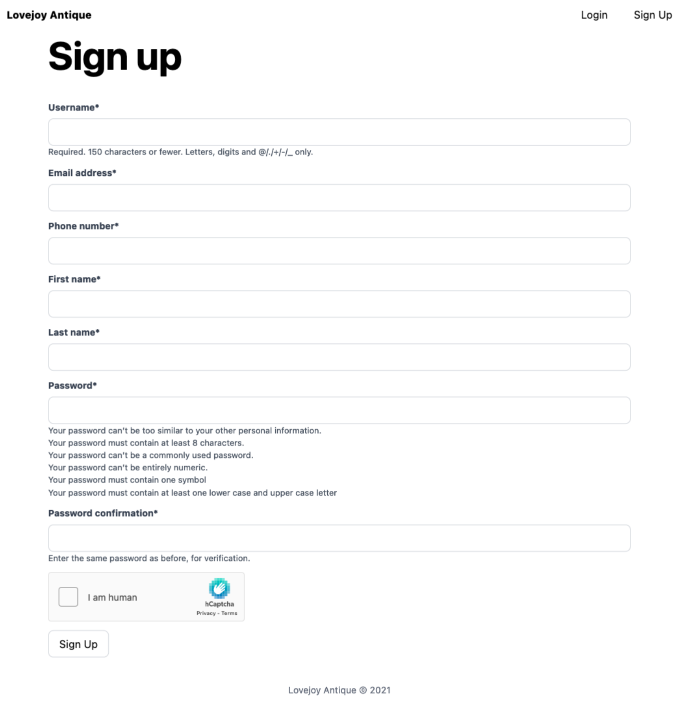
Introduction to Computer Security : Lovejoy Antique (Django)

Application URL: https://lovejoyantiques.xyz
Code: Zip file Location:
https://universityofsussex-my.sharepoint.com/:u:/g/personal/ep396_sussex_ac_uk/ERHQTpNu2JxJgYH8dYw6LBABqMvz_8aN5NKi2V0tMncMIw

Task 1



Lovejoy Antique

Login Sign Up

Sign up

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email address*

Phone number*

First name*

Last name*

Password*

Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.
Your password must contain one symbol
Your password must contain at least one lower case and upper case letter

Password confirmation*

Enter the same password as before, for verification.

I am human

Sign Up

Lovejoy Antique © 2021

Registration form code

accounts/models.py

```python
class User(AbstractUser):
   email = models.EmailField(_("email address"), unique=True, null=False,
blank=False)
   first_name = models.CharField(max_length=50, null=False, blank=False)
   last_name = models.CharField(max_length=50, null=False, blank=False)
   phone_number = PhoneNumberField(unique=True, null=False, blank=False)

   REQUIRED_FIELDS = ("email", "first_name", "last_name", "phone_number")

   def __str__(self):
       return self.username
```

accounts/forms.py
```python
class UserCreationForm(UserCreationForm):
   class Meta:
       model = User
       fields = ("username", "email", "phone_number", "first_name", "last_name")
```

accounts/templates/accounts/signup.html
```html
{% extends 'base.html' %}
{% load static %}

{% load tailwind_filters %}

{% if site_key %}
   {% block scripts %}<script src="https://js.hcaptcha.com/1/api.js" async
defer></script>{% endblock scripts %}
{% endif %}

{% block title %}Sign Up{% endblock title %}

{% block content %}
   <h2 class="text-6xl font-bold pb-10">Sign up</h2>
   <form method="post">
       {% csrf_token %}
       {{ form|crispy }}
       {% if site_key %}
           <div class="h-captcha mb-3" data-sitekey="{{ site_key }}"></div>
       {% endif %}
       <div>
           <button type="submit" class="py-2 bg-white border rounded-lg px-4
border-gray-300 ">Sign Up</button>
       </div>
   </form>
{% endblock content %}
```

Code when registration form submitted

accounts/views.py
```python
def signup_view(request):
   if request.user.is_authenticated:
       return redirect("/")

   form = UserCreationForm(request.POST or None)
   if settings.USE_HEROKU:
       data = {"form": form, "site_key": settings.HCAPTCHA_TOKEN}
   else:
       data = {"form": form}
   if request.method == "POST":
       if form.is_valid():
           if not check_hcaptcha(request):
               return redirect("accounts:signup")
           user = form.save(commit=False)
           user.is_active = False
           email = user.email
           user.save()

           mail_subject = "Activate your account."
           message = render_to_string(
               "accounts/validate_email.html",
               {
                   "user": user,
                   "domain": get_current_site(request).domain,
                   "uid": urlsafe_base64_encode(force_bytes(user.pk)),
                   "token": account_activation_token.make_token(user),
                   "protocol": "https" if request.is_secure() else "http",
               },
           )
           send_mail(mail_subject, message, settings.DEFAULT_FROM_EMAIL, [email])

           return render(request, "accounts/success_signup.html")

   return render(request, "accounts/signup.html", data)
```

lae/settings.py

```python
PASSWORD_HASHERS = [
   "django.contrib.auth.hashers.ScryptPasswordHasher",
   "django.contrib.auth.hashers.PBKDF2PasswordHasher",
   "django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher",
   "django.contrib.auth.hashers.Argon2PasswordHasher",
   "django.contrib.auth.hashers.BCryptSHA256PasswordHasher",
]
```

```python
if not DEBUG:
   SECURE_SSL_REDIRECT = True
   SESSION_COOKIE_SECURE = True
   SECURE_BROWSER_XSS_FILTER = True
   SECURE_HSTS_SECONDS = 31536000
   SECURE_HSTS_INCLUDE_SUBDOMAINS = True
   SECURE_HSTS_PRELOAD = True
   SECURE_CONTENT_TYPE_NOSNIFF = True
   CSRF_COOKIE_SECURE = True
```

```
AUTH_USER_MODEL = "accounts.User"
```

```
INSTALLED_APPS = [
   "django.contrib.admin",
   "django.contrib.auth",
   "django.contrib.contenttypes",
   "django.contrib.sessions",
   "django.contrib.messages",
   "django.contrib.staticfiles",
   "phonenumber_field",
   "crispy_forms",
   "crispy_tailwind",
   "app",
   "accounts",
   "storages",
]

MIDDLEWARE = [
   "whitenoise.middleware.WhiteNoiseMiddleware",
   "django.middleware.security.SecurityMiddleware",
   "django.contrib.sessions.middleware.SessionMiddleware",
   "django.middleware.common.CommonMiddleware",
   "django.middleware.csrf.CsrfViewMiddleware",
   "django.contrib.auth.middleware.AuthenticationMiddleware",
   "django.contrib.messages.middleware.MessageMiddleware",
   "django.middleware.clickjacking.XFrameOptionsMiddleware",
   "csp.middleware.CSPMiddleware",
]
```

Database Table

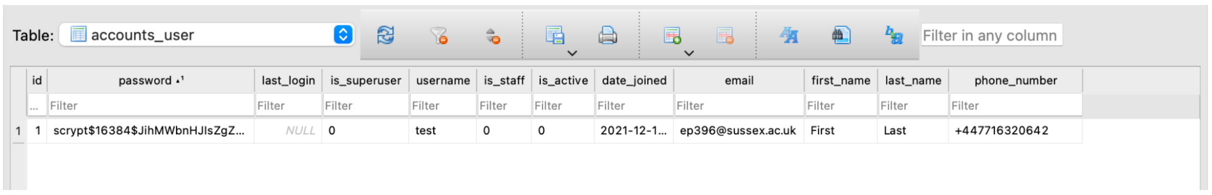


Table: accounts_user

Filter in any column

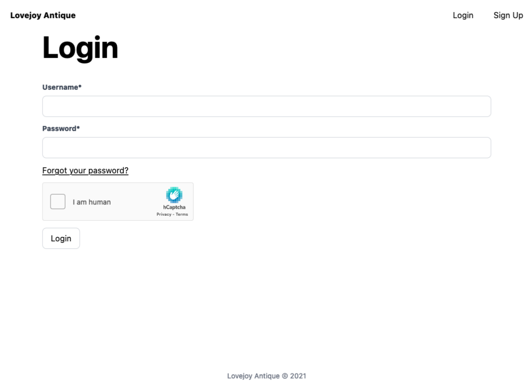
	id	password	last_login	is_superuser	username	is_staff	is_active	date_joined	email	first_name	last_name	phone_number
	...	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	scrypt16384JihMWbnHJIsZgZ...	NULL	0	test	0	0	2021-12-1...	ep396@sussex.ac.uk	First	Last	+447716320642

Why do you think it is secure?

- I have inherited Django abstract user class to create a User model which uses settings set in the settings.py file such as password validation and hashing
    - I use a 50 character max for the first and last name and make it required
    - I use an email field for the email as it checks if the email is in the correct format

        - I use an phone number field additionally for checking if the field content is a phone number or not
- I use Scrypt to hash the password and this is stored as the password, this also includes a salt which is a random value handled by the django authentication system
- I use Django templating, which automatically escapes html and sql inputs into forms by default
- I create the form using the User model and set the fields which the form will use
- I have implemented captcha in the template and view if the template is in production to try to prevent automation of the form, this is detailed further in task 4
- I have used an CSRF token to protect the form from CRSF attacks with the CRSF middleware
- I use the clickjack protection middleware
- I have enabled CSRF_COOKIE_SECURE which means CSRF cookies are only sent with an HTTPS connection when on a production server, same with session cookies

Task 2


Lovejoy Antique
Login Sign Up

Login

Username*

Password*

Forgot your password?

I am human

Login

Lovejoy Antique © 2021

Login Form code

accounts/templates/accounts/login.html

```
{% extends 'base.html' %}
{% load static %}

{% load tailwind_filters %}

{% if site_key %}
   {% block scripts %}<script src="https://js.hcaptcha.com/1/api.js" async
defer></script>{% endblock scripts %}
{% endif %}

{% block title %}Login{% endblock title %}

{% block content %}
   <h2 class="text-6xl font-bold pb-10">Login</h2>
   <form method="post">
```

```
       {% csrf_token %}
       {{ form|crispy }}
       <p class="mb-3 underline"><a href="{% url 'password_reset' %}">Forgot your
password?</a></p>
       {% if site_key %}
           <div class="h-captcha mb-3" data-sitekey="{{ site_key }}"></div>
       {% endif %}
       <div>
           <button type="submit" class="py-2 bg-white border rounded-lg px-4
border-gray-300 ">Login</button>
       </div>
   </form>
{% endblock content %}
```

Code when login form submitted

accounts/views.py

```
def login_view(request):
   if not request.user.is_authenticated:
       form = AuthenticationForm()

       if settings.USE_HEROKU:
           data = {"form": form, "site_key": settings.HCAPTCHA_TOKEN}
       else:
           data = {"form": form}

       if request.method == "POST":
           if not check_hcaptcha(request):
               return redirect("accounts:login")
           username = request.POST.get("username")
           password = request.POST.get("password")

           user = authenticate(request, username=username, password=password)
           if user is not None:
               request.session["pk"] = user.pk
               user.otp.save()
               return redirect("verify/")

       return render(request, "accounts/login.html", data)
   else:
       return redirect("app:index")
```

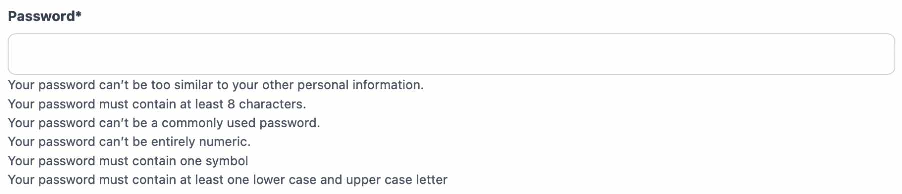
Why do you think it is secure?

- I have added a CSRF token to the form to mitigates known CSRF attacks

- I check if the captcha is passed if in the production server, explained in task 4
- I use django's authentication form which includes username and password but does not authenticate the user through the form
- I authenticate the user (not logging them in) in the view to check if the entered credentials are valid from the post request then if they are forward then to the 2fa otp stage to enter a one time password, otherwise they will just be redirected to the login page
- I use the templating system which mitigates known SQLi and XSS attacks through escaping the forms input
- Logging in happens if they pass the 2fa otp stage in task 3, but generation of the pin is within this stage

Task 3

Password policy


Password*

Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.
Your password must contain one symbol
Your password must contain at least one lower case and upper case letter

lae/password_validators.py

```
class MixedCaseValidator:
   def validate(self, password, user=None):
       if password.islower():
           raise ValidationError(_("Must contain at least one upper case letter"))
       if password.isupper():
           raise ValidationError(_("Must contain at least one lower case letter"))

   def get_help_text(self):
       return _(
           "Your password must contain at least one lower case and upper case 
letter"
       )



class SymbolValidator:
   def validate(self, password, user=None):
       if not any(not c.isalnum() for c in password):
           raise ValidationError(_("Must contain at least one symbol"))

   def get_help_text(self):
       return _("Your password must contain one symbol")
```

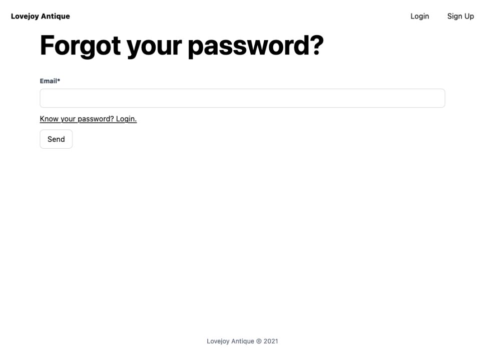
lae/settings.html

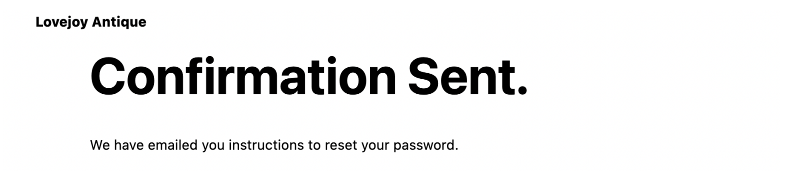
```
AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME":
"django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.CommonPasswordValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.NumericPasswordValidator",
    },
    {"NAME": "lae.password_validators.SymbolValidator"},
    {"NAME": "lae.password_validators.MixedCaseValidator"},
]
```

Explanation

- I have used four django supplied password validators and created two myself
- I have used a password validator which checks if the password inputted by the user is similar to other personal information inputted into the fom using the user attribute similarity validator
- There is a minimum length of 8 using the minimum length validator
- The common password validator checks if the password is one which is commonly used such as ‘password’ or ‘12345678’
- Also I check if the password is only numerals
- I created a validator which checks if there is at least one symbol is in the password
- Another created validator I made is to check if the password has both capitals and lower case, if a user uses only uppercase or lowercase (ignoring all non-alphabet characters) it will be invalid.

User identity management

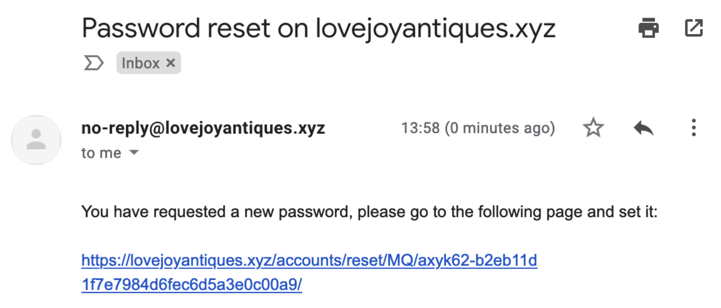




Lovejoy Antique

Confirmation Sent.

We have emailed you instructions to reset your password.



Password reset on lovejoyantiques.xyz

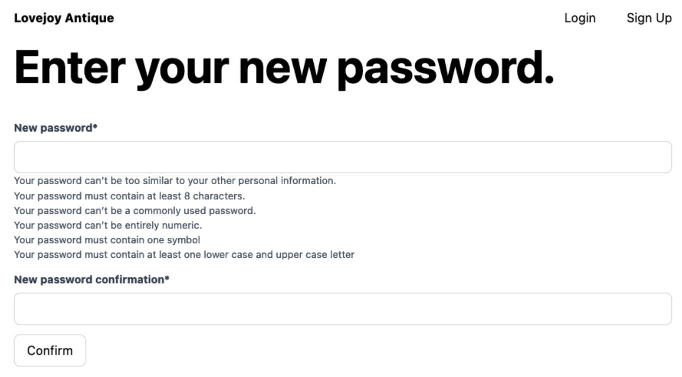
Inbox ×

no-reply@lovejoyantiques.xyz 13:58 (0 minutes ago)

to me

You have requested a new password, please go to the following page and set it:

https://lovejoyantiques.xyz/accounts/reset/MQ/axyk62-b2eb11d1f7e7984d6fec6d5a3e0c00a9/



Lovejoy Antique Login Sign Up

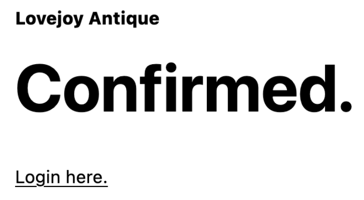
Enter your new password.

New password*

Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.
Your password must contain one symbol
Your password must contain at least one lower case and upper case letter

New password confirmation*

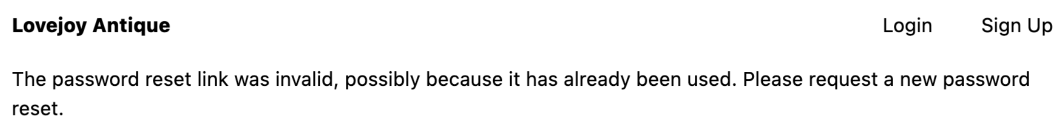
Confirm



Lovejoy Antique

Confirmed.

Login here.



Lovejoy Antique Login Sign Up

The password reset link was invalid, possibly because it has already been used. Please request a new password reset.

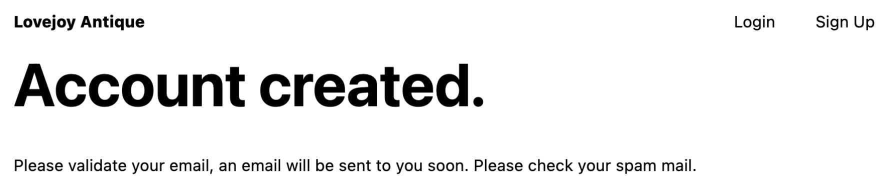
lae/settings.py

```
INSTALLED_APPS = [
   "django.contrib.admin",
   "django.contrib.auth",
   "django.contrib.contenttypes",
   "django.contrib.sessions",
   "django.contrib.messages",
   "django.contrib.staticfiles",
   "phonenumber_field",
   "crispy_forms",
   "crispy_tailwind",
   "app",
   "accounts",
   "storages",
]

MIDDLEWARE = [
   "whitenoise.middleware.WhiteNoiseMiddleware",
   "django.middleware.security.SecurityMiddleware",
   "django.contrib.sessions.middleware.SessionMiddleware",
   "django.middleware.common.CommonMiddleware",
   "django.middleware.csrf.CsrfViewMiddleware",
   "django.contrib.auth.middleware.AuthenticationMiddleware",
   "django.contrib.messages.middleware.MessageMiddleware",
   "django.middleware.clickjacking.XFrameOptionsMiddleware",
   "csp.middleware.CSPMiddleware",
]
```

Explanation

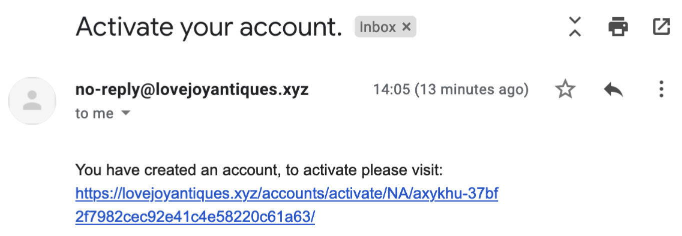
- The auth app includes password resetting and user management, the two tasks beforehand are self-programmed implementations for registration and logging in while resetting passwords is managed by Django's authentication system
- The authentication system generates a one time link with a token which is used to enable the user to reset the password and is sent to the users email address

Email verification for registration


Lovejoy Antique Login Sign Up

Account created.

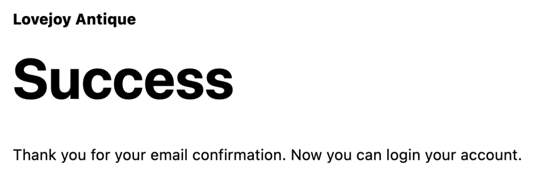
Please validate your email, an email will be sent to you soon. Please check your spam mail.


Activate your account. Inbox ×

no-referenced@lovejoyantiques.xyz 14:05 (13 minutes ago)
to me

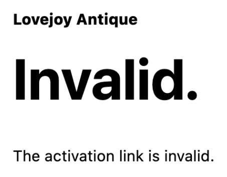
You have created an account, to activate please visit:
https://lovejoyantiques.xyz/accounts/activate/NA/axykhu-37bf2f7982cec92e41c4e58220c61a63/

(if using sussex account, it will be automatically validated, I assume this is because of the email filter visiting the page to check if malicious)


Lovejoy Antique

Success

Thank you for your email confirmation. Now you can login your account.


Lovejoy Antique

Invalid.

The activation link is invalid.

accounts/views.py

```
def validate_email(request, uidb64, token):
    return_title = "Invalid."
    return_text = "The activation link is invalid."

    try:
        user = User.objects.get(pk=urlsafe_base64_decode(uidb64))
    except (TypeError, ValueError, OverflowError, User.DoesNotExist):
        user = None

    if (
        user is not None
        and account_activation_token.check_token(user, token)
        and not user.is_active
    ):
        user.is_active = True
        user.save()
        return_title = "Success"
        return_text = (
            "Thank you for your email confirmation. Now you can login your account."
        )

    return render(
        request, "accounts/validate.html", {"title": return_title, "text":
return_text}
    )
```

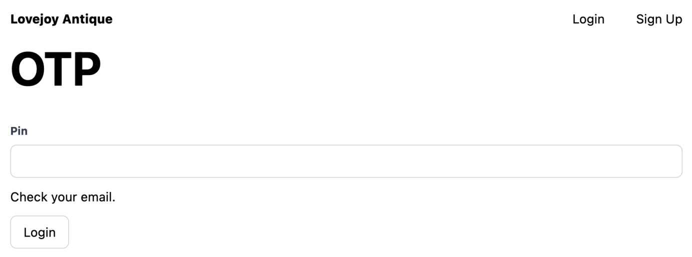
accounts/tokens.py

```
class AccountActivationTokenGenerator(PasswordResetTokenGenerator):
    def _make_hash_value(self, user, timestamp):
        return f"{user.pk}{timestamp}{user.last_login}{user.password}{user.email}"


account_activation_token = AccountActivationTokenGenerator()
```

Explanation

- The account activation inherits the same class the password reset flow uses to create the tokens, I adapt the hash value function to return a combination of the user id, the timestamp of the account activation request, the last login timestamp (so when the user logs in the hash is then invalidated it making it a one time link), password and email for more uniqueness which makes brute forcing difficult
- The account token is created in task 1, with the link being sent out, the users account is automatically deactivated and when the link is visited it sets activated to true for the account
- The url is formatted as such: accounts/uidb64/token/ where uidb64 is the user id in base64 to easily access the users primary key and then to check the hash against the user attributes to see if it is valid or not to activate the account

2 factor authentication

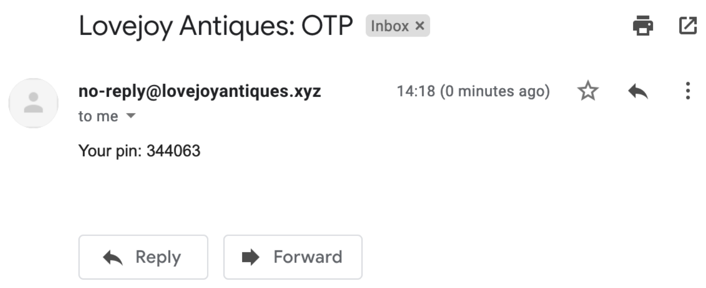

Lovejoy Antique Login Sign Up

OTP

Pin

Check your email.

Login

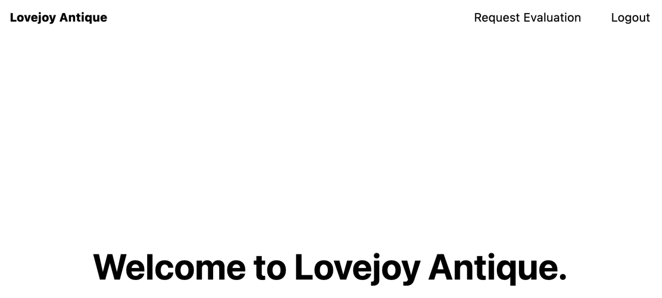

Lovejoy Antiques: OTP Inbox ×

no-reply@lovejoyantiques.xyz 14:18 (0 minutes ago)

to me

Your pin: 344063

Reply Forward


Lovejoy Antique Request Evaluation Logout

Welcome to Lovejoy Antique.

accounts/models.py
```
class OTP(models.Model):
   user = models.OneToOneField(User, on_delete=models.CASCADE)
   pin = models.CharField(max_length=6, blank=True)

   def save(self, *args, **kwargs):
       self.pin = str(randint(100000, 999999))
```

```
        super().save(*args, **kwargs)

    def __str__(self):
        return str(self.pin)

    @receiver(post_save, sender=User)
    def create_otp(sender, instance, created, **kwargs):
        if created:
            OTP.objects.create(user=instance)
```

accounts/forms.py

```
class OTPForm(forms.ModelForm):
    class Meta:
        model = OTP
        fields = ("pin",)
```

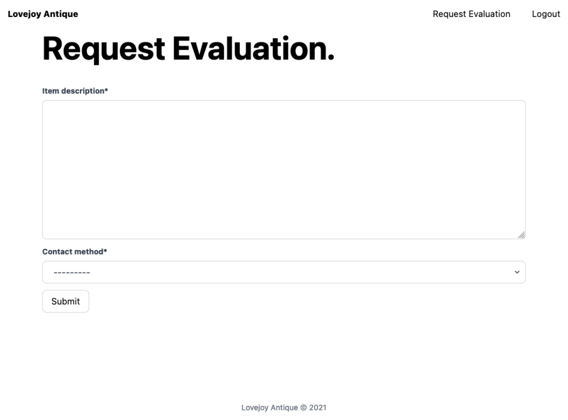
accounts/views.py

```
def verify_view(request):
    if request.session.get("pk") and not request.user.is_authenticated:
        form = OTPForm(request.POST or None)
        pk = request.session.get("pk")

        if pk:
            try:
                user = User.objects.get(pk=pk)
            except User.DoesNotExist:
                return Http404

            pin = user.otp.pin

            if not request.POST:
                mail_subject = "Lovejoy Antiques: OTP"
                message = render_to_string("accounts/otp_email.html", {"pin": pin})
                send_mail(
                    mail_subject, message, settings.DEFAULT_FROM_EMAIL, [user.email]
                )

            if form.is_valid():
                pin_input = form.cleaned_data.get("pin")
                if pin_input == pin:
                    otp.save()
                    login(request, user)
                    return redirect("app:index")

        return render(request, "accounts/otp.html", {"form": form})
    else:
        if request.user.is_authenticated:
            return redirect("app:index")
        else:
            return redirect("accounts:login")
```

Explanation

- This is the continuation of the login form code, you will be redirected to this verify stage with a session cookie which has the user primary key stored for the 2fa otp stage to continue the verification process after authenticating the username and password combination
- The OTP model randomly generates a six digit pin, which has roughly one million combinations, on save of the object which happens in the login stage
- I found a potential exploit where if the database was dumped and I pre-generate the otp they would be able to access the account easily, this was mitigated by generating in the login stage rather than after logging in for the next time
- We use a receiver to create a otp model for each user when an instance of such is made so it is compulsory
- We only send an email with the pin to the user if the request isn't a post request so
- I use templating tools which auto-escapes inputs into forms

Task 4



Request Evaluation form code

app/models.py

```
class EvaluationRequest(models.Model):
   CONTACT_METHODS = (
       ("E", _("Email")),
       ("P", _("Phone")),
   )

   user = models.ForeignKey(
       User, related_name="evaluation_requests", on_delete=models.CASCADE
   )
   item_description = models.TextField(validators=[MaxLengthValidator(5000)])
   contact_method = models.CharField(max_length=1, choices=CONTACT_METHODS)
```

app/forms.py
```
class EvaluationForm(forms.ModelForm):
   class Meta:
       model = EvaluationRequest
       fields = ("item_description", "contact_method")
```

app/templates/app/request_evaluation.html
```
{% extends 'base.html' %}
{% load static %}

{% load tailwind_filters %}

{% block title %}Request Evaluation{% endblock title %}

{% block content %}
   {% if not submitted %}
       <h2 class="text-6xl font-bold pb-10">Request Evaluation.</h2>
       <form method="post" enctype="multipart/form-data">
           {% csrf_token %}
           {{ form|crispy }}
           <div>
               <button type="submit" class="py-2 bg-white border rounded-lg px-4
border-gray-300 ">Submit</button>
           </div>
       </form>
   {% else %}
       <div class="flex h-full flex-col items-center justify-center">
           <p class="text-5xl font-bold text-center text-green-500">Submitted
request.</p>
       </div>
   {% endif %}
{% endblock content %}
```

Code when form submitted

app/views.py
```
@login_required(login_url="/accounts/login/")
def request_evaluation(request):
   submitted = False

   if request.method == "POST":
       form = EvaluationForm(request.POST)
```

```
        if form.is_valid():
            obj = form.save(commit=False)
            obj.user = User.objects.get(pk=request.user.id)
            obj.save()
            submitted = True
    else:
        form = EvaluationForm()

    return render(
        request,
        "app/request_evaluation.html",
        {"form": form, "submitted": submitted},
    )
```

Captcha for login and signup

accounts/views.py

```
def check_hcaptcha(request):
    if settings.USE_HEROKU:
        captcha_response = request.POST.get("h-captcha-response")
        data = {
            "secret": settings.HCAPTCHA_SECRET_KEY,
            "response": captcha_response,
        }
        r = requests.post(settings.HCAPTCHA_VERIFY_URL, data=data)
        result = r.json()
        return result["success"]
    else:
        return True
```

lae.settings.py

```
HEROKU_ENV = environ.Env(
    AWS_STORAGE_BUCKET_NAME=(str, ""),
    AWS_ACCESS_KEY_ID=(str, ""),
    AWS_SECRET_ACCESS_KEY=(str, ""),
    HCAPTCHA_TOKEN=(str, ""),
    HCAPTCHA_SECRET_KEY=(str, ""),
    HCAPTCHA_VERIFY_URL=(str, ""),
)
```

```
HCAPTCHA_TOKEN = HEROKU_ENV("HCAPTCHA_TOKEN")
HCAPTCHA_SECRET_KEY = HEROKU_ENV("HCAPTCHA_SECRET_KEY")
HCAPTCHA_VERIFY_URL = HEROKU_ENV("HCAPTCHA_VERIFY_URL")
```

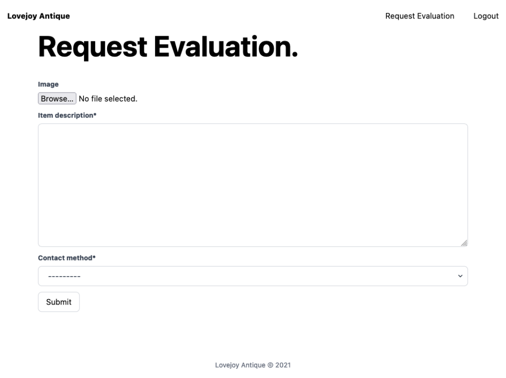
Explanation

- We insert a captcha challenge if we are running the site on a production server
- We take the request and extract the captcha response and send a post response to the verification url to see if the input was correct and then return true if passed or running locally, otherwise false
- If false, the user will be redirected to another page

Why do you think it is secure?

- I am using a model to store the results which uses a form where the inputs are automatically escaped
- I assign the requested user to the form entry once it has been validated so we can assign a users entry to them
- I use a max length of 5000 characters so a malicious user cannot try to bloat the database or try and perform something such as a buffer overflow attack if it was vulnerable to such
- The contact method is a choice of two inputs which is used to get the particular field from the user which anonymises the data if only the evaluation requests have been dumped

Task 5


Lovejoy Antique

Request Evaluation

Logout

Request Evaluation.

Image

Browse... No file selected.

Item description*

Contact method*

Submit

Lovejoy Antique © 2021

Code of the form

app/models.py

```python
def validate_image(image):
   file_size = image.file.size

   limit_mb = 5
   if file_size > limit_mb * 1024 * 1024:
       raise ValidationError("Max size of file is %s MB" % limit_mb)
```

```
class EvaluationRequest(models.Model):
   CONTACT_METHODS = (
       ("E", _("Email")),
       ("P", _("Phone")),
   )

   user = models.ForeignKey(
       User, related_name="evaluation_requests", on_delete=models.CASCADE
   )
   item_description = models.TextField(validators=[MaxLengthValidator(5000)])
   contact_method = models.CharField(max_length=1, choices=CONTACT_METHODS)
   image = models.ImageField(
       upload_to="images", null=True, blank=True, validators=[validate_image]
   )
```

app/forms.py

```
class EvaluationForm(forms.ModelForm):
   class Meta:
       model = EvaluationRequest
       fields = ("image", "item_description", "contact_method")
```

app/templates/app/request_evaluation.html

```
{% extends 'base.html' %}
{% load static %}

{% load tailwind_filters %}

{% block title %}Request Evaluation{% endblock title %}

{% block content %}
   {% if not submitted %}
       <h2 class="text-6xl font-bold pb-10">Request Evaluation.</h2>
       <form method="post" enctype="multipart/form-data">
           {% csrf_token %}
           {{ form|crispy }}
           <div>
               <button type="submit" class="py-2 bg-white border rounded-lg px-4
border-gray-300 ">Submit</button>
           </div>
       </form>
   {% else %}
       <div class="flex h-full flex-col items-center justify-center">
           <p class="text-5xl font-bold text-center text-green-500">Submitted
request.</p>
       </div>
   {% endif %}
{% endblock content %}
```

Code when the form is submitted

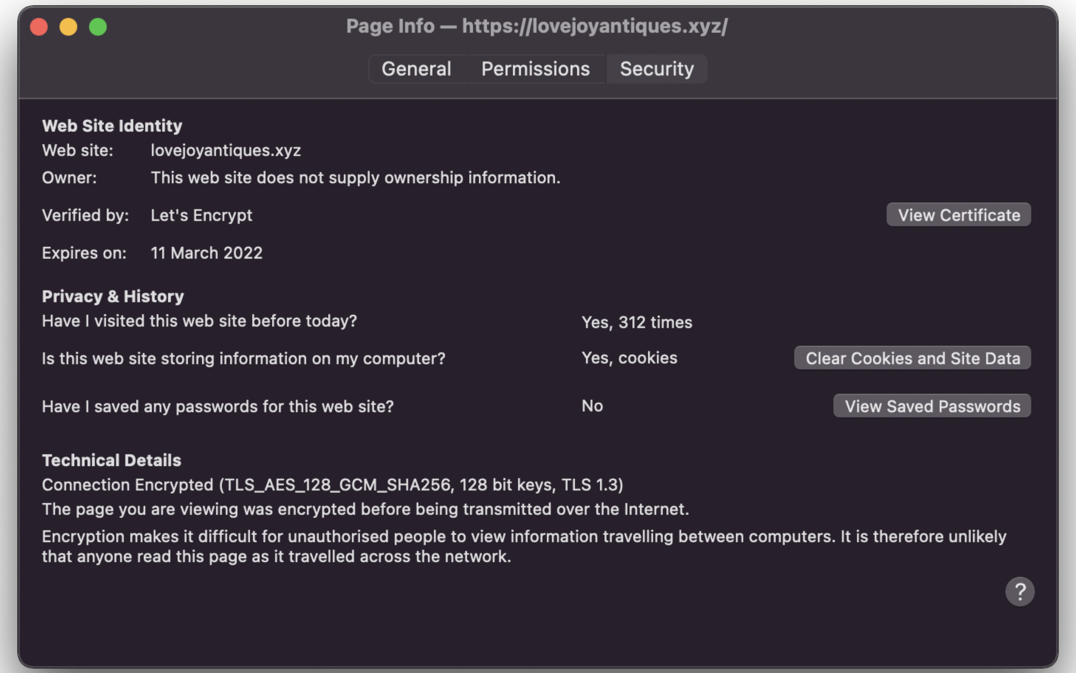
```python
@login_required(login_url="/accounts/login/")
def request_evaluation(request):
   submitted = False

   if request.method == "POST":
       form = EvaluationForm(request.POST, request.FILES)
       if form.is_valid():
           obj = form.save(commit=False)
           obj.user = User.objects.get(pk=request.user.id)
           obj.save()
           submitted = True
   else:
       form = EvaluationForm()

   return render(
       request,
       "app/request_evaluation.html",
       {"form": form, "submitted": submitted},
   )
```

lae/settings.py

```python
MEDIA_URL = "/media/"
MEDIA_ROOT = os.path.join(BASE_DIR, "media")
```

```python
if USE_HEROKU:
   DEFAULT_FILE_STORAGE = "storages.backends.s3boto3.S3Boto3Storage"
   STATICFILES_STORAGE = "whitenoise.storage.CompressedManifestStaticFilesStorage"
```

SSL


Page Info — https://lovejoyantiques.xyz/

Web Site Identity
Web site: lovejoyantiques.xyz
Owner: This web site does not supply ownership information.

Verified by: Let's Encrypt

Expires on: 11 March 2022

Privacy & History
Have I visited this web site before today? Yes, 312 times

Is this web site storing information on my computer? Yes, cookies

Have I saved any passwords for this web site? No

Technical Details
Connection Encrypted (TLS_AES_128_GCM_SHA256, 128 bit keys, TLS 1.3)
The page you are viewing was encrypted before being transmitted over the Internet.
Encryption makes it difficult for unauthorised people to view information travelling between computers. It is therefore unlikely that anyone read this page as it travelled across the network.

Explanation

- Heroku for paid servers (they call them dynos) allows you to use domain names with ssl, heroku is a cloud application platform rather than a traditional VPS so they can paywall free SSL certificates created with let's encrypt

Content Security Policy (CSP)

lae/settings.py

```
if not DEBUG:
   CSP_DEFAULT_SRC = "'none'"
   CSP_FONT_SRC = "'self'"
   CSP_STYLE_SRC = ("'self'", "https://*.hcaptcha.com", "https://hcaptcha.com")
   CSP_SCRIPT_SRC = ("'self'", "https://*.hcaptcha.com", "https://hcaptcha.com")
   CSP_IMG_SRC = ("'self'", "https://lovejoy-antique-media.s3.amazonaws.com")
   CSP_CONNECT_SRC = ("'self'", "https://*.hcaptcha.com", "https://hcaptcha.com")
   CSP_FRAME_SRC = ("'self'", "https://*.hcaptcha.com", "https://hcaptcha.com")
```

Explanation

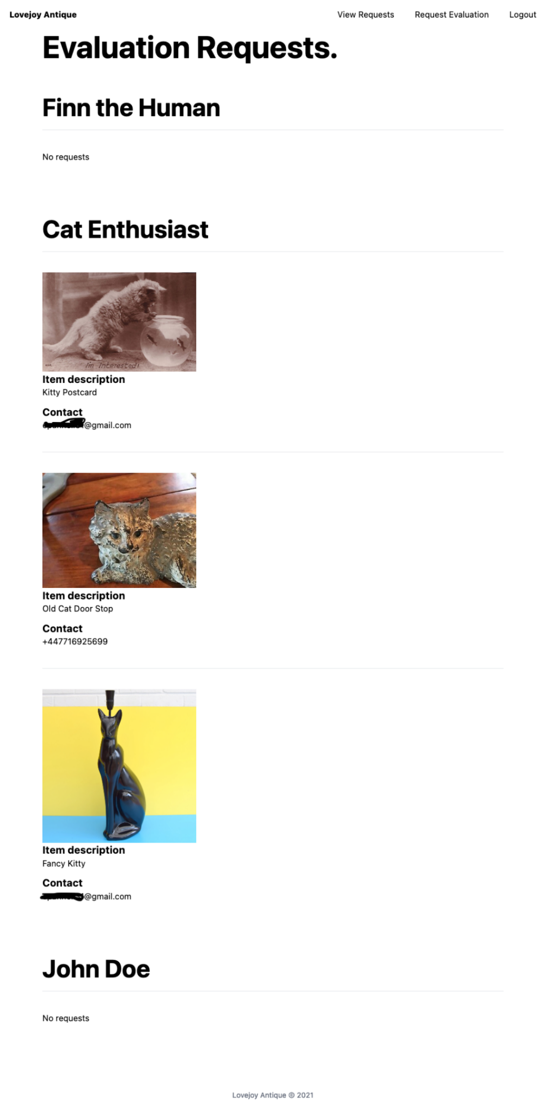
- Content security policy is another way to mitigate attacks such as XSS attacks if they were successful to a certain degree, as well as enforce https from the listed urls

- It makes an allowlist for trusted domains for particular attributes such as loading images, I have added my aws s3 bucket as an exception so I can load images off of there as well as hcaptcha which is the service i use for captcha verification

Why do you think it is secure?

- I extended the model to include an image field which only stores images with specific file extensions associated with raster file images (no svgs allowed)
- I created a validator which checks how large the file is, if it is over 5mb a warning will be displayed and included the field within the form associated with the model
- Continuing from the other tasks, I have used the templating system which auto-escapes inputs
- With aws, I use a signature for the image which makes it difficult to access the image unless you are an administrator
    - E.g., https://lovejoy-antique-media.s3.amazonaws.com/images/Antique-Kitty-Cat-1900s-Vintage-Iron-Door-Stop-_1.jpg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAXCSDOS7TGGGC27X6%2F20211220%2Feu-west-2%2Fs3%2Faws4_request&X-Amz-Date=20211220T133648Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=661da53e8ba4534e3a5a9e7492b0f4c5e870bf7bb3b2affc7ef06d4b3fabcd0f

Task 6



Lovejoy Antique   View Requests   Request Evaluation   Logout

Evaluation Requests.

Finn the Human

No requests

Cat Enthusiast

Item description
Kitty Postcard

Contact
[redacted]@gmail.com

Item description
Old Cat Door Stop

Contact
+447716925699

Item description
Fancy Kitty

Contact
[redacted]@gmail.com

John Doe

No requests

Lovejoy Antique © 2021

Code of the page

app/templates/app/admin_view_requests.html

```
{% extends 'base.html' %}
{% load static %}

{% load tailwind_filters %}

{% block title %}Evaluation Requests{% endblock title %}

{% block content %}
   <h2 class="text-6xl font-bold pb-16">Evaluation Requests.</h2>
   {% for user in user_requests %}
       <div class=" grid grid-cols-1 pb-16 divide-y divide-gray-200">
```

```
            <h3 class="text-5xl font-bold pb-5">{{ user.first_name }} {{
user.last_name }}</h2>
            {% if user.evaluation_requests.first %}
                {% for request in user.evaluation_requests.all %}
                    <div class="py-10">
                        {% if request.image %}
                            <img src="{{ request.image.url }}" class="w-1/3
object-contain"></img>
                        {% endif %}
                        <div class="pb-3">
                            <h4 class="font-bold text-xl" >Item description</h4>
                            <p>{{ request.item_description}}</p>
                        </div>

                        <div>
                            <h4 class="font-bold text-xl">Contact</h4>
                            <p>{% if request.contact_method == 'E'
%}{{user.email}}{% else %}{{user.phone_number}}{% endif %}</p>
                        </div>
                    </div>
                {% endfor %}
            {% else %}
                <div class="py-10">
                    <p> No requests </p>
                </div>
            {% endif %}
        </div>
    {% endfor %}
{% endblock content %}
```

Code for listing generation

app/views.py

```
def admin_view_requests(request):
    if (
        request.user.is_authenticated
        and request.user.is_staff
        or request.user.is_superuser
    ):
        user_requests = User.objects.prefetch_related("evaluation_requests").all()
        return render(
            request, "app/admin_view_requests.html", {"user_requests":
user_requests}
        )
    else:
        raise Http404
```

Why do you think it is secure?

- If a user was using a tool to scan directories of the web app to find specific administration tools they would be looking for redirects or other responses which were not 404 responses, so if the user is not authenticated as an administrator they will not know of the page even if scanned due to this countermeasure
- I have escaped the image tags src attribute with quotation marks so if an image is uploaded in using the form in the previous task with a malicious file name for an XSS attack or other attack vectors that bypasses the extension requirement for the image, it will not be performed.
- I use the built in tools django has supplied for templating throughout which minimises the chances of attacks taking place when visiting the page from escaping known, non-0day, exploits
- I used the burp suite community edition tool to try and find flaws however I could not find any with the current knowledge I have, and due to using sessions it would be hard to use a malicious string for the cookie value unless I knew a 0day for this as well
- Using nmap (zenmap gui) there were no open ports except port 80 and 443. On port 80 was heroku-router which is the hosting provider I am using and just redirects http to https, and port 443 was simply classified as ssl/https rather than detecting the django platform explicitly
- In task 3 I explained using the known password feature for the password validators, this is still being used