

Inhaltsverzeichnis

1 Grundlagen	3
1.1 Einführung	3
1.2 Ziel dieses Dokuments	3
2 Problemstellung	4
2.1 Einstufung der Toxizität	4
2.2 Performance	4
3 Projektziele	5
4 Organisationsphase (Paket 0)	6
4.1 Paket Inhalt	6
4.2 Sprintplanung	7
4.3 Aufteilung der Rollen	8
4.3.1 Beschreibung der Rollen	9
4.3.2 Kommunikationsmatrix	10
4.4 Einarbeitungszeit	11
4.4.1 Einarbeitungsbereich für jeden Entwickler	11
4.4.2 Einarbeitungsbereich der einzelnen Rollen	11
4.5 Ermittlung der Tools im Projekt	12
4.5.1 Discord einrichten	13
4.5.2 Github einrichten	13
4.5.3 Power BI	14
5 Grundgerüst der Software (Paket 1)	15
5.1 Aufsetzen der Docker Container	16
5.1.1 Dockerfile für Frontend	16
5.1.2 Backend Dockerfile	17
5.1.3 Docker-compose	17
5.2 Virtuelle Maschine aufsetzen	18
6 Implementierungsphase (Paket 2)	19
6.1 Backend	19
6.1.1 Chemicals	20
6.1.2 GET	21
6.1.3 PUT	21
6.1.4 POST	22
6.1.5 Authentifizierung	23
6.2 Datenbank	24
6.2.1 Integration der Datenbank	24
6.2.2 CRUD Operations	24

6.3 Frontend	25
6.4 Machine Learning	30
6.4.1 Implementierung im Backend	30
6.4.2 Data Mining	32
7 Lokales Starten der Anwendung	38
8 Fazit	39
9 Tabellenverzeichnis	40
10 Abbildungsverzeichnis	41

1 Grundlagen - Ömer Yanar

1.1 Einführung

Als chemischen Stoff bezeichnet man ein Element oder eine Verbindung von Substanzen, die bestimmte physikalische Eigenschaften besitzen. Diese Eigenschaften können für den Menschen gesundheitsgefährdende Folgen haben. Durch die richtige Vermischung zweier oder mehrere Substanzen, entstehen neue chemische Stoffe deren physikalische Eigenschaften ohne Überprüfung nicht bestimmbar sind.

Um das zeitnahe Arbeiten mit neu gemischten chemischen Substanzen zu ermöglichen, wurde daher im Sinne dieser Projektarbeit ein Webservice erstellt, der die Toxizität der einzelnen Stoffe ermittelt.

Durch die Eingabe der Smiles Code (**Simplified Molecular Input Line Entry Specification**), einem chemischen Struktur Code, vergleicht ein Machine Learning Algorithmus Stoffe, die eine ähnliche chemische Struktur aufweisen und bestimmt anhand dessen, ob diese als toxisch oder nicht toxisch eingestuft werden.

1.2 Ziel dieses Dokuments

Diese Dokument beschreibt die grundlegende Arbeitsweise und die im Projekt verwendeten Werkzeuge. Hierbei handelt es sich um ein „lebendes“ Dokument, das im Laufe des Projektes weiter ausgebaut und detaillierter nachjustiert werden kann. Somit wird ermöglicht, dass alle Projektmitglieder sich schnell in die Grundlagen einarbeiten oder den Grund für die getroffenen Entscheidungen nachvollziehen können. Ebenfalls wird neuen Projektmitgliedern der Einstieg vereinfacht.

2 Problemstellung - Ömer Yanar

2.1 Einstufung der Toxizität

Eine der Problemstellungen in unserem Projekt ist die korrekte Einstufung der chemischen Stoffe. Während die falsche Kennzeichnung eines nicht giftigen Stoffes keine größeren Gefahren erzeugen kann, könnte die falsche Kennzeichnung eines giftigen Stoffes sehr gravierende Folgen haben. Dies muss in dieser Projektarbeit berücksichtigt und behandelt werden.

2.2 Performance

Zudem spielt die Performance des Machine Learning- und Suchalgorithmus eine große Rolle. Die Menge der Daten, die zum Trainieren des Machine Learning Algorithmus notwendig sind und auch das Vorhersagen der Substanzen auf deren Toxizität ist sehr performancelastig. Hierzu muss das richtige Modell ausgewählt werden. Da zu Beginn der Arbeit die Größe der Datensätze nicht bestimmt werden konnte, war es nicht realisierbar ein passendes Modell zu finden. In dieser Arbeit ist es möglich, das Modell passend zu den Datensätzen frei zu wählen und bei Bedarf auch zu wechseln.

3 Projektziele - Ömer Yanar

Ziel des Projektes ist es, einen Webservice zur Ermittlung der Toxizität von chemischen Stoffen mittels Machine Learning zu entwickeln. Um eine ordentliche Projektvalidierung zu gewährleisten, wurde die Projektentwicklung in drei Pakete unterteilt. Jedes dieser Pakete repräsentiert einen Meilenstein, welches den Start in das darauffolgende Paket einleitet. Die Unterteilung in Pakete erleichtern die Verfolgung des Projektfortschritts und garantiert somit eine gute Projektsteuerung.

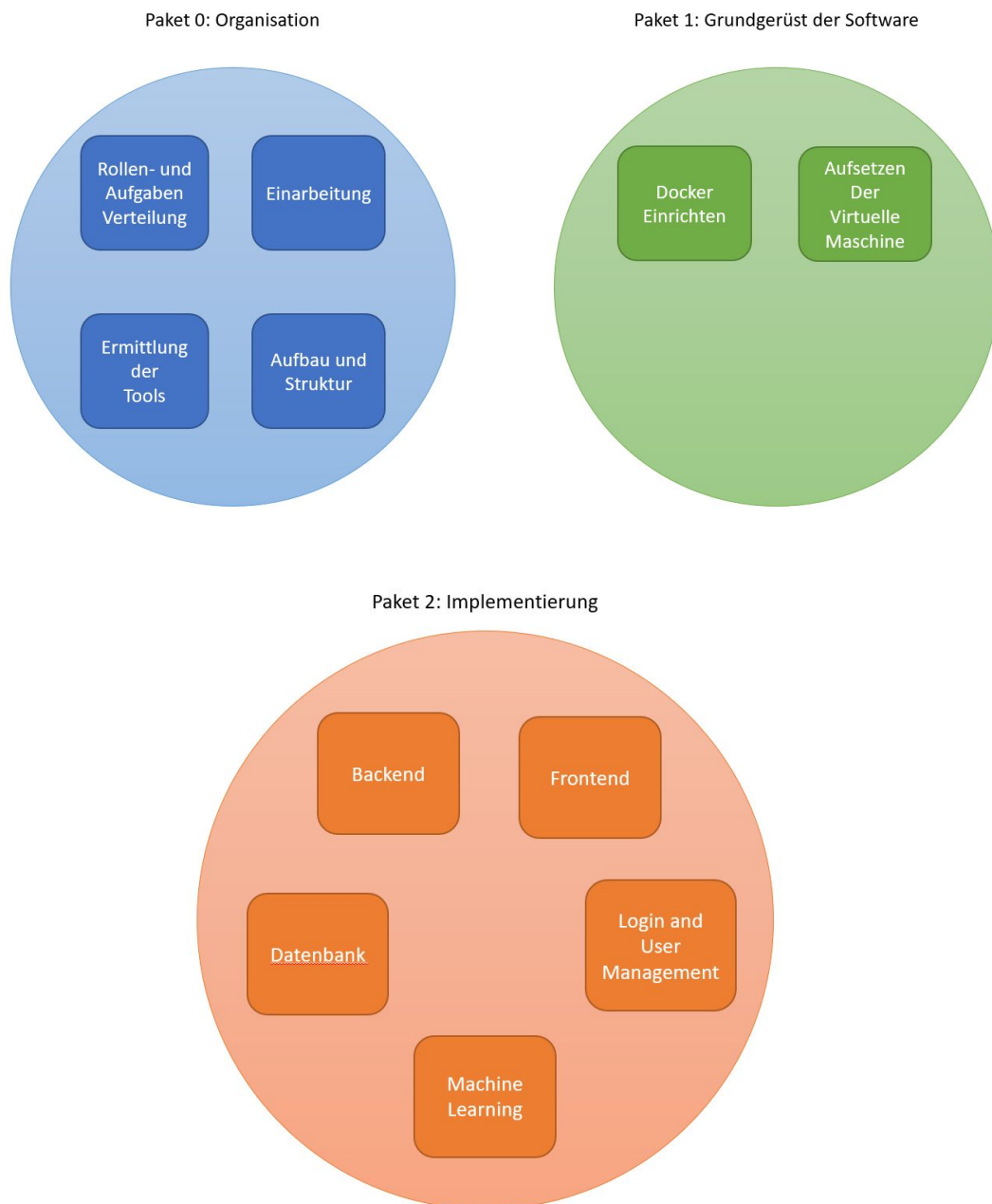


Abbildung 1: Pakete und deren Inhalt

4 Organisationsphase (Paket 0) - Ömer Yanar

Das erste Paket bezieht sich vorwiegend auf die Organisation im Projekt. Im Vordergrund stehen die vier Bereiche, Aufgaben- und Rollenverteilung, die Ermittlung der benötigten Tools im Projekt, der Aufbau der Software und die benötigte Einarbeitungszeit in die jeweiligen Bereiche.

4.1 Paket Inhalt

Paket 0:	Unterteilung	Beschreibung
Hier handelt es sich um die Einarbeitungs- und Organisationsphase für die jeweiligen Projektmitglieder.	Aufbau und Struktur der Software im Projekt.	Um eine klare Aufgabenverteilung zu garantieren, mussten zuerst die einzelnen Bausteine der Software bestimmt werden. Diese wurde in Backend, Frontend und Datenbank eingeteilt. Ebenfalls wurden die Anforderungen der Projektarbeit, wie zum Beispiel das Erstellen einer Dokumentation, berücksichtigt.
	Aufgaben und Rollenverteilung	Jedem der Teammitglieder wurde eine Rolle, mit bestimmten Aufgaben zugeteilt.
	Ermittlung der Tools im Projekt.	Für einen erfolgreichen Projektabschluss wurden bestimmte Tools in den Bereichen Kommunikation, Sourcecode Verwaltung und schriftlichem Informationsaustausch ausgewählt.
	Einarbeitung	Recherche für die Aufgaben

Tabelle 1 Aufteilung des Pakets

4.2 Sprintplanung

Die genannten vier Bereiche wurden in kleinere Tasks unterteilt. Dies vereinfachte die Planung und zudem wurde dadurch eine bessere Übersicht für jedes einzelne Teammitglied ermöglicht. Die Sprintplanung wurde jeden Sonntag aktualisiert und angepasst.

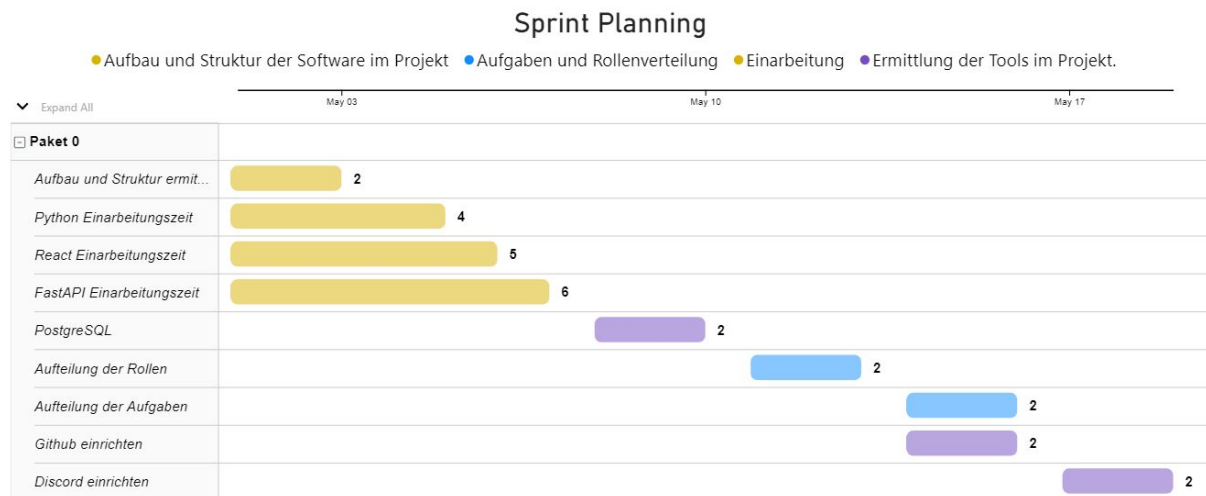


Abbildung 2: Sprintplan Paket 0

4.3 Aufteilung der Rollen

Zu Beginn des Projekts wurden die einzelnen Rollen definiert und diesen auch bestimmten Aufgaben und Pflichten zugewiesen. Jede einzelne Rolle konnte sich somit in die Aufgabengebiete einarbeiten und im Projekt wurde eine unmissverständlich Kompetenzverteilung gewährleistet. Wegen der geringen Mitgliederzahl kam es dazu, dass Projektmitglieder mehr als eine Rolle übernommen haben. Die Rollen wurden den Mitgliedern so zugeteilt, dass sie weitestgehend mit den schon gesammelten Erfahrungen korrelieren.

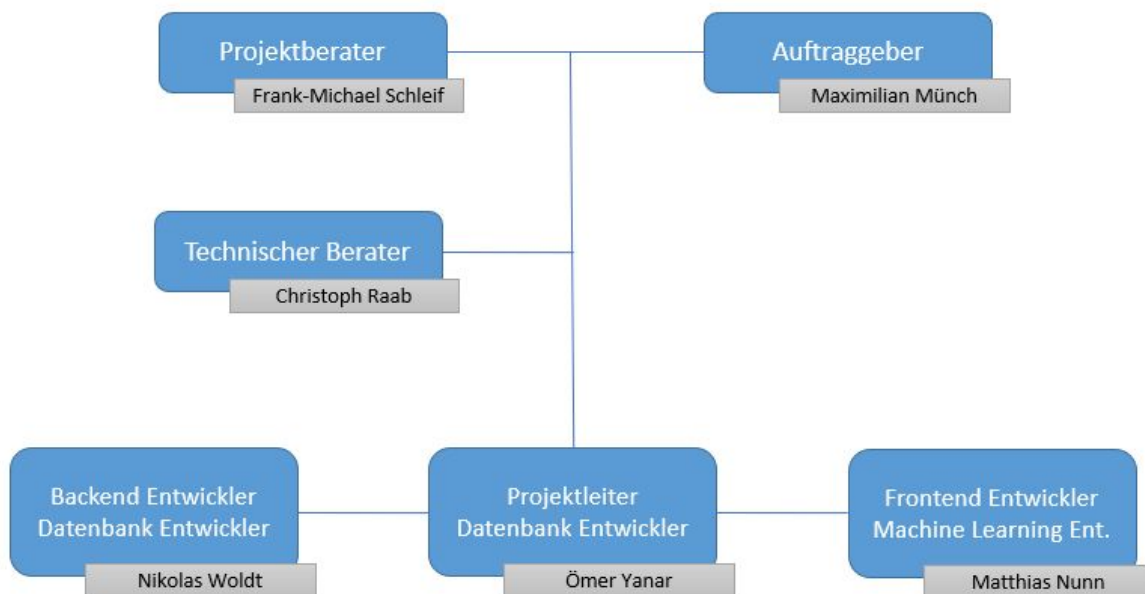


Abbildung 3: Rollenverteilung

Name	Erfahrungen
Matthias Nunn	Private Erfahrungen mit React. Kurs Intelligente Datenanalyse.
Nikolas Woldt	Erfahrungen mit React und PostgreSQL im betrieblichen Umfeld.
Ömer Yanar	Projektleiter eines IoT-Projekts, welches in Kooperation mit der FH in Schweinfurt geführt wurde.

Tabelle 2: Erfahrungen der Teammitglieder

4.3.1 Beschreibung der Rollen

Rolle	Aufgaben
Projektberater	<ul style="list-style-type: none"> • Hilfestellung beim Aufbau der Software • Bereitstellen benötigter Ressourcen
Auftraggeber	<ul style="list-style-type: none"> • Auftragserteilung • Feedback zu den Zwischenpräsentationen
Technischer Berater	<ul style="list-style-type: none"> • Unterstützung bei technischen Fragen • Hilfestellung beim Aufbau der Software
Projektleiter	<ul style="list-style-type: none"> • Aufgabenverteilung • Projektplanung • Projektziele festlegen • Priorisierung der Tasks • Teammeetings einleiten/moderieren • Bestimmung des Sprints/etc • Meilensteine festlegen • Dokumentiert Änderungswünsche des Kunden
Frontend Entwickler	<ul style="list-style-type: none"> • Darstellung der Website für den Internet Browser optimieren • Gestaltung der Seite • Bereitstellen der Funktionalität über UI
Backend Entwickler	<ul style="list-style-type: none"> • Einbindung ML Algorithmus • Auswerten der Daten • Bereitstellen der Funktionalität • Integration von Elementen und Features • Testabläufe
Datenbank Entwickler	<ul style="list-style-type: none"> • Strukturieren der Tabellen • Pflegen der Tabellen • Erweiterbare Datenbank • Suchalgorithmus
Machine Learning Entwickler	<ul style="list-style-type: none"> • Datenvorverarbeitung • Modellierung • Evaluation

Tabelle 3: Rollen und deren Beschreibung

4.3.2 Kommunikationsmatrix

Die Kommunikation gehört in der Softwareentwicklung zu den wichtigsten Aspekten. Ein Großteil der Zeit im Projekt geht durch mangelnde Kommunikation verloren. Den Mitgliedern ist nicht bewusst, an wen sie sich bei Fragen melden können. Um solch einen Zeitverlust zu vermeiden, wurde eine Kommunikationsmatrix erstellt. Eine reibungslose Kommunikation verbessert nicht nur den Informationsfluss, sondern hat auch positive Auswirkungen auf die Effizienz im Projekt. Die wöchentlichen Teammeetings sorgen für einen klaren Überblick im Projekt und haben durch den konstanten Fortschritt ebenfalls eine motivierende Wirkung.

Verantwortlicher Berichterstatter	Empfänger	Berichtsform	Inhalt	Häufigkeit
Team Tox	Projektberater, Auftraggeber, Technischer Berater	E-Mail	Fortschritt	alle 2 Wochen
Team Tox	Projektberater, Auftraggeber, Technischer Berater	mündlicher Austausch	Aktueller Stand im Projekt	bei Bedarf
Team Tox	Auftraggeber	E-Mail / mündlicher Austausch	Fragen bei Software Anpassungen	bei Bedarf
Projektleiter	Entwickler	mündlicher Austausch	Aktueller Stand im Projekt	min. wöchentlich / bei Bedarf mehrmals
Entwickler	Technischer Berater	E-Mail / mündlicher Austausch	Probleme und Fragen	bei Bedarf
Entwickler	Projektleiter	mündlicher Austausch	Anstehende Probleme in der Entwicklung	jedes mal wenn eine Verzögerung im Zeitplan besteht

Tabelle 4: Kommunikationsmatrix

4.4 Einarbeitungszeit

Nach der Aufteilung der Rollen und Aufgaben, müssen sich die Entwickler in die Themen einarbeiten. Diese Einarbeitungszeit wurde in zwei Bereiche aufgeteilt. Der erste Bereich betrifft jeden einzelnen Entwickler. Hier findet die Einarbeitung in die fundamentalen Bestandteile, die über das ganze Projekt hinweg nötig sind, statt. Der zweite ist spezifisch den Rollen zugeordnet. Für letzteres wurde ein 3-Stufen-Modell genutzt.

3-Stufen-Modell:

- Stufe 1: Jeder arbeitet sich durch Guides und Tutorials in das Themengebiet ein, bis ein bestimmtes Maß an Grundlagen beherrscht wird.
- Stufe 2: Die in Stufe 1 erlangten Kenntnisse werden dazu genutzt, um grobe Tasks zu definieren, die für das Projekt relevant sind. Diese Definition beinhaltet die Funktion des Tasks, welche Voraussetzungen für diesen Task erfüllt werden müssen und welche weiteren Entwicklungsmöglichkeiten dadurch begünstigt werden.
- Stufe 3: In der letzten Stufe, präsentieren die einzelnen Mitglieder ihre Tasks mit all deren Abhängigkeiten. Dieser Informationsaustausch verdeutlicht weiterhin die Vernetzungen der einzelnen Aufgaben untereinander. Den Mitglieder ist nun ersichtlich, in welcher Abhängigkeit die einzelnen Tasks stehen. Dies ist für die weitere Sprintplanung von großem Nutzen.

4.4.1 Einarbeitungsbereich für jeden Entwickler

Aufgrund der Bibliotheken RDKit und scikit-learn, die die Auswertung von chemischen Stoffen und das anschließende maschinelle Lernen abdecken, wurde Python als Programmiersprache ausgewählt. Diese Sprache muss jeder Entwickler beherrschen, damit diese bei taskübergreifenden Aufgaben, Verständnis für die Implementierung des jeweilig anderen Entwickler aufzeigen können und somit die Implementierung von Schnittstellen vereinfacht wird.

4.4.2 Einarbeitungsbereich der einzelnen Rollen

Hierbei handelt es sich um zeitintensive Themenbereiche, die auf die einzelnen Rollen verteilt wurden. Durch die Einarbeitung wird nochmals ein genaueres Verständnis für die zukünftigen Task gewährleistet. Die einzelnen Mitglieder können den Arbeitsaufwand der Tasks ungefähr einschätzen, welche dann in die Projektplanung eingetragen werden können. Ebenfalls wird hier deutlich, ob der Entwickler dem Arbeitsaufwand gerecht wird oder Unterstützung braucht.

4.5 Ermittlung der Tools im Projekt

Nach der Einarbeitung mussten am Ende nur noch die Werkzeuge, die im Rahmen der Projektarbeit genutzt werden, dokumentiert werden. Die zwei wichtigsten Werkzeuge hierbei sind Discord und PowerBI.

Name	Zweck	Icon
Docker	Software zur Isolierung und zur einfachen Bereitstellung von Applikationen, da benötigte Pakete bereits vorhanden sind.	
Conda	Paketmanager für Python. Bibliotheken können dabei in jeder Programmiersprache geschrieben sein.	
Discord	Kommunikations- und Informationsaustausch unter den Teammitgliedern.	
Google Docs	Dokumentierung des Berichts.	
PowerPoint	Präsentation und Grafiken.	
Zoom	Kommunikationsaustausch für das gesamte Team	
Power BI	Erstellen der Gantt Diagramme.	
Ubuntu 18.04	Virtuelle Maschine	

Tabelle 5: Tools die im Projekt genutzt werden

4.5.1 Discord einrichten

Discord wird als Kommunikations- und Informationsquelle unter den Entwicklern genutzt. Hier werden die wöchentlichen Teammeetings abgehalten, während in den einzelnen Textkanälen der Fortschritt der Software und relevante Guides festgehalten werden. Diese Notizen werden vom Projektleiter in das Product Backlog aufgenommen und für die Weiterverarbeitung der Gantt-Diagramme genutzt. Diese übersichtliche Sammlung an Informationen war für das Team, das nur aus Studierenden besteht, eine große Erleichterung in der Entwicklung. Durch die unterschiedlichen Vorlesungs- und Arbeitszeiten, war es der Gruppe meist nicht möglich ein Treffen unterhalb der Woche zu organisieren. Mit Discord als Informationsverwaltungstool, hatte das Team jederzeit die Option, den aktuellen Stand im Projekt zu überprüfen und um somit unnötige Wartezeiten zu vermindern.

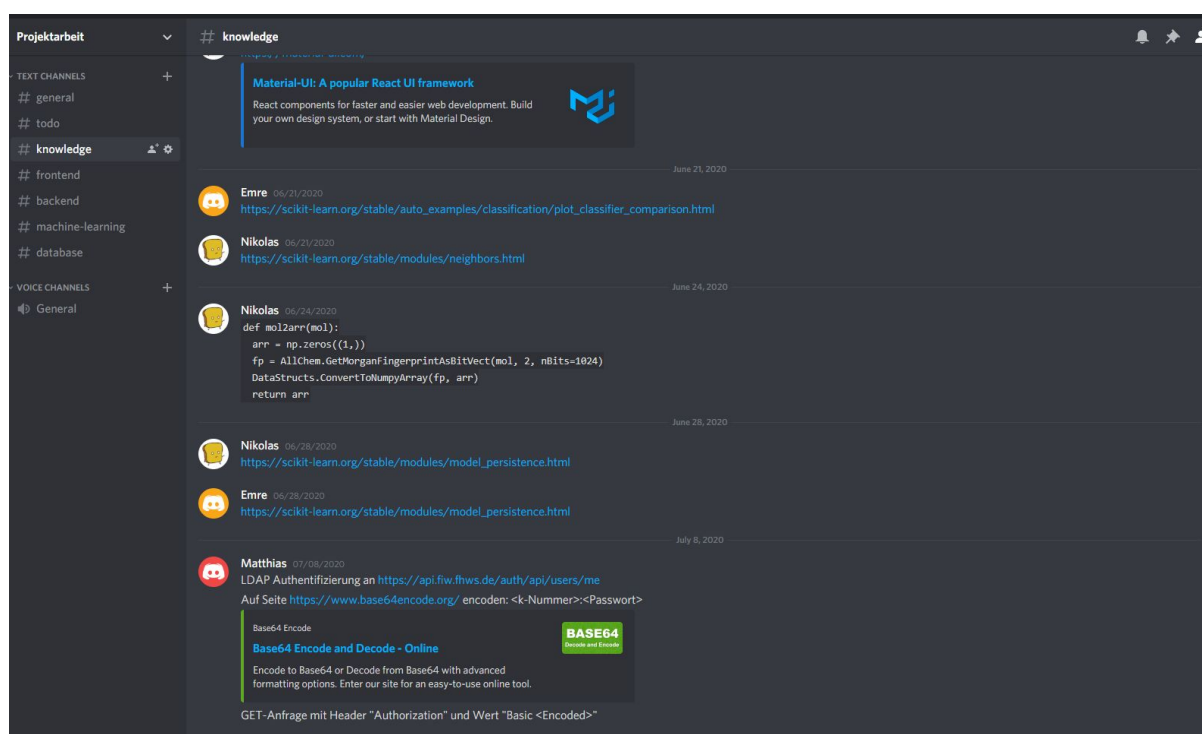


Abbildung 4: Informationsaustausch mit Discord

4.5.2 Github einrichten - Nikolas Woldt

Zur Versionsverwaltung wird Git mit einem Repository auf Github verwendet. Git ist das meist verwendete Tool für die verteilte Versionsverwaltung und Github ermöglicht die Erstellung eines kostenlosen Repository.

4.5.3 Power BI

Das Microsoft Produkt Power BI wurde für die Erstellung der Gantt Diagramme genutzt und verhalf dem Team zu einer besseren Übersicht innerhalb des Projekts. Das Product Backlog, welches in einer Excel Datei gespeichert wurde, wird ins Power BI übernommen und gespeichert. Diese übersichtliche Darstellung verhalf bei der weiteren Planung.

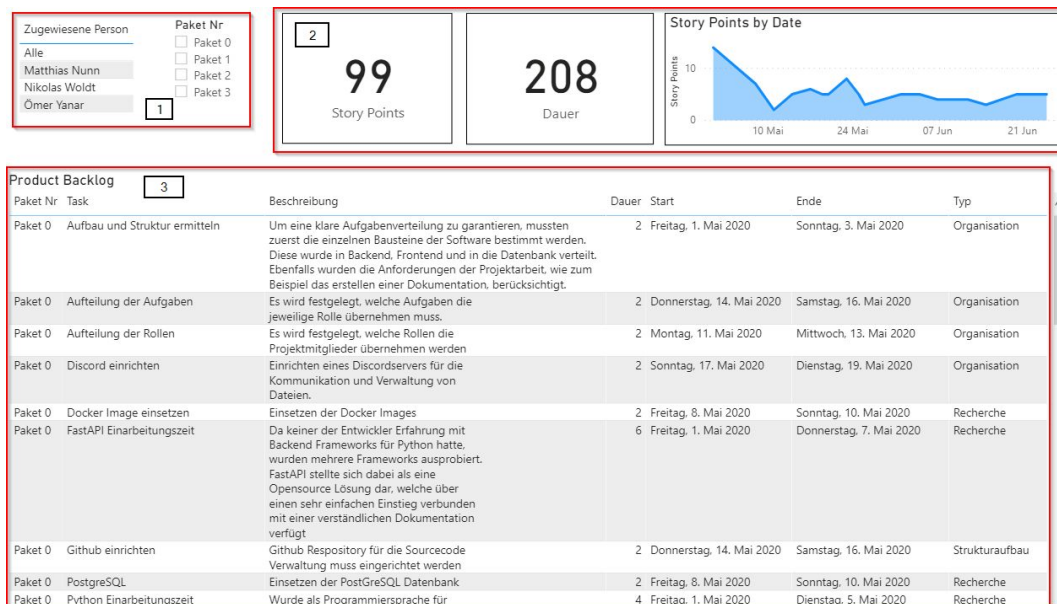


Abbildung 5: Power BI Report der Tasks

Grafik 1: Zeigt die Personen an. Hier kann man die Tabelle mit den Tasks auf die zugewiesene Person und auf das entsprechende Paket filtern.

Grafik 2: Zeigt eine Übersicht der gesamten Story-Points und die Dauer in Tagen an. Um eine faire Verteilung der Tasks zu gewährleisten, wurde ein Diagramm erzeugt. Dieses zeigt an, wie die User Story zu Zeit Verteilung im gesamten Projekt aussehen wird. Hier wurde sich stets um eine einheitliche Verteilung bemüht. Während anfangs Probleme mit der Einteilung entstanden sind, wurde gegen Ende eine gute Verteilung gewährleistet.

Grafik 3: Zeigt die gesamten Tasks im Projekt an. Diese Tabelle passt sich der dementsprechenden Filterung der Grafik 1 an.

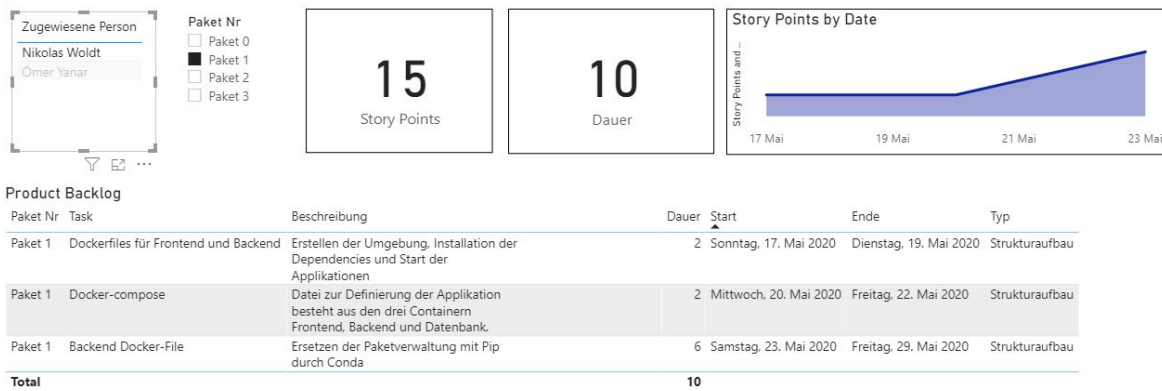


Abbildung 6: Gefilterte Darstellung des Power BI Reports

5 Grundgerüst der Software (Paket 1) - Nikolas Woldt

Paket 1:	Unterteilung	Beschreibung
Grundgerüst der Software wurde aufgebaut.	Aufsetzen der Docker Container	Frontend, Backend und Datenbank laufen jeweils in ihren eigenen Docker Container.
	Virtuelle Maschine aufsetzen	Eine virtuelle Maschine der FHWS wurde zum Testen eingerichtet

Tabelle 6: Paket 1 Inhalt - Ömer Yanar

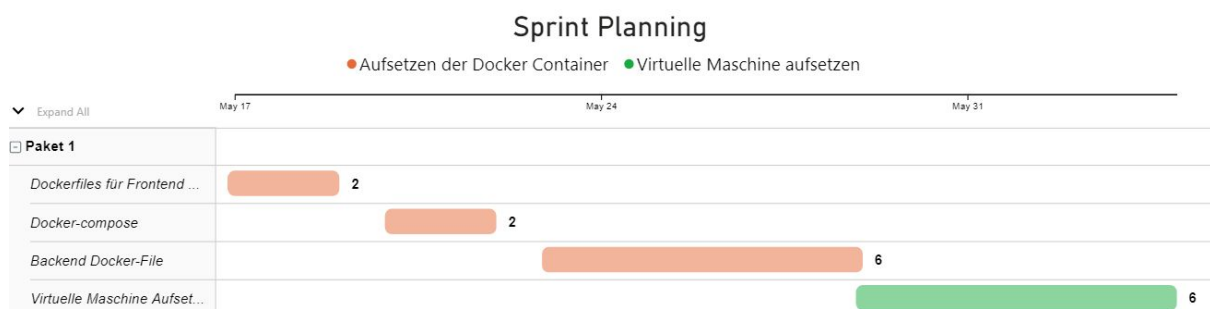


Abbildung 7: Sprintplanung Paket 1- Ömer Yanar

5.1 Aufsetzen der Docker Container

Zur lokalen Entwicklung, Deployment und Anwendungsisolierung werden Docker Container verwendet. Die Services Frontend, Backend und Datenbank existieren dabei je in ihren eigenen Containern. Über die *docker-compose* Datei wird die Struktur der Multicontainer-Applikation definiert. Das Frontend und das Backend benötigen weiterhin eigene *Dockerfiles*, da ihre Laufzeitumgebungen zusätzliche Abhängigkeiten benötigen, die beim Erstellen der Container installiert werden müssen.

5.1.1 Dockerfile für Frontend

Da für das Frontend die JavaScript Bibliothek React verwendet wird, bietet sich *Node.js* Laufzeitumgebung an.

Die Abhängigkeiten der Applikation sind in der *package.json* zu finden und werden mit *npm install* installiert.

```
FROM node:latest

ADD package.json /package.json

ENV NODE_PATH=/node_modules
ENV PATH=$PATH:/node_modules/.bin
RUN npm install

WORKDIR /app
ADD . /app

EXPOSE 8000
EXPOSE 35729

ENTRYPOINT ["/bin/bash", "/app/run.sh"]
CMD ["start"]
```

Abbildung 8: Frontend Dockerfile

5.1.2 Backend Dockerfile

Für das Backend ist eine klassische Laufzeitumgebung und Paketverwaltung mit *pip* nicht möglich. Da die Applikation RDKit zum Einlesen von SMILES Codes benutzt und RDKit keine reine Python Bibliothek, sondern primär in C++ implementiert ist, musste auf die Paketverwaltung mit Conda gewechselt werden. Mit Conda lassen sich zusätzlich zu den gängigen Python Bibliotheken auch solche in Sprachen wie C++, Java, R oder Ruby nutzen. Die Abhängigkeiten für eine Conda Applikation befinden sich in der *environment.yml*, welche auch lokal genutzt werden kann, um die entsprechende Entwicklungsumgebung für das Backend zu erzeugen.

```
FROM continuumio/miniconda3

RUN mkdir /app
WORKDIR /app

COPY environment.yml .
RUN conda env create -f environment.yml

SHELL ["conda", "run", "-n", "toxenv", "/bin/bash", "-c"]

COPY . .

ENTRYPOINT ["conda", "run", "-n", "toxenv"]
```

Abbildung 9: Backend Dockerfile

5.1.3 Docker-compose

Mit *docker-compose* können Applikationen definiert werden, welche aus mehreren Docker Containern bestehen.

In der YAML Konfiguration werden alle die einzelne Container beziehungsweise Services beschrieben: Welche Ports benutzt werden, Umgebungsvariablen, ob der Container von einem bereits existierenden Image erstellt werden soll oder ein Dockerfile benutzt werden soll.

Der Datenbank Container zum Beispiel verfügt über kein eigenes Dockerfile, da hier das Image von [Docker Hub](#) ausreicht.

5.2 Virtuelle Maschine aufsetzen

Um dem Kunden und den Beratern die Möglichkeit zu geben die Applikation selbst auszuprobieren, wurde eine virtuelle Maschine bei der Hochschule beantragt. Dabei handelt es sich um eine Linux VM mit Ubuntu 18.04, welche auch außerhalb des FHWS Netzwerks angesteuert werden kann.

6 Implementierungsphase (Paket 2)

Paket 2:	Unterteilung	Beschreibung
Implementierung	Frontend	Erstellung und Anpassungen der UI.
	Backend	Erstellung von Controllern und die Bereitstellung der Funktionalität.
	Datenbank	Erstellen der CRUD Operationen und das Einsetzen der Datenbank ins Backend.
	Machine Learning	Der Machine Learning Algorithmus wird mit den Daten trainiert. Erste Vorhersagen werden getroffen.

Tabelle 7: Aufteilung des zweiten Pakets - Ömer Yanar

6.1 Backend - Nikolas Woldt

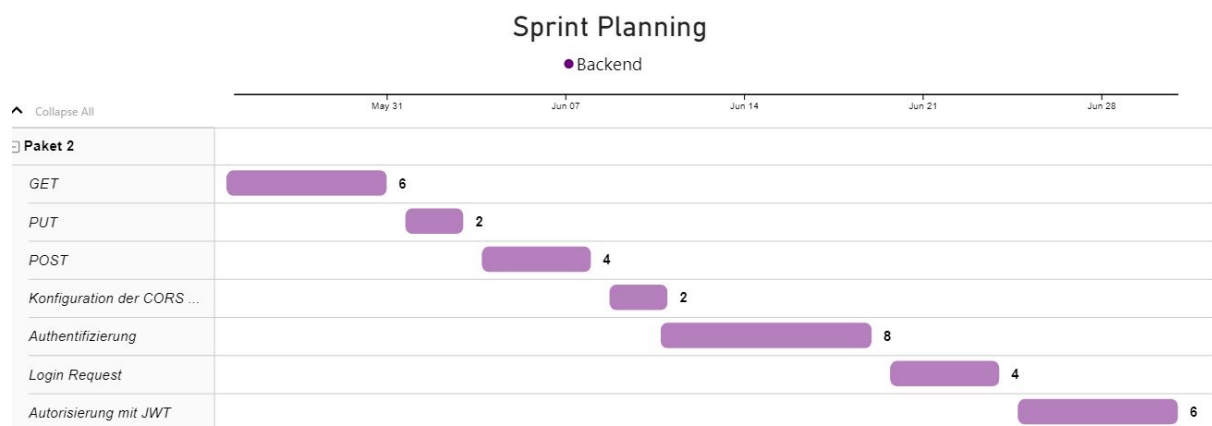


Abbildung 10: Sprintplanung des Backends - Ömer Yanar

6.1.1 Chemicals

Jede Chemikalie wird von der Applikation in die Datenbank in der *chemicals* Tabelle gespeichert. Aus der *SQLAlchemy* Definition für diese Tabelle sind die entsprechenden Felder gut ersichtlich. Die *id* ist der primäre Schlüssel der Datenbank, *smiles* ist der chemische Strukturcode und bei *code* handelt es sich um einen wissenschaftlichen Bezeichner.

Label ist ein Integer Wert, wobei 1 bedeutet, dass die Chemikalie toxisch ist und 0, dass sie dies nicht ist. Das *predicted* Feld gibt im Falle von *True* an, dass der *label*-Wert durch ein Machine Learning Modell erstellt wurde und daher die Korrektheit nicht garantiert werden kann.

```
class Chemical(Base):
    __tablename__ = "chemicals"

    id = Column(Integer, primary_key=True, index=True)
    smiles = Column(String)
    code = Column(String, unique=True)
    label = Column(Integer)
    predicted = Column(Boolean)
```

Abbildung 11: Chemical SQLAlchemy Definition

Unter */chemicals* finden sich die Endpunkte um mit der Datenbank und dem Machine Learning Modell zu interagieren. Für die Nutzung muss der User sich erst mit einem JWT der FHWS authentifizieren. Diesen erhält er durch ein Login mit seinen FHWS Credentials an dem Login Endpunkt.

Ist der User ein Student der FHWS, hat er Zugriff auf alle vorhandenen Chemikalien oder kann neue Chemikalien durch die Applikation klassifizieren. Um eine Chemikalie upzudaten, zum Beispiel wenn die Toxizität falsch vorhergesagt wurde, werden höhere Rechte benötigt.

6.1.2 GET

```
@router.get("/", response_model=List[Chemical])
async def get_chemicals(
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db),
    skip: int = Query(0, ge=0),
    limit: int = Query(10, ge=1, le=100)
):
    return chemicals_crud.get_chemicals(db, skip, limit)
```

Abbildung 12: GET Endpunkt zum Laden mehrerer Chemikalien

HTTP Endpunkte zum Abfragen von Chemikalien. Es können entweder spezifische Chemikalien gesucht oder eine ganze Liste an Chemikalien geladen werden. Im Falle von *get_chemicals* existiert dazu serverseitiges Paging, damit die Performance des Frontends nicht beeinträchtigt wird, wenn tausende von Chemikalien gleichzeitig geladen werden müssen. Dabei ist die zu ladende Anzahl an Chemikalien durch den Parameter *limit* auf maximal 100 Objekte beschränkt.

6.1.3 PUT

Wenn der Nutzer über einen POST Request die Toxizität einer bestimmten Chemikalien vorhersagen lassen hat, wurde diese in der Datenbank abgespeichert. Mit einem PUT Request kann nun die Toxizität korrigiert werden, falls diese nach einer wissenschaftlichen Überprüfung nicht der Vorhersage durch das Machine Learning Modell entspricht.

Diese Funktionalität ist aber nur für FHWS Mitglieder möglich, die kein Student sind.

Falls ein User mit der Rolle *student* versucht, das *label* zu ändern, wird eine HTTP Response mit dem Statuscode 403 zurückgegeben.

```

@router.put("/smiles/{smiles}", response_model=PredictionAnswer)
async def predict_chemical_toxicity(
    smiles: str,
    label: int = Query(..., ge=0, le=1),
    code: Optional[str] = None,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db),
):
    if current_user.role == "student":
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Invalid permissions.",
        )
    chemicals_crud.update_chemical(db, smiles, label, code)

    chem_db = chemicals_crud.get_chemical_by_smiles(db, smiles)
    return PredictionAnswer(chemical=chem_db, new=False)

```

Abbildung 13: PUT Endpunkt zum Updaten von Chemikalien

6.1.4 POST

Um die Toxizität einer Chemikalie zu bestimmen, kann ein HTTP Post Request an den Endpoint geschickt werden. Dazu wird ein SMILES Code benötigt, der die chemische Struktur der Chemikalie beschreibt.

Als erster Schritt wird überprüft, ob die Chemikalie bereits in der Datenbank vorhanden ist. In diesem Fall muss kein neuer Wert durch ein Machine Learning Modell bestimmt werden, da das *label* bereits einen Wert hat. Tritt dieser Fall ein, wird dies in der HTTP Response über das Feld *new* entsprechend gekennzeichnet.

Falls der SMILES Code noch nicht in der Datenbank vorhanden ist, wird eine neue Vorhersage durch das gewählte Machine Learning Modell erstellt. Dieses im Vorhinein trainierte Modell kann dann eine Aussage treffen, ob der Stoff eher toxisch oder nicht-toxisch ist und das Ergebnis wird unter dem *label* Feld gespeichert. Die Chemikalie wird schließlich mit dem *predicted* Attribut auf *True* in der Datenbank gespeichert, da es sich hier nur um eine Vorhersage handelt und das Modell auch falsch liegen kann. Also muss für den Nutzer gekennzeichnet werden, ob er sich auf das Ergebnis verlassen kann.

```

@router.post("/smiles/{smiles}", response_model=PredictionAnswer)
async def predict_chemical_toxicity(
    smiles: str,
    model: MLModel,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db),
):
    chemical = chemicals_crud.get_chemical_by_smiles(db, smiles)

    if chemical is not None:
        return PredictionAnswer(chemical=chemical, new=False)

    label = predict_with_model(smiles, model)

    chem = ChemicalCreate(smiles=smiles, predicted=True, label=label)
    chem_db = chemicals_crud.create_chemical(db, chem)

    return PredictionAnswer(chemical=chem_db, new=True)

```

Abbildung 14: POST Endpunkt für neue Chemikalien

6.1.5 Authentifizierung (beinhaltet Login, JWT)

Die Applikation betreibt keine komplett eigene Nutzerverwaltung. Der Login funktioniert über den LDAP Server der FHWS. Auf diese Weise können sich alle Nutzer der FHWS an dem Webservice anmelden.

Dazu schickt das Frontend die FHWS Credentials an das Backend, welches diese an die FHWS API weiterleitet. Bei erfolgreicher Authentifizierung enthält die HTTP Response diverse nicht-vertrauliche Daten. Von diesen werden nur die E-Mail und Rolle temporär gespeichert und der JWT wird an das Frontend gesendet. Bei weiteren Requests muss der JWT zur Authentifikation genutzt werden und wird jedes Mal mit der FHWS verifiziert. Auf diese Weise muss das Backend keine spezifische Session Informationen speichern, da alle relevanten Daten bereits im JWT enthalten sind und ist somit *stateless*.

Soll nun ein Endpunkt abgesichert werden, muss der Methode ein *current_user: User = Depends(get_current_user)* hinzugefügt werden, wodurch bei jedem Aufruf die Methode *get_current_user* ausgeführt wird. Diese führt dann die Authentifizierung mit der FHWS aus.

6.2 Datenbank - Nikolas Woldt

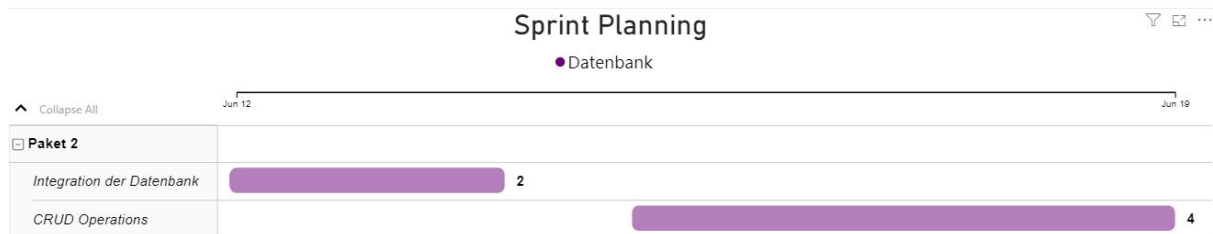


Abbildung 15: Sprintplanung für die Datenbank - Ömer Yanar

6.2.1 Integration der Datenbank

Um mit der Datenbank in einer Python Applikation zu interagieren wird das ORM-Framework SQLAlchemy verwendet. Zur Erstellung der *Engine*, welche die eigentliche Datenbank darstellt, wird nur die Datenbank URI benötigt, welche *sich in der Laufzeitumgebung befindet*.

SessionLocal kann durch die `get_db()` Methode in jedem Endpoint benutzt werden, falls Datenbank Interaktion notwendig ist. Dazu muss nur `Depends(get_db)` den Methoden Parametern hinzugefügt werden, da FastAPI alle Abhängigkeiten mit `Depends()` automatisch via Dependency Injection erfüllt.

```
engine = create_engine(
    config.SQLALCHEMY_DATABASE_URI,
)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Abbildung 16: Datenbank Integration

6.2.2 CRUD Operations

Zum speichern und laden von Chemikalien, werden eine Reihe von Methoden zur Verfügung gestellt.

Dazu gehören Create, Read und Update Operationen in mehreren Varianten. Löschen ist nicht möglich, da dies den derzeitigen Anforderungen entspricht. Jede Methode enthält die im vorhergegangenen Abschnitt erwähnte *Session* als Parameter. Mit dieser wird dann die entsprechende Operation ausgeführt.

```
def get_by_smiles_query(db: Session, smiles: str):
    return db.query(Chemical).filter(Chemical.smiles == smiles)
```

Abbildung 17: Beispiel für eine Datenbankabfrage

6.3 Frontend - Matthias Nunn

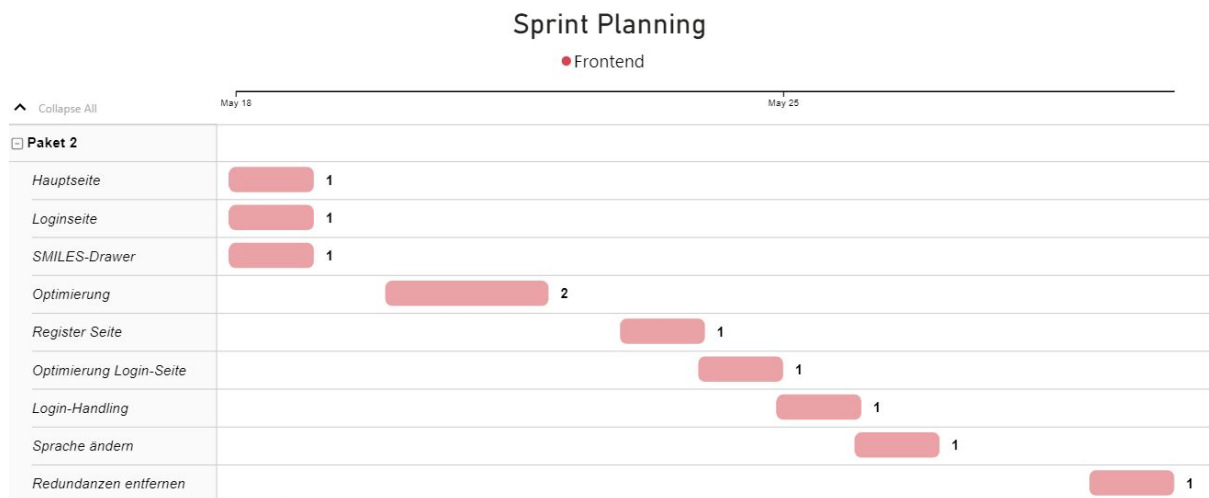


Abbildung 18: Sprint Frontend - Ömer Yanar

Im folgenden wird statt den einzelnen Tasks, das Gesamtergebnis beschrieben.

Um nur autorisierten Personen den Zugriff auf die Datenbank zu gewähren, erfolgt eine Benutzerabfrage nach FHWS Zugehörigkeit. Dazu werden die K-Nummer sowie das Passwort an das Backend gesendet und von dort am FHWS-Server authentifiziert.

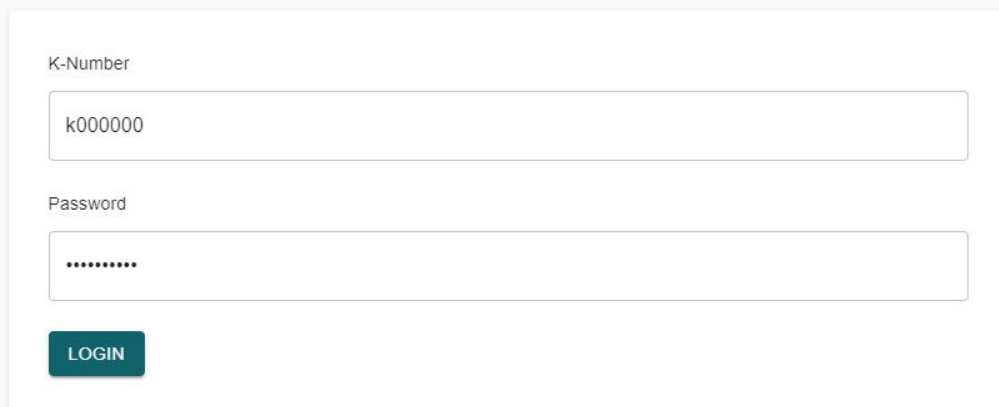


Abbildung 19: Login-Maske

Die Hauptseite des Dienstes ist in vier Bereiche unterteilt:

Ein Feld zur Eingabe des SMILES-Code, gefolgt von der Auswahl des Klassifikationsalgorithmus. Bei einem Klick auf die Lupe erscheinen die restlichen drei Bereiche.

Damit der Anwender gleich einen ersten Eindruck auf die Chemikalie erhält, wird der SMILES-Code mit Hilfe einer [Bibliothek](#) gezeichnet.

Auf Kundenwunsch sollen weitere Informationen zu der angeforderten Chemikalie geliefert werden. Dafür wird die API der größten, frei zugänglichen Datenbank für chemische Informationen *PubChem* des *National Center for Biotechnology Information* abgefragt. Diese liefert unter anderem den *IUPAC Namen* und das *Molekulargewicht*.

```

const fetchChemicalInfos = async () => {

    setChemicalInfoIsLoading( true );

    const response = await fetch(
        `https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/fastidentity/smiles/${input}/JSON` );

    if( response.status !== 200 )
    {
        setChemicalInfo( "Error!" );
        setChemicalInfoIsLoading( false );
        return;
    }

    const json = await response.json();

    const properties = json.PC_Compounds[0].props;

    const chemicalInfo:any = {};

    properties.forEach((child:any) => {
        if( child.urn.label === "IUPAC Name" && child.urn.name === "Preferred" )
            chemicalInfo.IUPACName = child.value.sval;
        else if( child.urn.label === "Molecular Formula" )
            chemicalInfo.MolecularFormula = child.value.sval;
        else if( child.urn.label === "Molecular Weight" )
            chemicalInfo.MolecularWeight = child.value.fval;
        else if( child.urn.label === "InChi" )
            chemicalInfo.InChi = child.value.sval;
        else if( child.urn.label === "InChiKey" )
            chemicalInfo.InChiKey = child.value.sval;
    });

    setChemicalInfo( chemicalInfo );
    setChemicalInfoIsLoading( false );
}

```

Abbildung 20: Methode zum Parsen weiterer Informationen

Das Ergebnis, ob ein Stoff giftig oder nicht giftig ist, erfolgt über das Backend. Dazu wird der SMILES-Code mit dem gewünschten Klassifikationsalgorithmus als Parameter übergeben. Anhand der Informationen vom Backend, ob eine Vorhersage getroffen wurde (*predicted*) und dem *Label*, wird das Ergebnis angezeigt.

```

const fetchToxicState = async () => {

    setResultIsLoading( true );

    const headers = new Headers();
    headers.append( "Authorization", `Bearer ${token}` );

    const response = await fetch(
        `${BACKEND_URL}/chemicals/smiles/${input}?model=${selectedAlgorithm}`, {method: "POST", headers: headers});

    if( response.status !== 200 )
    {
        setResult( "Error!" );
        setResultIsLoading( false );
        return;
    }

    const result = await response.json();

    setResultIsLoading( false );

    if( result.chemical.label === 0 && result.chemical.predicted === false )
        setResult( "Chemical is known and not toxic!" );
    else if( result.chemical.label === 0 && result.chemical.predicted === true )
        setResult( "Chemical is not known and were predicted as not toxic!" );
    else if( result.chemical.label === 1 && result.chemical.predicted === false )
        setResult( "Chemical is known and toxic!" );
    else if( result.chemical.label === 1 && result.chemical.predicted === true )
        setResult( "Chemical is not known and were predicted as toxic!" );
}

```

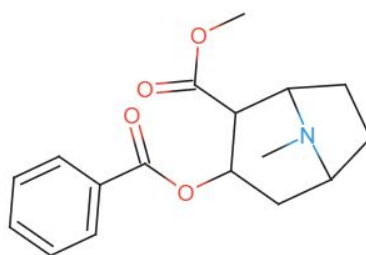
Abbildung 21: Methode für den Request ans Backend

CN1C2CCC1C(C(C2)OC(=O)C3=CC=CC=C...

Complement Naive Bayes ▾



Structure



Properties

IUPAC Name	methyl 3-benzoyloxy-8-methyl-8-azabicyclo[3.2.1]octane-2-carboxylate
Molecular Formula	C ₁₇ H ₂₁ NO ₄
Molecular Weight	303.35 g/mol
InChI	InChI=1S/C ₁₇ H ₂₁ NO ₄ /c1-18-12-8-9-13(18)15(17(20)21-2)14(10-12)22-16(19)11-6-4-3-5-7-11/h3-7,12-15H,8-10H2,1-2H3
InChIKey	ZPUCINDJBIVPJ-UHFFFAOYSA-N

Result

Chemical is not known and were predicted as not toxic!

Abbildung 22: Eingabemaske mit Ergebnissen

6.4 Machine Learning - Ömer Yanar

Der Machine Learning Teil in der Implementierung kann als Herzstück des Projekts gesehen werden. Hier werden die Eingaben gelesen, verarbeitet und am Ende wird eine Vorhersage geliefert. Im ersten Abschnitt wird die grobe Funktionsweise verdeutlicht.

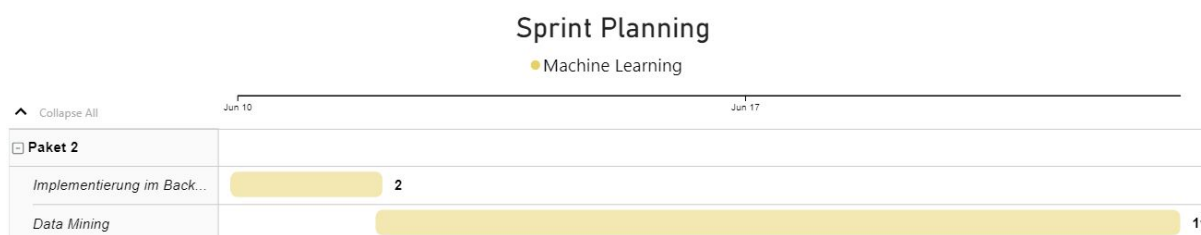


Abbildung 23: Sprint Machine Learning

6.4.1 Implementierung im Backend

Ein wichtiger Bestandteil der Software ist die Vorhersage der Toxizität. Diese soll bestimmen, ob der eingegebene Smiles Code giftig ist. Hier der Code unterteilt in die "Umwandlung" und "Modell Anwendung".

```
def predict_with_model(smiles: str, model: MlModel):  
    try:  
        mol = Chem.MolFromSmiles(smiles)  
        bit_vector = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=1024)  
  
        if model == MlModel.CNB:  
            predicted = cnb_model.predict([np.asarray(bit_vector)])  
        else:  
            predicted = gnb_model.predict([np.asarray(bit_vector)])  
    except Exception as e:  
        raise HTTPException(  
            status_code=status.HTTP_400_BAD_REQUEST,  
            detail='Error during toxicity prediction. The smiles code was probably invalid'  
        )  
  
    return predicted[0]
```

Abbildung 24: Bitvektor Erstellung Auswahl des Models

Umwandlung:

Um eine Vorhersage zu starten, wandeln wir den Smiles Codes, den wir als Parameter zur Verfügung gestellt bekommen haben, in ein Mol-Objekt um. Dies geschieht mittels Methode `"MolFromSmiles(smiles)"`. Das Ergebnis für den Smiles Code `"CCCC"` ist an der folgenden Grafik zu erkennen.



Abbildung 25: Smiles `"CCCC"` wird als Mol angezeigt

Als nächstes findet die Umwandlung in Bit-Vektoren statt. Dafür ist die Methode `"GetMorganFingerPrintAsBitVect()"` zuständig. Als Ergebnis bekommen wir ein Array das nur aus Einsen und Nullen besteht.

```
[0. 0. 0. ... 0. 0. 0.] ([ 33, 80, 294, 640, 794]
```

Abbildung 26: Das linke Array besteht nur aus Nullen und Einsen. Das rechte hingegen zeigt an welcher Stelle im Array eine 1 vorkommt.

Modell Anwendung:

Im zweiten Code Abschnitt wird bestimmt, anhand welchem Machine Learning Modell das Ergebnis ausgewertet werden soll. Da jedes Modell, basierend auf den Datensätzen, unterschiedliche Vor- und Nachteile mitbringt, ist in der von uns definierten Methode das Modell für den Algorithmus frei wählbar. Dies spielt eine große Rolle im Projekt, da einer der Problemstellungen die Performance des Machine Learning Algorithmus beinhaltet. Die Unterschiede in Performance und Auswertung werden später weiter verdeutlicht.

6.4.2 Data Mining - Matthias Nunn

Das Data Mining erfolgte nach dem Vorgehensmodell CRISP-DM.

Business Understanding

Täglich werden zehntausende von Chemikalien mit schlecht verstandenen Eigenschaften in die Umwelt freigesetzt. Um die Giftigkeit dieser Chemikalien festzustellen, sind lang andauernde und teure Tests nötig. Um diesen Vorgang abzukürzen soll eine Software entwickelt werden, die dem Anwender eine schnelle erste Vorhersage gibt.

Data Understanding

Wir erhielten vom Projektauftraggeber eine CSV-Datei mit 66.175 Zeilen. Jede Zeile stellt einen Datensatz dar und enthält eine *fortlaufende Nummer*, das *Label* (ist toxisch, ist nicht toxisch), einen *SMILES-Code* (Strukturcode, bei dem die Struktur von Molekülen stark vereinfacht als Zeichenkette wiedergegeben werden) sowie einen *Code* des NIH Chemical Genomics Center (NCGC).

Beim Durchsehen wurden keine fehlenden Werte festgestellt. Jedoch kamen viele *SMILES-Codes* mehrfach und teils mit verschiedenen *Labels* vor.

Data Preparation

Im ersten Schritt wurden die Daten zur leichteren Weiterverarbeitung aus der CSV-Datei eingelesen. Dabei werden nur die *SMILES-Codes* sowie das *Label* in Arrays gespeichert.

Gleichzeitig wurden mehrfach vorkommende Datensätze zusammengefasst. Dabei wurde ein *SMILES-Code*, der einmal als toxisch vorkommt, auch als toxisch erfasst.

```
chemicals = []

with open('tox_21_66000.csv') as csvfile:
    readCSV = csv.reader( csvfile, delimiter=',' )
    next( readCSV, None ) # skip header

    for row in readCSV:
        # row[0]: no
        # row[1]: smiles
        # row[2]: code
        # row[3]: label (0 = nicht giftig, 1 = giftig)

        isInArrayChemicals = False

        for chemical in chemicals:
            if row[1] == chemical[0]: # same smiles
                isInArrayChemicals = True

                if row[3] == "1":
                    chemical[1] = "1"

        if not isInArrayChemicals:
            chemicals.append([ row[1], row[3] ])
```

Abbildung 27: Datenvorverarbeitung I

Im nächsten Schritt wurde anhand eines SMILES-Code ein Objekt vom Typ *Mol* aus der Bibliothek *RDKit* erstellt. Dabei trat bei vereinzelt Chemikalien ein Fehler bei der Umwandlung auf, weshalb diese ignoriert wurden.

Die Klasse *Mol* besitzt eine Methode, welche einen Fingerprint in Form eines Bitvektors generiert. Die Länge des Vektors betrug je nach Durchführung* 1024 oder 2048 (Standardwert).

* Mit 2048 wurden keine großen Unterschiede bei der Erkennung festgestellt, weshalb die folgenden Werte für 1024 gelten.

```
bitVects = []
labels = []

for chemical in chemicals:

    # chemical[0]: smiles
    # chemical[1]: label (0 = nicht giftig, 1 = giftig)

    try:

        mol = Chem.MolFromSmiles( chemical[0] )
        bitVect = AllChem.GetMorganFingerprintAsBitVect( mol, 2, nBits=1024 )

        bitVects.append( np.asarray(bitVect) )
        labels.append( int(chemical[1]) )

    except Exception as e:

        print( f"Fehler bei {chemical[0]}" )
```

Abbildung 28: Datenvorverarbeitung II

Nach der Datenvorverarbeitung bestand der Datensatz aus 1949 toxischen Stoffen und 5629 nicht toxischen Stoffen. Das entspricht einer Verteilung von etwa 75% zu 25%.

Modeling

Um eine möglichst genaue Vorhersage zu treffen, wurden die Daten auf acht verschiedene Algorithmen angewendet. Die Auswahl erfolgte anhand in scikit-learn implementierten Algorithmen. Um ein möglichst breites Spektrum abzubilden, wurden Algorithmen aus jeder "Kategorie" (Support Vector Classifier, K Nearest Neighbor, Naive Bayes, Ensemble Classifier) verwendet.

Mit Hilfe einer fünffachen Kreuzvalidierung wurden die Daten in 80% Trainingsdaten und 20% Testdaten aufgeteilt und trainiert.

```

algorithms = {
    "SupportVectorClassifier": SVC(),
    "LinearSupportVectorClassifier": LinearSVC(),
    "KNearestNeighbor": KNeighborsClassifier( n_neighbors=77 ), # Wurzel aus 6062 (80%)
    "GaussianNaiveBayes": GaussianNB(),
    "MultinomialNaiveBayes": MultinomialNB(),
    "ComplementNaiveBayes": ComplementNB(),
    "RandomForestClassifier": RandomForestClassifier(),
    "DecisionTreeClassifier": DecisionTreeClassifier()
}

for key in algorithms:

    value = algorithms[key]

    print( value )

    model = value

    scores = []
    fnr = [] # false negative rate
    tpr = [] # true positive rate

    kFold = KFold( n_splits=5, shuffle=True )

    for train_index, test_index in kFold.split( X ):

        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        model.fit( X_train, y_train )

        scores.append( model.score(X_test, y_test) )

        confusionMatrix = confusion_matrix( y_test, model.predict(X_test) )

        fnr.append( confusionMatrix[1][0] / (confusionMatrix[1][0] + confusionMatrix[1][1]) ) # fn / (fn + tp)
        tpr.append( confusionMatrix[1][1] / (confusionMatrix[1][1] + confusionMatrix[1][0]) ) # tp / (tp + fn)

        print( confusionMatrix )

    print( np.mean(scores) )
    print( np.mean(fnr) )
    print( np.mean(tpr) )

```

Abbildung 29: Modeling

Evaluation

Es werden für jeden Algorithmus die durchschnittliche *Genauigkeit* / *false negative rate* / *true positive rate* sowie für jede Iteration der Kreuzvalidierung die *confusion Matrix* angegeben, wobei gilt:

tn = true negative = Korrekte Ablehnung

fp = false positive = Falscher Alarm

fn = false negative = Fehlschlag

tp = true positive = Treffer

Support Vector Classifier

Genauigkeit: 80,26%

false negative rate: 68,30%

true positive rate: 31,70%

		<i>Iteration 1</i>		<i>Iteration 2</i>		<i>Iteration 3</i>		<i>Iteration 4</i>		<i>Iteration 5</i>	
<i>tn</i>	<i>fp</i>	1073	42	1092	34	1122	30	1101	32	1077	26
<i>fn</i>	<i>tp</i>	275	126	274	116	240	124	259	123	284	128

Linear Support Vector Classifier

Genauigkeit: 75,30%

false negative rate: 58,02%

true positive rate: 41,98%

		<i>Iteration 1</i>		<i>Iteration 2</i>		<i>Iteration 3</i>		<i>Iteration 4</i>		<i>Iteration 5</i>	
<i>tn</i>	<i>fp</i>	974	141	958	168	956	154	1005	145	994	134
<i>fn</i>	<i>tp</i>	218	183	212	178	232	174	213	152	255	132

K Nearest Neighbor

Genauigkeit: 75,02%

false negative rate: 96,65%

true positive rate: 3,35%

		<i>Iteration 1</i>		<i>Iteration 2</i>		<i>Iteration 3</i>		<i>Iteration 4</i>		<i>Iteration 5</i>	
<i>tn</i>	<i>fp</i>	1126	3	1138	1	1130	2	1132	1	1094	2
<i>fn</i>	<i>tp</i>	375	12	362	15	372	12	367	15	408	11

Gaussian Naive Bayes

Genauigkeit: 68,82%

false negative rate: 49,12%

true positive rate: 50,88%

		<i>Iteration 1</i>		<i>Iteration 2</i>		<i>Iteration 3</i>		<i>Iteration 4</i>		<i>Iteration 5</i>	
<i>tn</i>	<i>fp</i>	824	307	839	285	839	285	851	264	871	264
<i>fn</i>	<i>tp</i>	184	201	196	196	184	208	215	185	179	201

Multinomial Naive Bayes

Genauigkeit: 74,33%

false negative rate: 52,76%

true positive rate: 47,24%

		<i>Iteration 1</i>		<i>Iteration 2</i>		<i>Iteration 3</i>		<i>Iteration 4</i>		<i>Iteration 5</i>	
<i>tn</i>	<i>fp</i>	955	169	932	193	971	179	916	187	938	189
<i>fn</i>	<i>tp</i>	214	178	220	171	195	171	210	202	189	199

Complement Naive Bayes

Genauigkeit: 68,86%

false negative rate: 42,01%

true positive rate: 57,99%

		<i>Iteration 1</i>		<i>Iteration 2</i>		<i>Iteration 3</i>		<i>Iteration 4</i>		<i>Iteration 5</i>	
<i>tn</i>	<i>fp</i>	793	331	828	287	823	285	839	310	805	328
<i>fn</i>	<i>tp</i>	163	229	160	241	185	223	159	207	152	230

Random Forest Classifier

Genauigkeit: 78,91%

false negative rate: 65,77%

true positive rate: 34,23%

		<i>Iteration 1</i>		<i>Iteration 2</i>		<i>Iteration 3</i>		<i>Iteration 4</i>		<i>Iteration 5</i>	
<i>tn</i>	<i>fp</i>	1067	65	1054	54	1076	64	1040	81	1075	53
<i>fn</i>	<i>tp</i>	255	129	253	155	253	123	263	131	257	130

Decision Tree Classifier

Genauigkeit: 71,60%

false negative rate: 55,65%

true positive rate: 44,35%

		<i>Iteration 1</i>		<i>Iteration 2</i>		<i>Iteration 3</i>		<i>Iteration 4</i>		<i>Iteration 5</i>	
<i>tn</i>	<i>fp</i>	893	238	926	203	908	216	948	200	888	209
<i>fn</i>	<i>tp</i>	205	180	203	184	242	150	195	172	241	177

Deployment

Ziel ist es, die Anzahl der Fehlschläg gering zu halten, auch wenn das bedeutet, dass die Genauigkeit sinkt. Daher sollte die false negative rate so gering wie möglich sein. Sowohl beim *Gaussian Naive Bayes* als auch beim *Complement Naive Bayes* lag die Quote unter 50 Prozent. Um dem Kunden ein möglichst gutes Ergebnis zu gewährleisten, entschieden wir uns daher mehrere Modelle bereitzustellen.

7 Lokales Starten der Anwendung - Nikolas Woldt

Um die Applikation lokal zu verwenden wird eine Linux Distribution wie Ubuntu 18.04 mit einer funktionierenden Docker Engine benötigt. Wie Docker auf Ubuntu zu installieren ist erfährt man [hier](#). Falls noch nicht vorhanden, muss auch docker-compose installiert werden. Der Befehl auf Ubuntu dazu lautet `sudo apt-get install docker-compose`.

Nun kann die Multicontainer-Applikation im Root Verzeichnis des Projekts mit `(sudo) docker-compose up -d` gestartet werden.

Des Weiteren muss das Datenbank Backup mit den bereits vorhanden Chemikalien eingespielt werden. Dafür können verschiedene Tools benutzt werden, wie zum Beispiel *pgadmin*, *IntelliJ Idea* oder *psql*. Im folgenden wird beschrieben, wie dies mit *psql* auf der Konsole unter Ubuntu funktioniert.

Psql ist ein Commandline Tool zur Verwaltung und Benutzung von PostgreSQL Datenbanken. Installiert wird dies mit `sudo apt-get install postgresql-client-12`. Mit dem folgenden Befehl kann im Root Verzeichnis der Applikation dann das Backup eingespielt werden, sofern der Datenbank Container derzeit aktiv ist: `psql --file=dump.sql --username=postgres --host=localhost --port=5432`. Falls nach dem Passwort gefragt wird, ist dieses für die lokale Entwicklung in der `docker-compose.yml` unter `services:db:environment` hinterlegt und ist, falls nicht manuell geändert, `password`.

Nun kann das Frontend im Browser über `localhost:8000` und das Backend über `localhost:8888` erreicht werden.

Um die API des Backends genauer zu betrachten, kann die API Dokumentation über `localhost:8888/docs` aufgerufen werden.

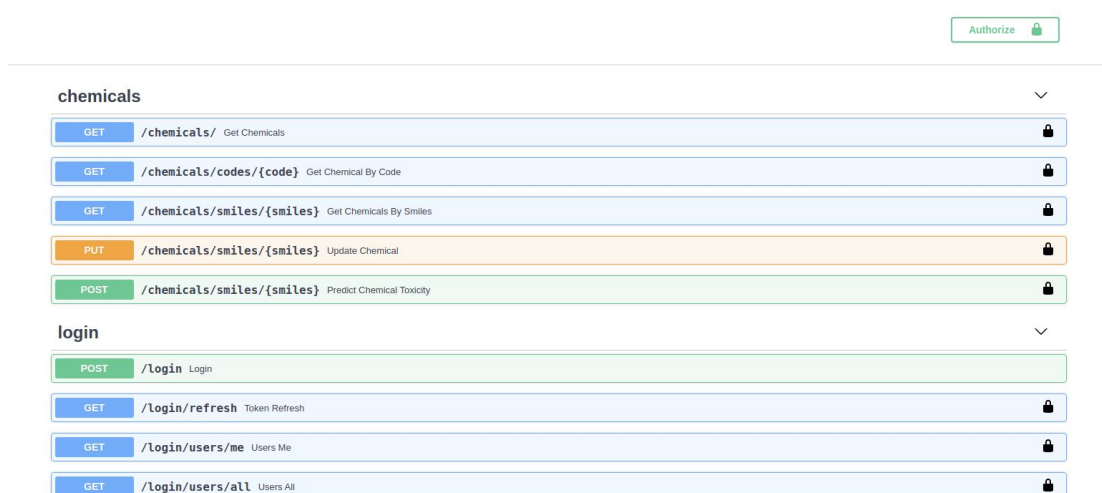


Abbildung 30: Swagger UI für API Dokumentation

8 Fazit

Dieses Projekt stellte sich, entsprechend den derzeitigen Umständen durch Corona, für jeden von uns als eine große Herausforderung dar. Das verkürzte Sommersemester hatte die Folgen, dass mehr Vorlesungsstoff in einem kürzeren Zeitraum bewerkstelligt werden musste und die Sperrung der Fachhochschule verhinderte ein gemeinsames Arbeiten unter den Teammitgliedern. Dennoch können wir stolz berichten, dass wir das Projekt mit vollem Erfolg abschließen konnten.

Diese Herausforderung stellte sich für uns als einen großen Erfahrungszuwachs dar. Der Zeitmangel verdeutlichte uns, wie essentiell eine gute Planung für den Erfolg in einem Projekt ist. Durch die klare Verteilung der Rollen war jedem im Projekt klar, welche Aufgaben man erfüllen musste. Auch die Dokumentation der Projektplanung erwies sich als sehr hilfreich, da dadurch die Zusammenhänge der einzelnen Tasks verdeutlicht wurden. Ebenfalls sorgten die frequentierten Meetings und der konstante Austausch von Informationen für ein positives Gemeinschaftsgefüge. Somit entstand ein großer Zusammenhalt im Team, obwohl persönliche Treffen nicht möglich war.

Zum Abschluss können wir sagen, dass die schwerwiegenden Umstände sich sehr positiv auf unseren zukünftigen Werdegang ausgewirkt haben. In der Arbeitswelt gibt es häufig Projekte, die ein persönliches Treffen nicht ermöglichen können und deshalb diese nur virtuell stattfinden. Auch der Zeitmangel und das Priorisieren der unterschiedlichen Aufgaben spielt in jedem Projekt eine entscheidende Rolle. Wir sind froh, diese Erfahrungen gemacht zu haben und fühlen uns in unseren Fähigkeiten bestätigt.

9 Tabellenverzeichnis

Tabelle 1: Aufteilung des Pakets	06
Tabelle 2: Erfahrungen der Teammitglieder	08
Tabelle 3: Rollen und deren Beschreibung	09
Tabelle 4: Kommunikationsmatrix	10
Tabelle 5: Tools die im Projekt genutzt werden	12
Tabelle 6: Paket 1 Inhalt	15
Tabelle 7: Aufteilung des zweiten Pakets	19

10 Abbildungsverzeichnis

Abbildung 1: Pakete und deren Inhalt	05
Abbildung 2: Sprintplan Paket 0	07
Abbildung 3: Rollenverteilung	08
Abbildung 4: Informationsaustausch mit Discord	13
Abbildung 5: Power BI Report der Tasks	14
Abbildung 6: Gefilterte Darstellung des Power BI Reports	15
Abbildung 7: Sprintplanung Paket 1	15
Abbildung 8: Frontend Dockerfile	16
Abbildung 9: Backend Dockerfile	17
Abbildung 10: Sprintplanung des Backends	19
Abbildung 11: Chemical SQLAlchemy Definition	20
Abbildung 12: GET Endpunkt zum Laden mehrerer Chemikalien	21
Abbildung 13: PUT Endpunkt zum Updaten von Chemikalien	22
Abbildung 14: POST Endpunkt für neue Chemikalien	23
Abbildung 15: Sprintplanung für die Datenbank	24
Abbildung 16: Datenbank Integration	24
Abbildung 17: Beispiel für eine Datenbankabfrage	25
Abbildung 18: Sprint Frontend	25
Abbildung 19: Login-Maske	26
Abbildung 20: Methode zum Parsen weiterer Informationen	27
Abbildung 21: Methode für den Request ans Backend	28
Abbildung 22: Eingabemaske mit Ergebnissen	29
Abbildung 23: Sprint Machine Learning	30
Abbildung 24: Bitvektor Erstellung Auswahl des Models	30
Abbildung 25: Smiles "CCCC" wird als Mol angezeigt	31
Abbildung 26: Das linke Array [...]	31
Abbildung 27: Datenvorverarbeitung I	32
Abbildung 28: Datenvorverarbeitung II	33
Abbildung 29: Modeling	34
Abbildung 30: Swagger UI für API Dokumentation	38