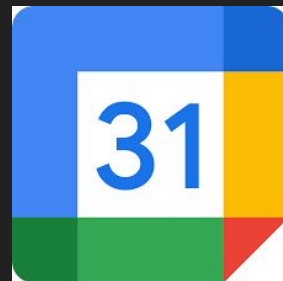


Canvas to Google

Assignment Calendar Connections

Farley, Gallagher, Utter



Project Utilized:

- Python
- TKinter
- SQLAlchemy
- Pydantic
- Google Cloud APIs
- JSON
- FastAPI

Getting Classes and Assignments

- Utilize a request statement to access the Canvas website
- auth_header is the Canvas Token

```
# Iterate over the filtered courses
for course in specific_course_ids:
    # Make a request to get assignments for the current course
    assignments_url = BASE_URL + f'/api/v1/courses/{specific_course_ids[count]}/assignments'
    assignments_params = {"per_page": str(PER_PAGE)}
    assignments_request = requests.get(assignments_url, headers=auth_header, params=assignments_params)
    assignments_request.raise_for_status()

    # Add assignments to the list
    assignments = assignments_request.json()
```

Storing in a DataFrame

- Assignments then get placed into a DataFrame to make creating Google Calendar events easy

```
assignments_df = pd.DataFrame(assignments)
assignments_result = assignments_df.to_string(
    columns=['id', 'name', 'due_at', 'points_possible']
)
print(f"Assignments for {specific_course_ids[count]}:")
print("-----")
print(assignments_result)
print("\n")
```

Inserting Assignments

- Grab name and time from Canvas, then create an event to put into the calendar

```
# Create an event in Google Calendar
event = {
    'summary': assignment_name,
    'description': f'Assignment due: {assignment_name}',
    'start': {'dateTime': due_date.isoformat(), 'timeZone': 'UTC'},
    'end': {'dateTime': (due_date + timedelta(hours=1)).isoformat(), 'timeZone': 'UTC'},
}

# Insert the event
event = service.events().insert(calendarId='primary', body=event).execute()

print(f'Event created: {event.get("htmlLink")}')

```

Hosted Server for the backend

- Code that starts a server for the database

```
conn_string = f"postgresql://{USERNAME}:{PASSWORD}@{HOST}/{NAME}"  
engine = create_engine(conn_string)  
BaseTable.metadata.create_all(bind=engine)
```

```
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

```
router = APIRouter()
```

5 usages 👤 JackGallagher41

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

```
from fastapi import FastAPI
```

```
from api import userApis, courseApis, assignmentApis
```

```
app=FastAPI()
```

```
app.include_router(userApis.router, prefix="/users", tags=["Users"])
```

```
app.include_router(courseApis.router, prefix="/courses", tags=["Courses"])
```

```
app.include_router(assignmentApis.router, prefix="/assignments", tags=["Assignments"])
```

Tables created in Python using SQLAlchemy

```
class User(BaseTable):  
    __tablename__ = "users"  
    uid = Column(Integer, primary_key=True, index=True)  
    username = Column(String)  
    canvasAccessToken = Column(String)  
    googleCalendarAccessToken = Column(String)
```

```
class Assignment(BaseTable):  
    __tablename__ = "assignments"  
    assignmentID = Column(Integer, primary_key=True, index=True)  
    title = Column(String)  
    description = Column(String)  
    dueDate = Column(DateTime)  
    courseID = Column(Integer)
```

JSON Models

- Example of models used in correlation with rest api to put the data into a JSON format

```
class userCreate(BaseModel):  
    username: str  
    canvasAccessToken: str  
    googleCalendarAccessToken: str
```

4 usages 👤 JackGallagher41

```
class courseResponse(BaseModel):  
    courseID: int  
    courseName: str  
    profFName: str  
    profLName: str  
    crn: str  
    uid: int
```


Rest APIS

- Example of rest apis used to communicate the ui form and canvascomms with the database

```
@router.delete(path: "{user_id}", response_model=str)
async def deleteUser(user_id: int, session: Session = Depends(get_db)):
    userToDelete = session.query(User).filter(User.uid == user_id).first()
    if userToDelete:
        session.delete(userToDelete)
        session.commit()
        return f"User with the following ID has been deleted: {user_id} "
    else:
        raise HTTPException(status_code=404, detail="User not found")
```

API that gets a user based off their Canvas Token

👤 JackGallagher41

```
@router.get(path: "/CAT_retrieve/{user_CAT}", response_model=dict)
async def getUser(user_CAT: str, session: Session = Depends(get_db)):
    user = session.query(User).filter(User.canvasAccessToken == user_CAT).first()
    if user is None:
        raise HTTPException(status_code=404, detail="User not found")
    uid = {"user_uid": user.uid}

    return uid
```

API that gets a user based off their user id

👤 JackGallagher41

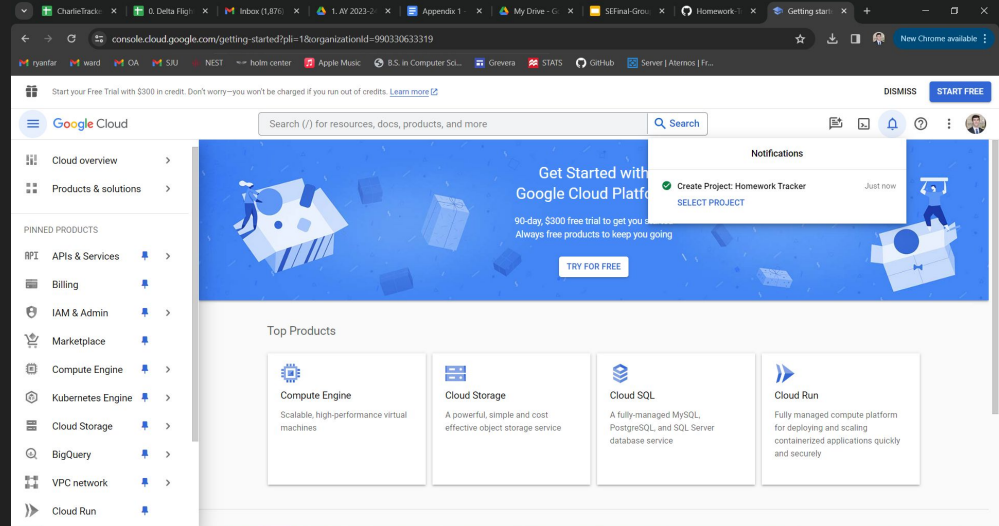
```
@router.get(path: "/id_retrieve/{user_id}", response_model=userResponse)
async def getUser(user_id: int, session: Session = Depends(get_db)):
    user = session.query(User).filter(User.uid == user_id).first()
    if user is None:
        raise HTTPException(status_code=404, detail="User not found")
    return user
```

```
@router.post(path: "/create/{create_user}", response_model=userResponse)
async def createUser(user: userCreate, session: Session = Depends(get_db)):
    newUser = User(**user.dict())
    session.add(newUser)
    session.commit()
    session.refresh(newUser)
    return newUser
```

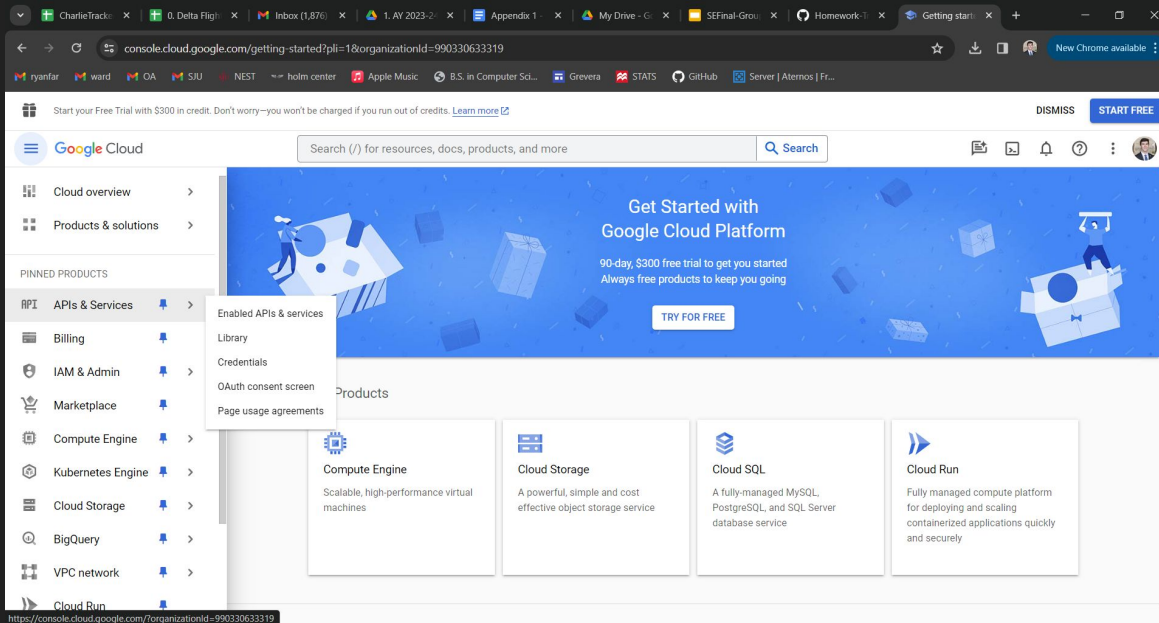
Google API Steps

Steps for getting your Google Path JSON

1. Search in your browser 'Google Cloud Console' and click on Google's link
2. Click on the 'Select a Project' button and change the organization at the top to sju.edu, then click 'NEW PROJECT'
3. Name the project 'Homework Tracker' and leave the location and organization as sju.edu, then create



4. Now navigate to the left side of your screen to 'API's and Services', then select your project. At the top hit '+ ENABLE APIS AND SERVICES'



Enabled APIs & services

Page not viewable for organizations. To view this page, select a project.

Select a recent project

Homework Tracker

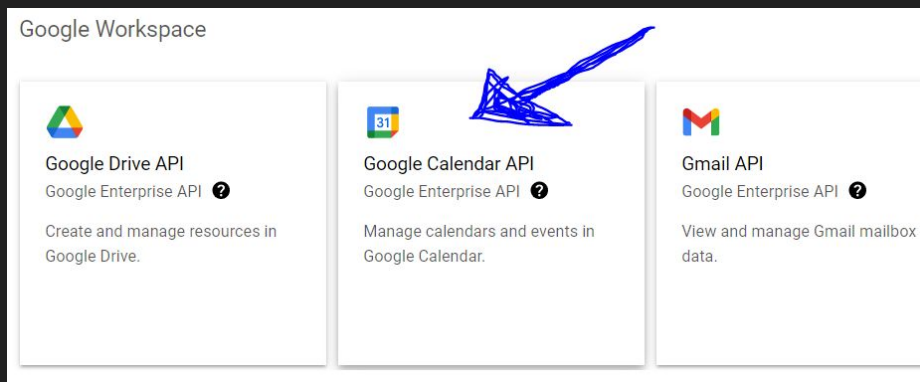
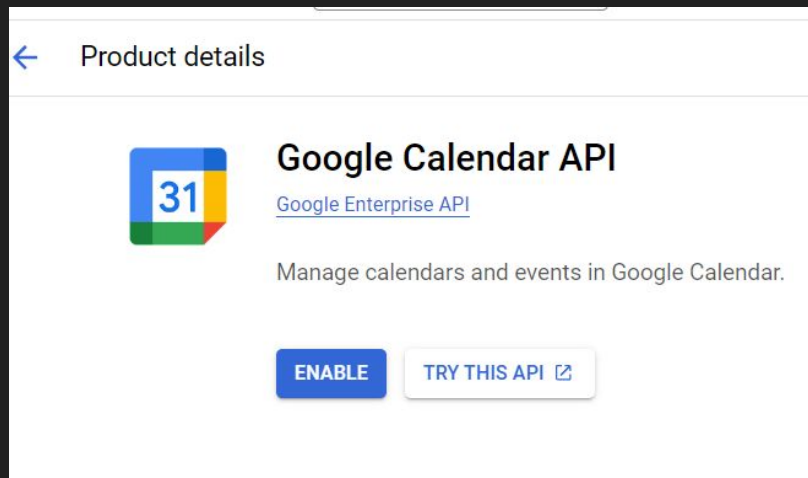
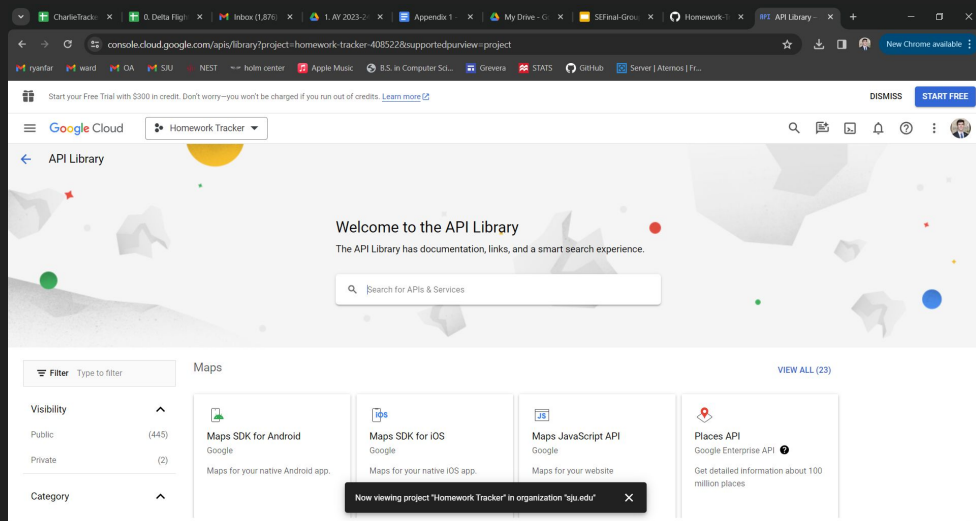
Project ID: homework-tracker-408522

Organization: sju.edu

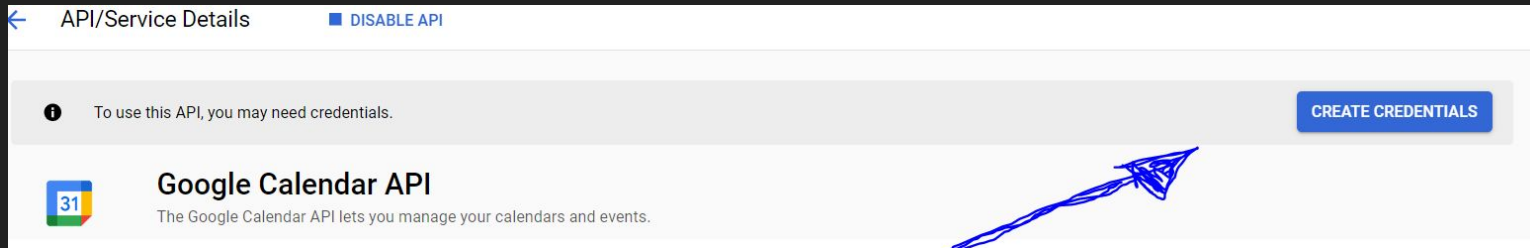
Accessed just now

Ancestry: sju.edu

5. Scroll down to the Google Calendar API, click on it and hit 'ENABLE'



6. Now hit 'CREATE CREDENTIALS' on the top of your screen. Leave the API on Google Calendar and select 'Application data' and hit 'NEXT'



Select an API *
Google Calendar API ▼

What data will you be accessing? *

Different credentials are required to authorize access depending on the type of data that you request. [Learn more](#) ↗

☐ User data ?
Data belonging to a Google user, like their email address or age. User consent required. This will create an OAuth client.

☒ Application data
Data belonging to your own application, such as your app's Cloud Firestore backend. This will create a service account.

NEXT

7. Name it the same as what you named the project then skip past the optional parts and finalize it


1 Service account details

Display name for this service account

Service account ID *

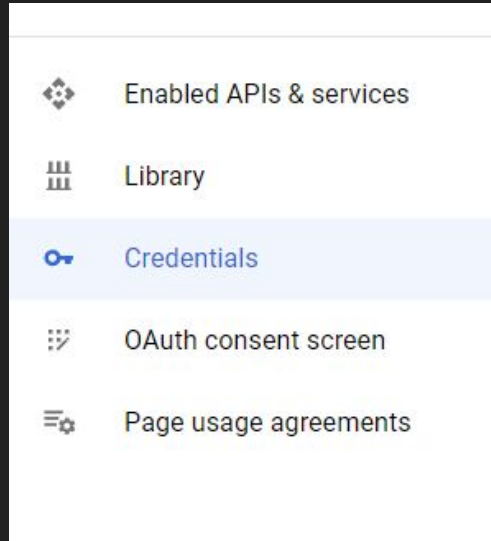
✕ ↺

Email address: homework-tracker@homework-tracker-408522.iam.gserviceaccount.com

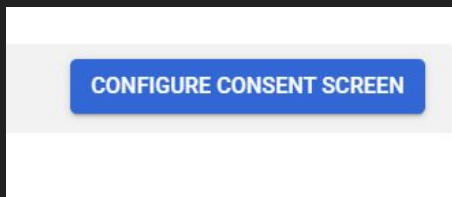
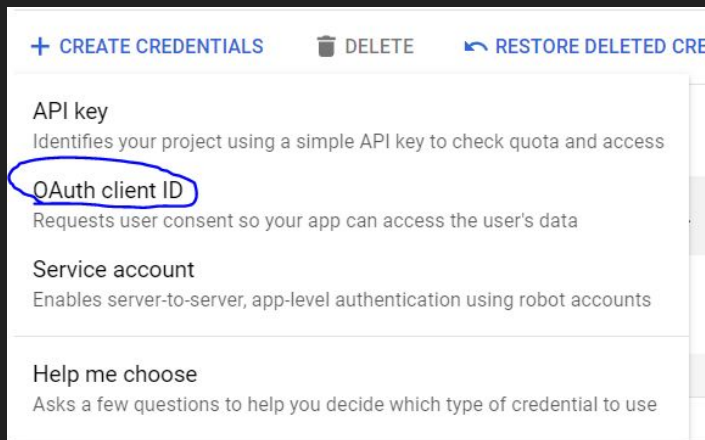


Describe what this service account will do

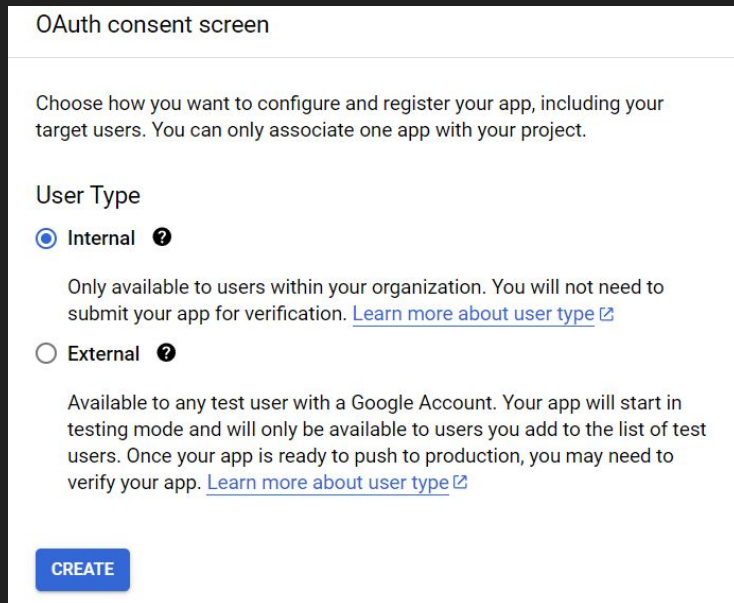
8. Now navigate to the 'Credentials' tab on the left side of your screen and create another credential by hitting the '+ CREATE CREDENTIALS' at the top of your screen



9. Click 'OAuth Client ID' and then hit 'Configure Consent Screen'



10. Make sure you click 'Internal' then create.



11. Type in the same project name you have been using into the 'App name' box and then use your school email in the following two mandatory boxes
12. Skip past the 'Scopes' page and finalize the consent screen.
13. Repeat step #8

App information

This shows in the consent screen, and helps end users know who you are and contact you

App name *

Homework Tracker

The name of the app asking for consent

User support email *


rf745933@sju.edu

For users to contact you with questions about their consent. [Learn more](#)

App logo

This is your logo. It helps people recognize your app and is displayed on the OAuth

14. Click 'OAuth Client ID' again and put the application type to Web Application and name it


 Create OAuth client ID

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See [Setting up OAuth 2.0](#) for more information. [Learn more](#) about OAuth client types.

Application type *
Web application

Name *
Web client 1

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

 The domains of the URIs you add below will be automatically added to your [OAuth consent screen](#) as [authorized domains](#).

15. Scroll down until you see Authorized URI's. You are going to want to make six. Make sure your six are:

Authorized redirect URIs ?

For use with requests from a web server

URIs 1 *
https://calendar.google.com/calendar/u/0/r

URIs 2 *
https://calendar.google.com/calendar/u/0/r/

URIs 3 *
https://www.calendar.google.com/calendar/u/0/r

URIs 4 *
https://www.calendar.google.com/calendar/u/0/r/

URIs 5 *
http://localhost:8080/

URIs 6 *
http://localhost:8080

[+ ADD URI](#)

Authorized redirect URIs ?

For use with requests from a web server

[+ ADD URI](#) ←

16. Once it's created you should get a pop-up confirming creation. At the bottom of that pop-up hit 'DOWNLOAD JSON'

17. Find that downloaded file in your downloads and copy the path of the file.

