Overview
The aim of this task is to pull information from an API into a database and query it to address the task below.
We would like you to use SQLAlchemy for this task, and store the results in a SQLite database. Bear in mind that, with the free account for the API, there is a limit of 30 requests per day.
We have provided some skeleton code to get you started accessing the API.

Please submit your challenge on a gitlab repo we can have access to.
We will be evaluating the python code, python project structure and the gitlab pipeline setup, together with all the tools supporting the software development process.
You can use gitlab issues to manage the tasks associated with the project. Please make sure there is a relationship between issues and branches or commits.

Task Description
Given a list of train station codes (three letters), and a starting time, find the arrival time at the destination station.

For example,

- given the following set of train codes: ['LBG','SAJ','NWX','BXY']
- a start time of '2025-06-04T07:00:00+01:00',
- and choosing the train that arrives earliest to the next station,

The arrival time is 08:11.

from LBG to SAJ departure 07:19 arrival 07:28
from SAJ to NWX departure 07:32 arrival 07:34
from NWX to BXY departure 07:45 arrival 08:11

Notes:
- You should store information in the database to speed up subsequent runs of the code.
- You should follow the route, eg LBG, to SAJ, then SAJ to NWX, then NWX to BXY **not** find the shortest time between start and destination stations eg LBG to BXY.
- You will be given the maximum time (in minutes) that someone is willing to wait at a station. If they have to wait longer you should display an error.
- The Transport API can be found https://developer.transportapi.com/. You need to sign up for a free trial account.

Bonus Points
- Implement an API that allows you call your function through a REST/gRPC call

References

https://developer.transportapi.com/docs#get-/v3/uk/train/station_timetables/-id-.json

Code example for retrieving the train departures from a station calling at a destination station

```
with httpx.Client(
   params=dict(
       app_id=os.getenv("app_id"),
       app_key=os.getenv("app_key"),
       live=False,
       station_detail="calling_at",
       train_status="passenger",
       datetime=starting_time,
       limit=1000,
       calling_at=station_to,
   )
) as client:
   response = client.get(f"https://transportapi.com/v3/uk/train/station_timetables/{station_from}.json")
for departure in response.json()["departures"]["all"]:
          aimed_departure_time=departure["aimed_departure_time"],
          aimed_arrival_time=departure["station_detail"]["calling_at"][0]["aimed_arrival_time"],
```

For example for 'LBG' station to 'SAJ' station the response.json() looks like


```
{
 "date": "2025-06-04",
 "time_of_day": "07:00",
 "request_time": "2025-06-04T11:54:44+01:00",
 "station_name": "London Bridge",
 "station_code": "LBG",
 "departures": {
   "all": [
     {
       "mode": "train",
       "service": "24659005",
       "train_uid": "P18700",
       "platform": "1",
       "operator": "SE",
       "operator_name": "Southeastern",
       "aimed_departure_time": "07:07",
       "aimed_arrival_time": "07:06",
       "aimed_pass_time": "",
       "origin_name": "London Cannon Street",
       "destination_name": "London Cannon Street",
       "source": "ATOC",
       "category": "OO",
       "service_timetable": {
         "id": "https://transportapi.com/v3/uk/train/service_timetables/P18700:2025-06-04.json?app_id=XXX&app_key=XXX"
       },
       "station_detail": {
         "calling_at": [
           {
             "station_name": "St Johns",
             "platform": "",
             "station_code": "SAJ",
             "tiploc_code": "STJOHNS",
             "aimed_arrival_time": "08:31",
             "aimed_departure_time": "08:32",
             "aimed_pass_time": ""
           }
         ]
       }
     },
```


Code example for setting up sqlalchemy

```
import sqlalchemy.orm

Base = sqlalchemy.orm.declarative_base()



@contextlib.contextmanager
def get_engine(
   drivername: str, username: str, password: str, host: str, database: str) -> Generator[Engine, Any, None]:
   """Return a sqlalchemy engine. An engine is a pool and a factory of connections.
   All the connections are disposed once the
   """
   an_url = sqlalchemy.engine.URL.create(
       drivername=drivername, username=username, password=password, host=host, database=database
   )
   logging.getLogger("sqlalchemy.engine").setLevel(logging.ERROR)
   engine = sqlalchemy.create_engine(url=an_url)
   yield engine
   engine.dispose(close=True)


@contextlib.contextmanager
def get_session(engine: sqlalchemy.engine.Engine) -> Generator[Session, Any, None]:
   session = sqlalchemy.orm.sessionmaker(autocommit=False, autoflush=False, bind=engine)
   with session() as a_session:
       yield a_session
   a_session.close()

def generate_ddl(drivername, username, host, password, database):
   with get_engine(
       drivername=drivername, username=username, password=password, host=host, database=database
   ) as engine:
       with get_session(engine=engine) as a_session:
           Base.metadata.create_all(a_session.get_bind())
```