Secure Software Engineering

Agenda

Anforderungen
Tech Stack
Django Builtins
Umgesetze Funktionen
CI/CD
DSGVO
AI Usage
Live Demo
Fazit

Anforderungen

- User-Registrierung mit Verifizierungs-E-Mail
- Notes in Markdown erstellen und automatisch zu HTML rendern
- Erweiterung durch ein Social-Plugin zum Einbetten von YouTube-Videos
- Notes können öffentlich oder privat sein
- Suche nach Notes
- Aufruf von Notes über einen direkten Link
- Schnittstelle (API) für andere Anwendungen
- Korrekte Authentifizierung & Autorisierung
- DSGVO konform

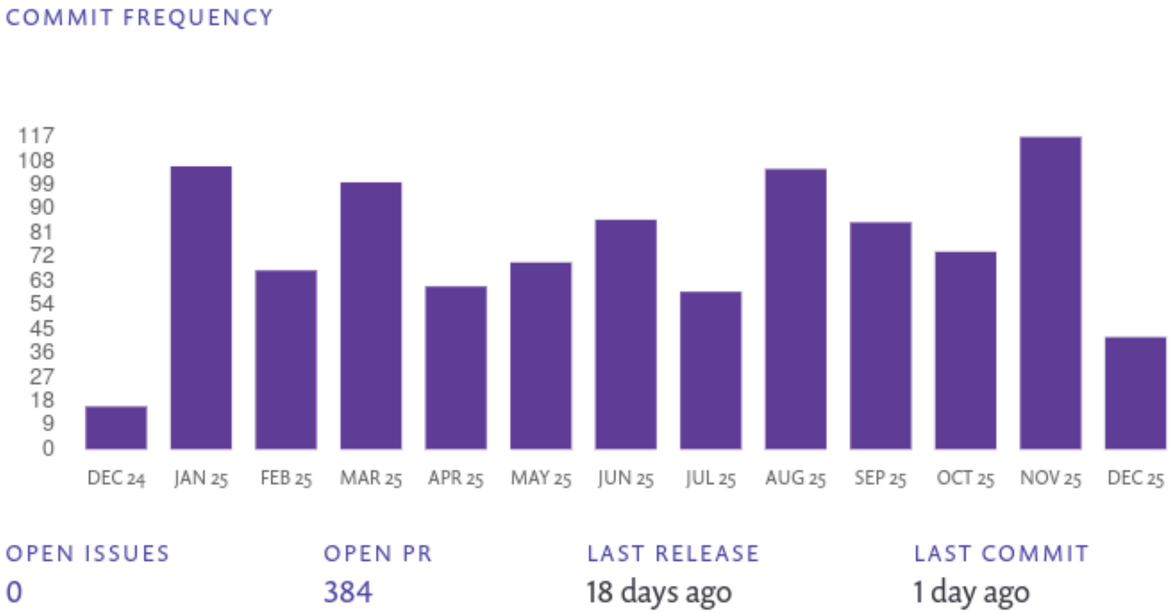
Tech stack

Backend

Frontend

Warum Django?

- Weit verbreitet
- Aktiv gewartet (Active Maintenance)
- Python-basiert
- Viele Sicherheitsmechanismen bereits integriert
- Einfach erweiterbar



Views

- Funktion, die Webanfragen verarbeitet und Antworten zurückliefert
- Beliebige Logik kann implementiert werden
- Werden einer URL zugeordnet (registriert)

```python
def preview_note(request):
    markdown = request.POST.get('markdown') or ''
    return HttpResponse(markdown2html_safe(markdown).encode())
```

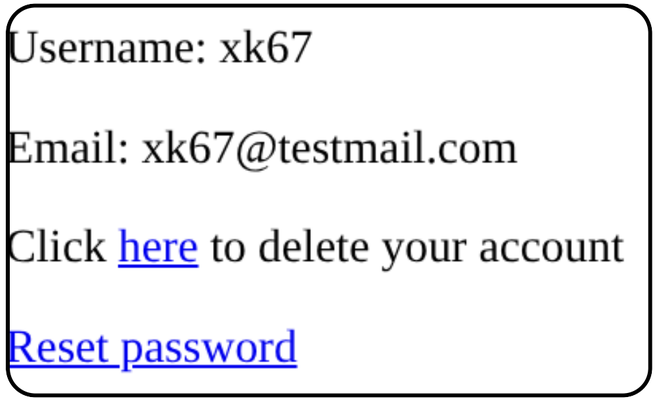
```python
urlpatterns = [
    path('note/preview', web.preview_note, name="preview_note")
]
```

Templating

- Generierung von dynamischen HTML durch die Kombination von statischem HTML mit dynamischen Daten
- Variablen werden automatisch escaped, um XSS zu verhindern
- Entwickler können Logik, Schleifen und Bedingungen in Templates nutzen

```python
def profile(request):
    return render(request, 'users/profile.html')
```

```
<p>Username: {{ user.username }}</p>
<p>Email: {{ user.email }}</p>
<p>Click <a href="{% url 'users:delete_account' %}">here</a> to delete your account</p>
<p><a href="{% url 'users:password_reset' %}">Reset password</a></p>
```



Authentifizierung

- Die Authentifizierung erfolgt über die sitzungsbasierte Authentifizierungs-Middleware von Django
- Erfolgreicher Login → erstellt serverseitige Sitzung & Cookie im Brower wird gesetzt
- **Cookie Eigenschaften**
  - session_key: random 32-character String Identifier, Value von *sessionid*
  - session_data: Serialisierte Sitzungsinhalte, die mit dem *SECRET_KEY* des Projekts signiert sind
  - expire_date: Zeitstempel, nach dem die Sitzung als ungültig betrachtet wird (Default: 14 Tage)
  - HttpOnly verhindert das auslesen durch JavaScript

Set-Cookie	sessionid=q0z9kbrv24b3tkty9vzaf84wgn6y4x65; expires=Sun, 08 Feb 2026 12:58:57 GMT; HttpOnly; Max-Age=1209600; Path=/; SameSite=Lax

```
sqlite> select * from django_session;
q0z9kbrv24b3tkty9vzaf84wgn6y4x65|.eJxVjEEOwiAQRe_C2pACA0NduvcMhGFAqoYmpV0Z765NutDtf-
_9lwhxW2vYel7CxOIslDj9bhTTI7cd8D222yzT3NZlIrkr8qBdXmfOz8vh_h3U2Ou3BgDDYL1TxoNKgOAdDsWlMiZNyoMpSAhKG9RchsieNaG1xL64MWfx_gCqpjcq:1vjzhh:
8C5kkh17IojUPUkjiFkDcDvQFRsk7j5WOAZyGLo0m_A|2026-02-08 12:58:57.528077
```

```
{'_auth_user_id': '1', '_auth_user_backend': 'django.contrib.auth.backends.ModelBackend', '_auth_user_hash':
'4443d458613841c4748670f6cf9c2b1843f7b7412372df0ad8d2b755bd8f69ee'}
```

Autorisierung

- Die AuthenticationMiddleware stellt den authentifizierten Benutzer über request.user bereitt
  - Ermöglicht Autorisierung auf Basis von Benutzer
- Zugriffsbeschränkungen auf Views werden mit @login_required umgesetzt

```python
@login_required
def view_note(request, uuid):

    note = Note.objects.get(uuid=uuid)

    if request.method == "POST":
        if note.owner != request.user:
            raise Http404()
```

CSRF Protection

- Setzt ein zufälliges CSRF-Cookie im Browser (csrftoken)
- Fügt bei POST-Requests ein verstecktes CSRF-Token (csrfmiddlewaretoken) hinzu
- CSRF-Cookie und Token müssen im Request vorhanden sein → 403 wenn nicht
  - Django prüft Gültigkeit: Token Secret wird mit Cookie Secret verglichen

```
<form method="post">
    {% csrf_token %}
    {{ form }}
    <button type="submit">Delete account</button>
</form>
```

```
<form method="post">
    <input type="hidden" name="csrfmiddlewaretoken" value="XvgPCkZ28E7IcoUzYrz4kZaI5MAiPnGz2ve52voYb9hjx9kasnxnyDz6qqTquhtc">
```

Object-Relational Mapper (ORM)

- Django ORM schützt vor SQL Injection, indem parametrisierte Abfragen für alle Datenbankoperationen verwendet werden
- Benutzereingaben werden nie direkt in SQL-Statements eingefügt
- Escaped Parameter sicher und erstellt die Abfragen korrekt

```
Note.objects.get(uuid=uuid)
```

```
SELECT * FROM notes_note WHERE uuid = %s;
```

User Registration

- Endpoint erwartet: username, email, password1, password2, csrfmiddlewaretoken
- Django Built-in Validierungen:
  - **Username**
    - Maximal 150 Zeichen
    - Erlaubte Zeichen: Buchstaben, Zahlen sowie @ . + - _
  - **Email**
    - Formatprüfung (z. B. <name>@<domain>)
  - **Password**
    - Übereinstimmung von password1 und password2
    - Mindestlänge von 15
    - Prüfung gegen häufig verwendete Passwörter (pwnedpasswords-v6-top20k.txt)
    - Prüfung, ob das Passwort nicht ausschließlich numerisch ist
    - Ähnlichkeitsprüfung zu Benutzerattributen (Username, Email)
- Passwortspeicherung
  - Speicherung als gehashter Wert in der Datenbank
  - Algorithmus: pbkdf2_sha256
  - Iterationen: 1.000.000
  - Salt: random String
- Account-Aktivierung
  - Nach erfolgreicher Registrierung wird eine Bestätigungs-E-Mail versendet
  - Der Account ist erst nach Klick auf den Aktivierungslink nutzbar
- Mögliche Schwachstellen & Gegenmaßnahmen
  - Username / Email Enumeration
    - Einheitliche Fehlermeldungen
  - Data Breach
    - Starke Passwort-Hashing + Salting
  - SQL Injection
    - Django ORM verwenden
  - Cross-Site Request Forgery
    - Django CSRF Schutz verwenden

User Login

- Endpoint erwartet: username, password, csrfmiddlewaretoken
- Django AuthenticationForm überprüft Benutzername & Passwort
  - Erfolgreicher Login → Cookie wird gesetzt
- Mögliche Schwachstellen & Gegenmaßnahmen
  - Username Enumeration
    - Einheitliche Fehlermeldungen
  - SQL Injection
    - Django ORM verwenden
  - Cross-Site Request Forgery
    - Django CSRF Schutz verwenden

```
urlpatterns = [
    path("login", LoginView.as_view(template_name="users/login.html", redirect_authenticated_user=True), name="login"),
]
```

Note Erstellung

- Endpoint erwartet: title, content, privat, csrfmiddlewaretoken
- Sanitization
  - sanitize_title(title) → safe title
  - markdown2html_safe(content) → safe content
- Mögliche Schwachstellen & Gegenmaßnahmen
  - SQL Injection
    - Django ORM verwenden
  - Cross-Site Request Forgery
    - Django CSRF Schutz verwenden

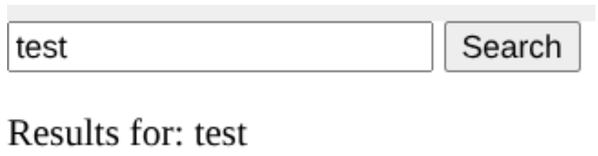
```python
class Note(models.Model):
    owner = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name="notes"
    )
    title = models.CharField(max_length=32)
    content = models.TextField()
    private = models.BooleanField(default=True)
    uuid = models.UUIDField(default=uuid.uuid4, editable=False, unique=True)
```

markdown2html_safe

- **Markdown Parser**
  - Implementiert mit der Markdown Bibliotek
  - Markdown-Parser wird mit einer Yt2iframe-Extension erweitert
  - Die Extension prüft das Pattern: *![<Optionaler Titel>](embed:<Link>)*
  - Extrahiert die Video-ID wenn:
    - http / https
    - gültiger YouTube-Link
  - Konvertiert den Link zu: https://www.youtube-nocookie.com/embed/<video_id>
  - Erstellt ein iframe und bettet es in ein Template mit Consent-Banner ein
  - Rendert Markdown zu HTML
- **Sanitizing**
  - Implementiert mit der Bleach Bibliotek
  - Entfernt unsichere Tags, Attribute und Protokolle (z. B. script, onerror, javascript:)
  - Verwendet eine vordefinierte Allowlist für erlaubte Tags und Attribute
  - Erlaubt explizit iframe src nur für youtube-nocookie.com

Note Suche

- Erwartet Query-Parameter q in der URL: /note/search?q=test
- Sucht nach Notes anhand des Titels
- Rendert Query-Paramter im HTML
- Gibt eigene Notes sowie öffentliche Notes anderer Nutzer zurück
- Mögliche Schwachstellen & Gegenmaßnahmen
  - Broken Access Control
    - Korrekte Implementierung der Autorisierung
  - Cross-Site Scripting
    - Django Templating verwenden
  - SQL Injection
    - Django ORM verwenden




```python
notes_private = Note.objects.filter(
    owner=request.user,
    title__icontains=q
)

notes_public = Note.objects.filter(
    private=False,
    title__icontains=q
```

Note ansehen

- GET-Request zu /notes/<uuid>
- Notes können unabhängig vom öffentlichkeits Status angesehen werden, aber User muss eingeloggt sein
- Um Inhalte HTML-unescaped im Template zu rendern: mark_safe(note.content)
- Mögliche Schwachstellen & Gegenmaßnahmen
  - SQL Injection
    - Django ORM verwenden
  - Cross-Site Scripting
    - Inhalt sanitizen

Reset Passwort

- Endpoint erwartet: email, csrfmiddlewaretoken
- Existiert ein Benutzer mit dieser E-Mail-Adresse, wird eine E-Mail versendet
  - Link in der E-Mail: /reset/<uidb64>/<token>
- Neues Passwort muss zweimal eingegeben werden
- Gleiche Passwort-Validatoren wie bei der Account-Erstellung & password1 == password2
- Der Token wird nach erfolgreichem Zurücksetzen oder nach 3 Tagen ungültigt

- Mögliche Schwachstellen & Gegenmaßnahmen
  - User Enumeration
    - Einheitliche Rückmeldungen
  - SQL Injection
    - Django ORM verwenden

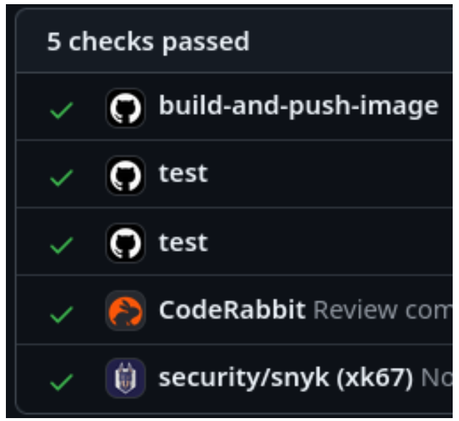
Weitere Funktionen

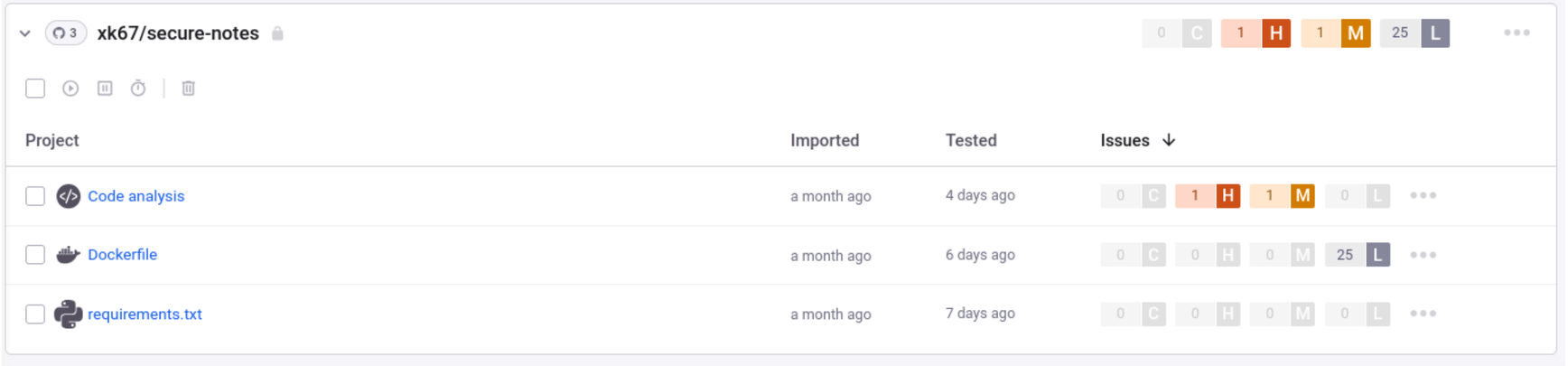
- User Logout
- User Profil
- Account Löschung
- Note Preview
- Liste der Notes
- Note Löschung
- API

https://github.com/xk67/secure-notes/tree/main/docs/docs

CI/CD

- Automatische Abhängigkeits-Updates und Sicherheitswarnungen mit GitHub Dependabot
- GitHub Actions:
  - Sicherheitsanalyse des Projekts mit Snyk
  - Code-Qualitätsanalyse mit CodeRabbit
  - Frontend-Tests mit Selenium
  - Backend-Tests mit dem Django-Test-Framework
  - Build der Anwendung als Docker-Image
  - Veröffentlichung des Images in der GitHub Container Registry bei Pushes auf main oder bei Releases





DSGVO

- Es ist vorhanden:
  - Impressum
  - DSGVO Erklärung
  - Consent-Banner

C. Categories of Personal Data

The following categories of personal data are processed in this web application:

- Account information: username, email address, password, date joined, and last login.
- User-generated content: notes created by the user.
- Technical data required for the operation of the website, such as session cookies and CSRF tokens.
- API tokens for authentication with the site's own API, including user_id and creation timestamp
- Third-party content data (YouTube videos), which is processed only after the user gives explicit consent.
- Users can embed external images in their notes. Loading external images may transmit personal data such as IP addresses to third-party servers. The website does not control these external servers and is not responsible for their data processing.

- Es können jederzeit einzelne Notes oder der gesamte Account gelöscht werden

AI Usage

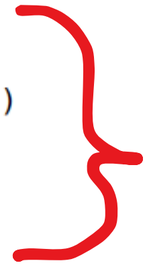
- ChatGPT genutzt, um Django-Konzepte zu verstehen
- Einzelne Funktionen über einen AI-Agenten implementiert
- Sicherheitsrelevante Aspekte in Prompts erwähnt

```python
def get_note(request, uuid):

    try:
        UUID(uuid, version=4)
    except ValueError:
        raise Http404()

    try:
        note = Note.objects.get(uuid=uuid)
    except Note.DoesNotExist:
        raise Http404("Not found.")

    serializer = NoteContentSerializer(note)
    return Response(serializer.data)
```



- Bei der Implementierung der Funktion nicht generiert
- HTTP 500 bei ungültiger UUID

Live Demo

Fazit

Was ich gelernt habe:

- Funktionalität gründlich prüfen, bevor Bibliotheken implementiert werden
- Umgang mit Git Workflow und neuen Commands
- Django praktisch anwenden
- Testen, testen, testen (jede neue Feature-Implementierung kann Bugs einführen)
- Wie komplex die DSGVO ist
- Chromium eignet sich besser als Firefox für Website-Debugging

Es hat allgemein Spaß gemacht, eine sichere Webanwendung aufzubauen (Blue-Team), dabei Sicherheitskonzepte zu vertiefen und meine Kompetenz im Pentesting auszubreiten

Danke fürs zuhören :)

Fragen?

Quellen

- https%3A%2F%2Fseeklogo.com%2Fimages%2FD%2Fdjango-logo-4C5ECF7036-seeklogo.com.png&f=1&nofb=1&ipt=dcc453da87c920847d35a2eb83a95ee73d617cdff040a57c4fb52801d1051b57
- https%3A%2F%2Fvectorseek.com%2Fwp-content%2Fuploads%2F2022%2F02%2FSQLite-Logo-Vector-730x730.jpg&f=1&nofb=1&ipt=2cb70b9b4e06ad32e635422bd84f387977f597dead747849362f3469fe37dd72
- https%3A%2F%2Fthumbs.dreamstime.com%2Fb%2Fvector-collection-web-development-shield-signs-html-css-javascript-isolated-icons-white-background-38571884.jpg&f=1&nofb=1&ipt=dc23c6ef8d1a45fb04a8a1bfc8aab60c64d33aed118abb786c788f66d9b9438b
- https://gist.github.com/roycewilliams/226886fd01572964e1431ac8afc999ce