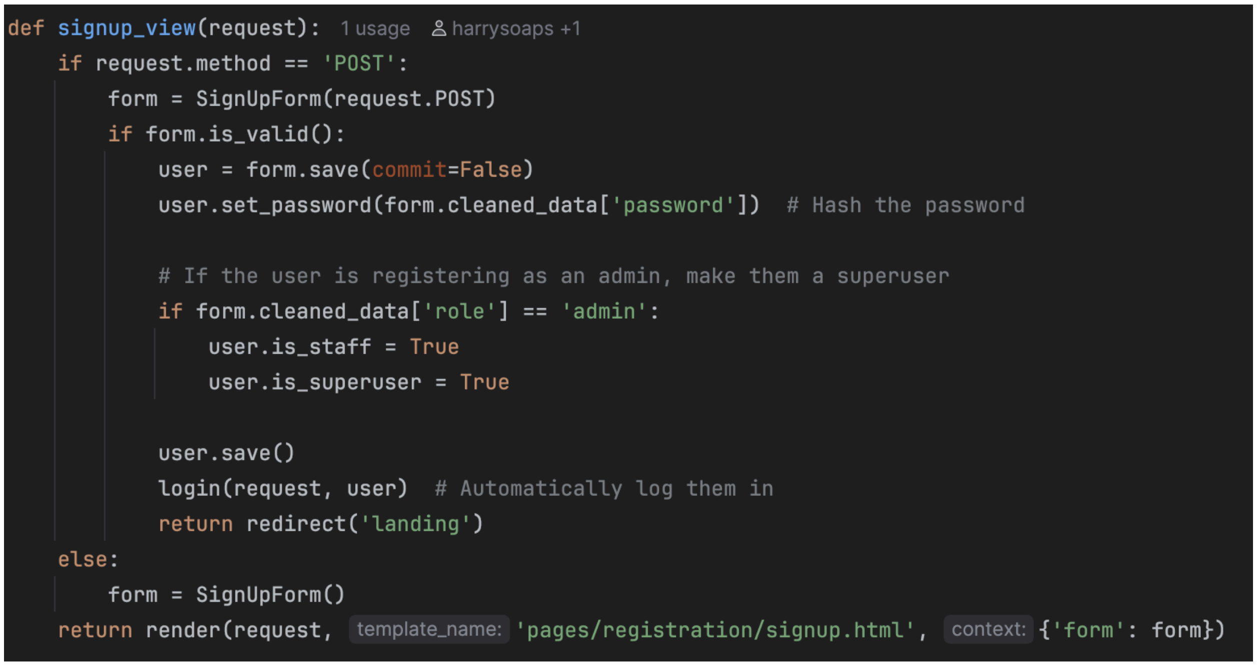
1. Identify one part of your project 2 code that abided by SOLID or GRASP principles. Explain the benefits of these principles. Your answer must include screenshots of any relevant code to support your statements.

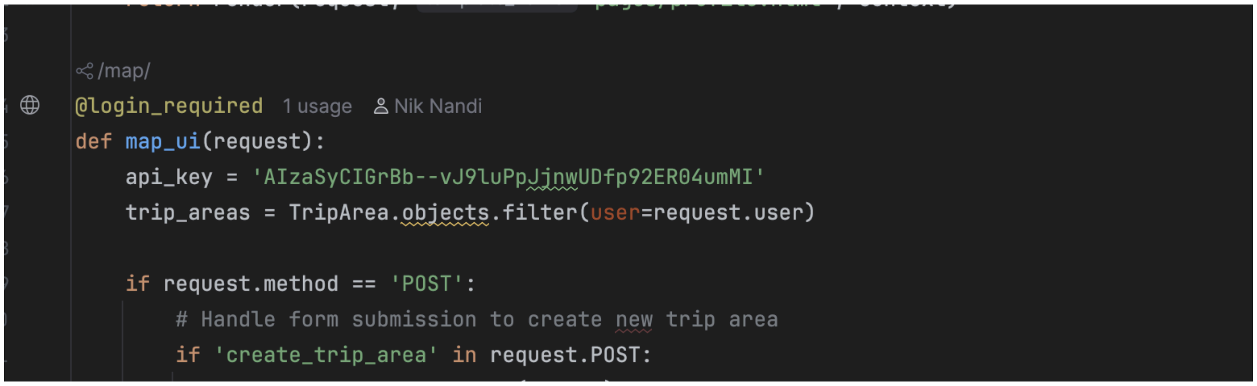
The signup_view portion of our project exhibits the single responsibility principle in SOLID because it only serves and handles one purpose, which is signing up for the website. It doesn't touch any other parts of the system.



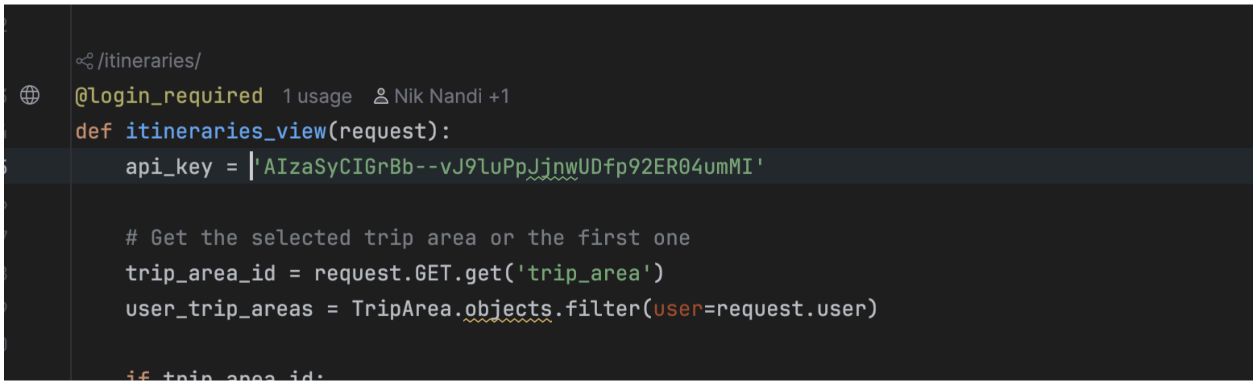
```
def signup_view(request):  1 usage  harrysoaps +1
    if request.method == 'POST':
        form = SignUpForm(request.POST)
        if form.is_valid():
            user = form.save(commit=False)
            user.set_password(form.cleaned_data['password'])  # Hash the password

            # If the user is registering as an admin, make them a superuser
            if form.cleaned_data['role'] == 'admin':
                user.is_staff = True
                user.is_superuser = True

            user.save()
            login(request, user)  # Automatically log them in
            return redirect('landing')
    else:
        form = SignUpForm()
    return render(request, template_name: 'pages/registration/signup.html', context: {'form': form})
```

2. Identify and remove two code smells in your project 2 codebase. Your answer must include screenshots of the code before and after your refactoring

1. Hardcoded API Key
    a. Instead of retyping the API key every time it's used, we moved it to the project settings.py file and imported it into files we needed to use it

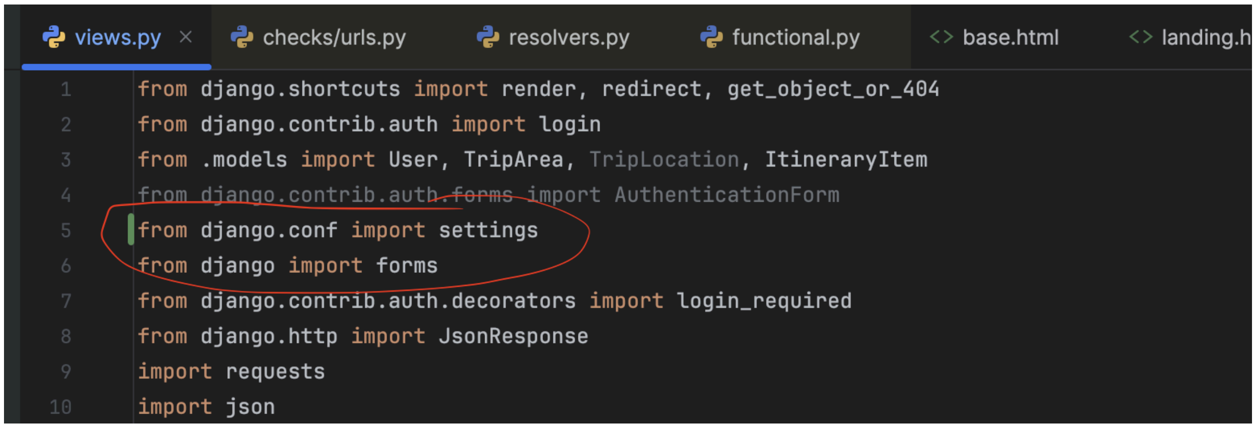


```
/map/
@login_required  1 usage  Nik Nandi
def map_ui(request):
    api_key = 'AIzaSyCIGrBb--vJ9luPpJjnwUDfp92ER04umMI'
    trip_areas = TripArea.objects.filter(user=request.user)

    if request.method == 'POST':
        # Handle form submission to create new trip area
        if 'create_trip_area' in request.POST:
```

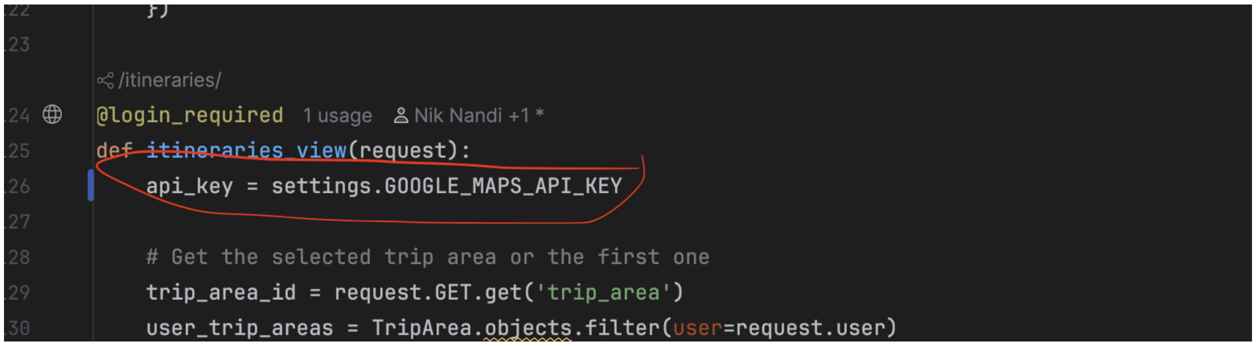


```
@login_required
def itineraries_view(request):
    api_key = 'AIzaSyCIGrBb--vJ9luPpJjnwUDfp92ER04umMI'

    # Get the selected trip area or the first one
    trip_area_id = request.GET.get('trip_area')
    user_trip_areas = TripArea.objects.filter(user=request.user)

    if trip_area_id:
```

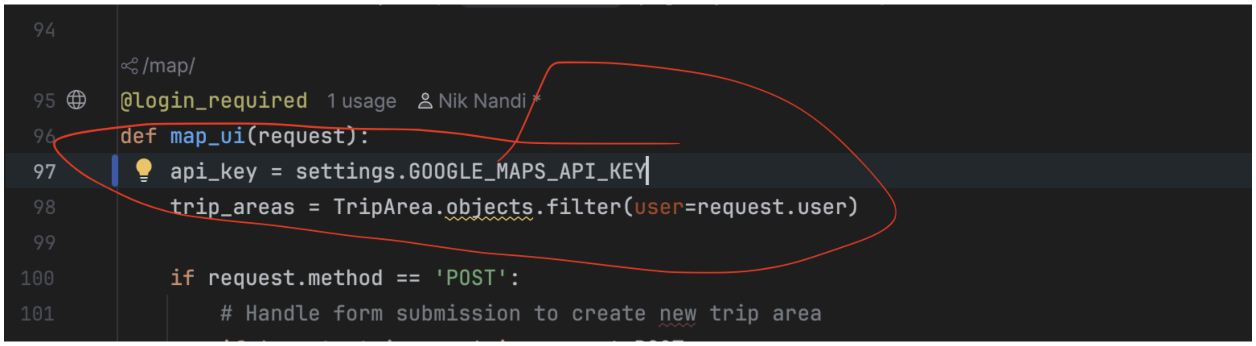
AFTER:



```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import login
from .models import User, TripArea, TripLocation, ItineraryItem
from django.contrib.auth.forms import AuthenticationForm
from django.conf import settings
from django import forms
from django.contrib.auth.decorators import login_required
from django.http import JsonResponse
import requests
import json
```

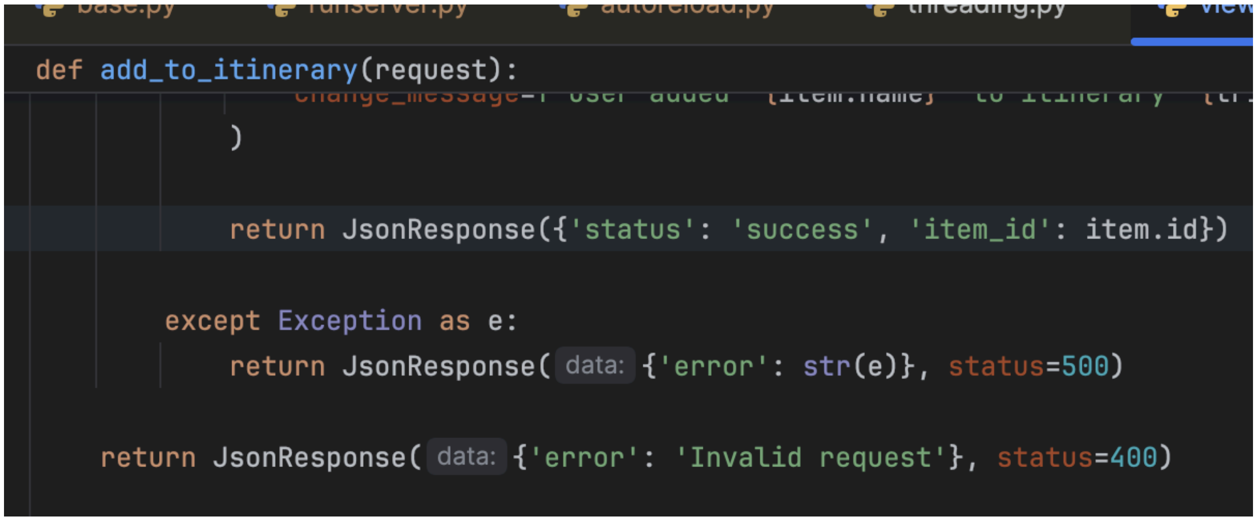


```
    })

@login_required
def itineraries_view(request):
    api_key = settings.GOOGLE_MAPS_API_KEY

    # Get the selected trip area or the first one
    trip_area_id = request.GET.get('trip_area')
    user_trip_areas = TripArea.objects.filter(user=request.user)
```

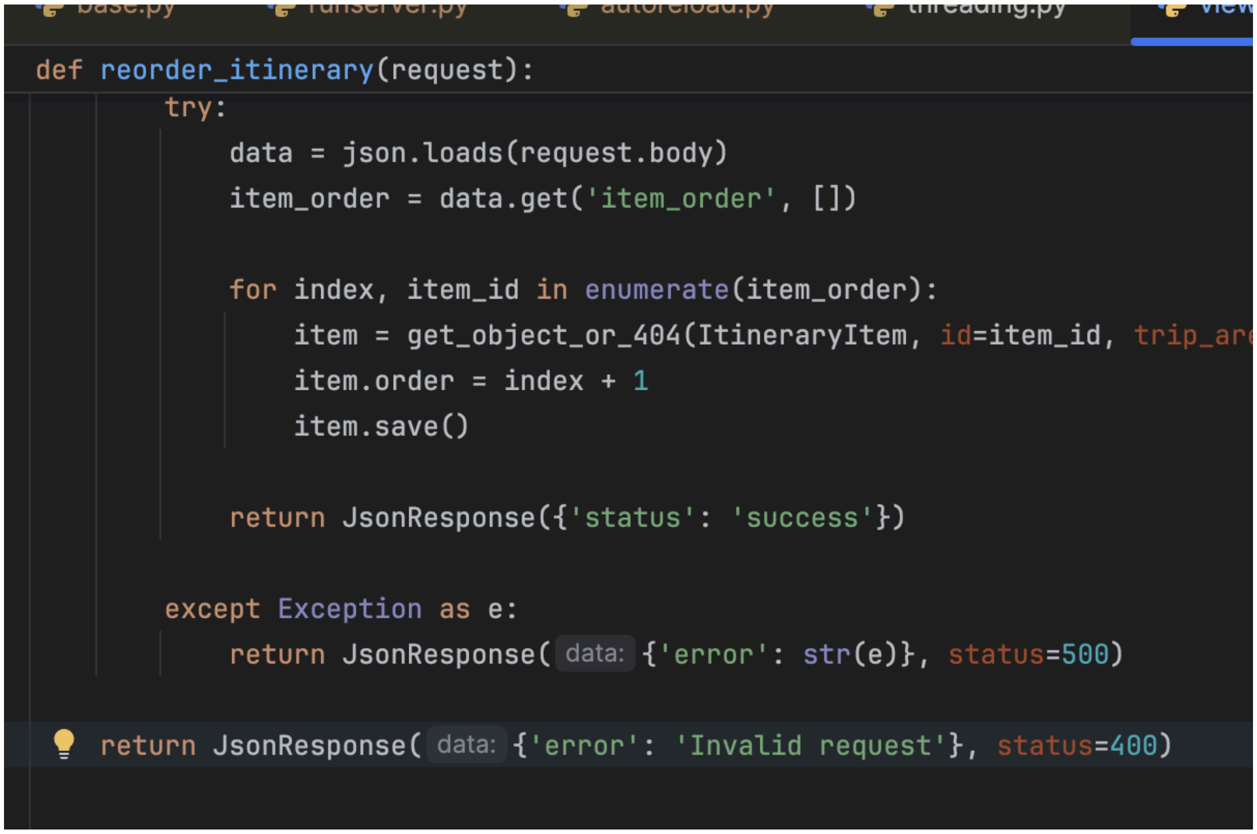


```
@login_required
def map_ui(request):
    api_key = settings.GOOGLE_MAPS_API_KEY
    trip_areas = TripArea.objects.filter(user=request.user)

    if request.method == 'POST':
        # Handle form submission to create new trip area
```

2. Duplicate error handling
    a. Created a handle_error helper method to remove duplicate lines of code and improve maintainability.

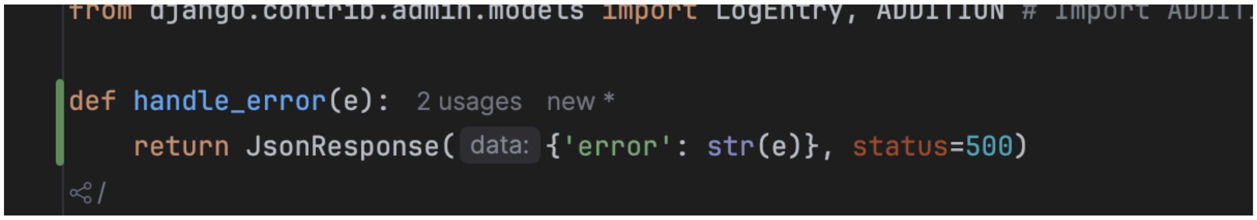


```
def add_to_itinerary(request):
                change_message=f"User added '{item.name}' to itinerary '{tr
            )

            return JsonResponse({'status': 'success', 'item_id': item.id})

        except Exception as e:
            return JsonResponse( data: {'error': str(e)}, status=500)

    return JsonResponse( data: {'error': 'Invalid request'}, status=400)
```

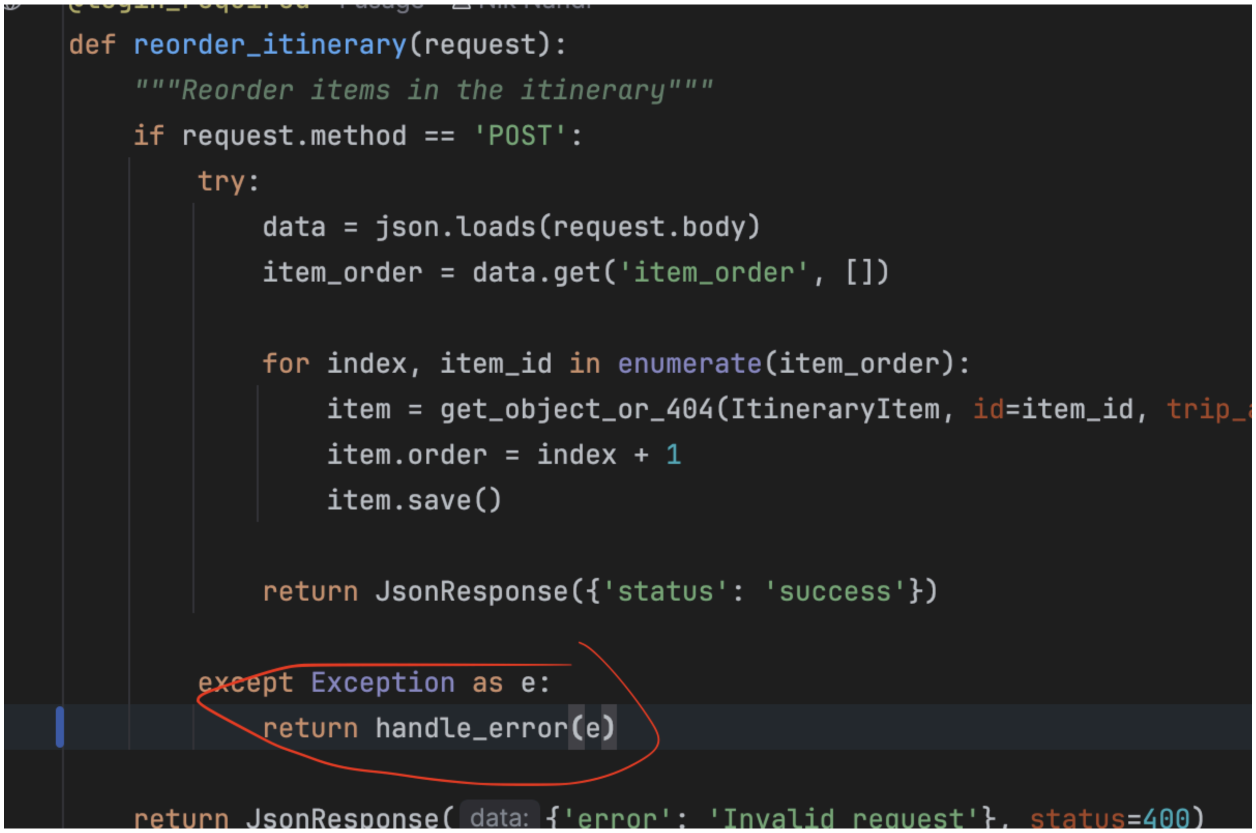


```
def reorder_itinerary(request):
        try:
            data = json.loads(request.body)
            item_order = data.get('item_order', [])

            for index, item_id in enumerate(item_order):
                item = get_object_or_404(ItineraryItem, id=item_id, trip_are
                item.order = index + 1
                item.save()

            return JsonResponse({'status': 'success'})

        except Exception as e:
            return JsonResponse( data: {'error': str(e)}, status=500)

    return JsonResponse( data: {'error': 'Invalid request'}, status=400)
```

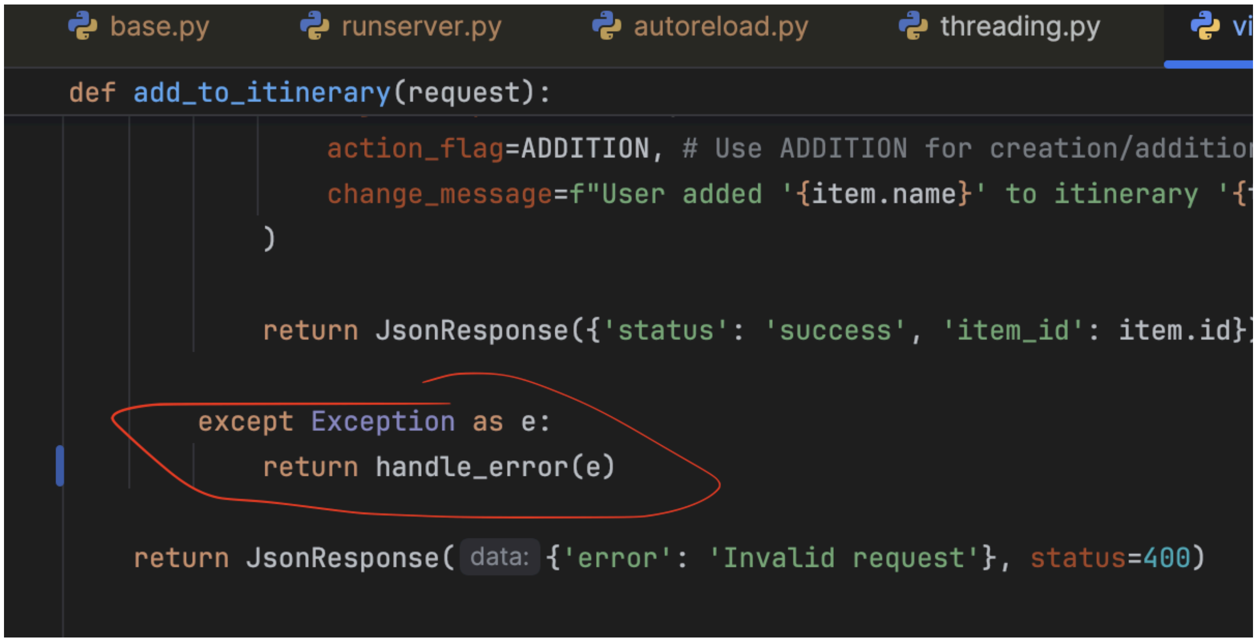
AFTER:



```
from django.contrib.admin.models import LogEntry, ADDITION # import ADDIT

def handle_error(e):  2 usages   new *
    return JsonResponse( data: {'error': str(e)}, status=500)
```



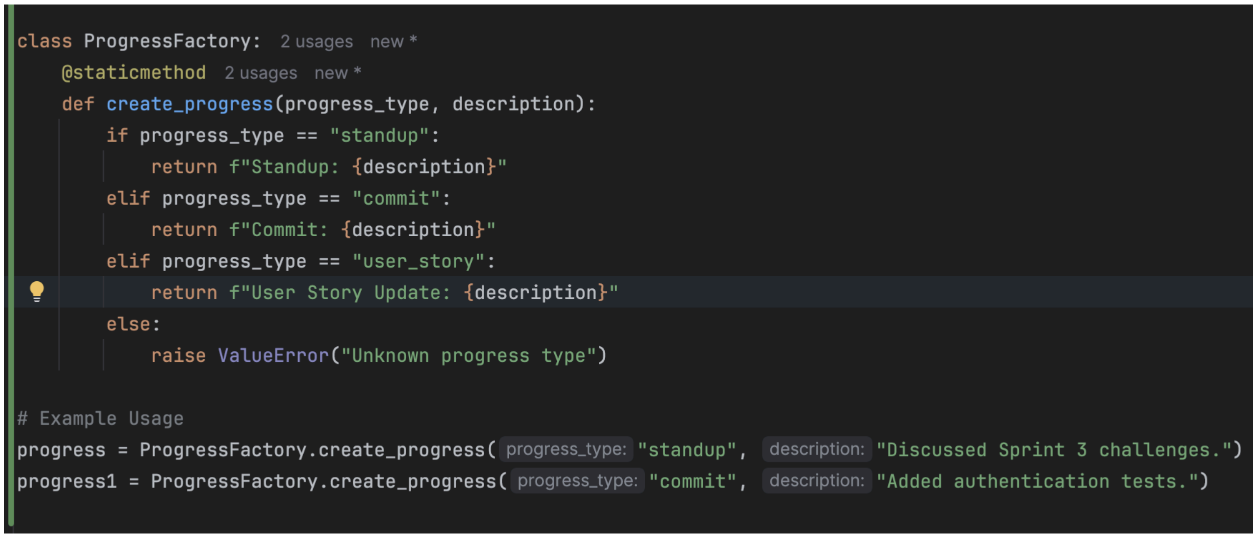
```
def reorder_itinerary(request):
    """Reorder items in the itinerary"""
    if request.method == 'POST':
        try:
            data = json.loads(request.body)
            item_order = data.get('item_order', [])

            for index, item_id in enumerate(item_order):
                item = get_object_or_404(ItineraryItem, id=item_id, trip_
                item.order = index + 1
                item.save()

            return JsonResponse({'status': 'success'})

        except Exception as e:
            return handle_error(e)

    return JsonResponse( data: {'error': 'Invalid request'}, status=400)
```



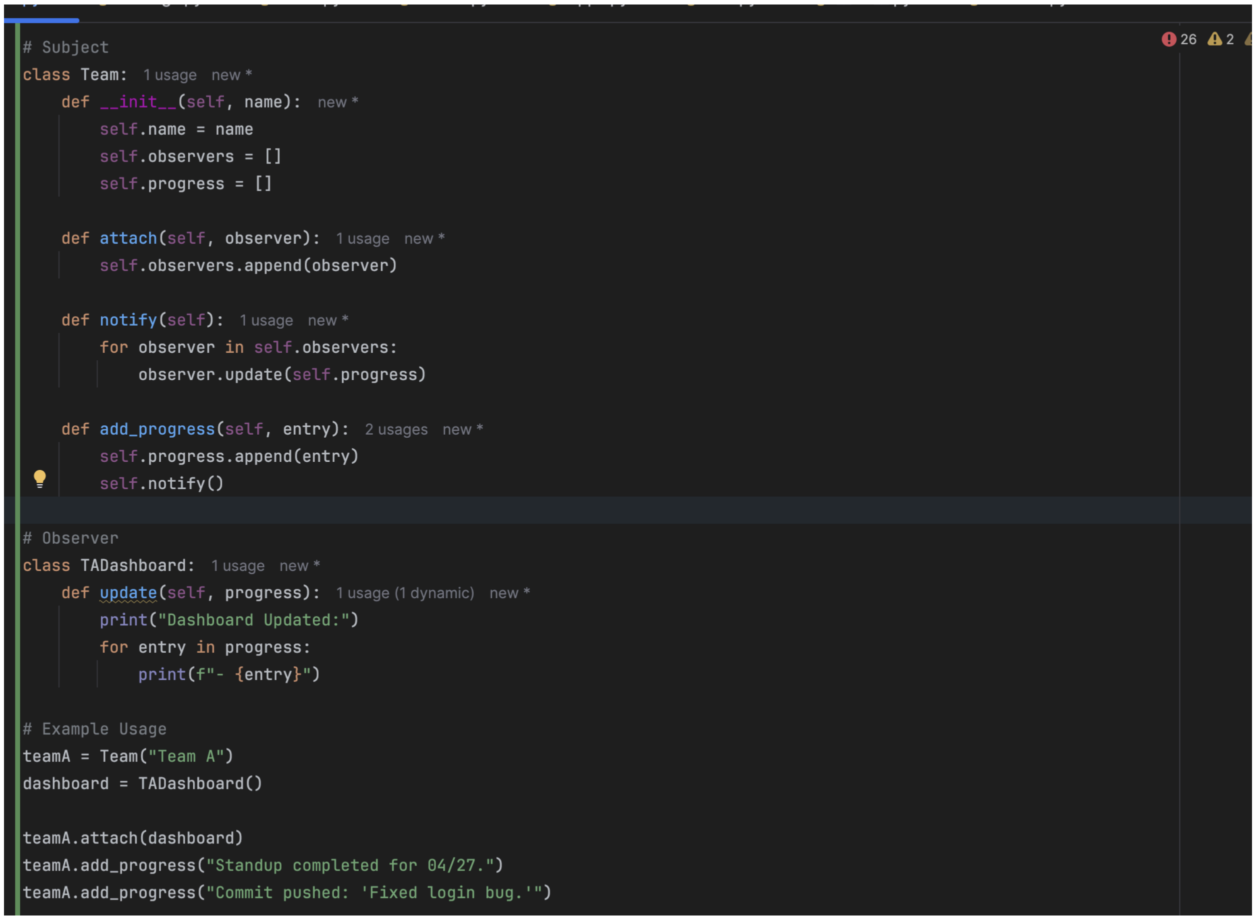
```
def add_to_itinerary(request):
                action_flag=ADDITION, # Use ADDITION for creation/additio
                change_message=f"User added '{item.name}' to itinerary '{
            )

            return JsonResponse({'status': 'success', 'item_id': item.id}

        except Exception as e:
            return handle_error(e)

    return JsonResponse( data: {'error': 'Invalid request'}, status=400)
```

3. You are tasked with creating a software to help CS 2340 TAs keep track of their team's progress (i.e. stand ups, commits, user story progress). Describe how you would implement this app using one or more design patterns and provide short code snippets to support your argument.

We would implement this app using the Observer and factory method patterns. The observer pattern will help with real-time updates by notifying the observer (TAs) when subjects (Teams) post standups, commits, and updates. The Factory Method Pattern will help with creating different progress entries (i.e., standups, commits, updates) without repetition and hard coding.



```python
class ProgressFactory:  2 usages  new *
    @staticmethod  2 usages  new *
    def create_progress(progress_type, description):
        if progress_type == "standup":
            return f"Standup: {description}"
        elif progress_type == "commit":
            return f"Commit: {description}"
        elif progress_type == "user_story":
            return f"User Story Update: {description}"
        else:
            raise ValueError("Unknown progress type")

# Example Usage
progress = ProgressFactory.create_progress( progress_type: "standup",  description: "Discussed Sprint 3 challenges.")
progress1 = ProgressFactory.create_progress( progress_type: "commit",  description: "Added authentication tests.")
```



```python
# Subject
class Team:  1 usage  new *
    def __init__(self, name):  new *
        self.name = name
        self.observers = []
        self.progress = []

    def attach(self, observer):  1 usage  new *
        self.observers.append(observer)

    def notify(self):  1 usage  new *
        for observer in self.observers:
            observer.update(self.progress)

    def add_progress(self, entry):  2 usages  new *
        self.progress.append(entry)
        self.notify()

# Observer
class TADashboard:  1 usage  new *
    def update(self, progress):  1 usage (1 dynamic)  new *
        print("Dashboard Updated:")
        for entry in progress:
            print(f"- {entry}")

# Example Usage
teamA = Team("Team A")
dashboard = TADashboard()

teamA.attach(dashboard)
teamA.add_progress("Standup completed for 04/27.")
teamA.add_progress("Commit pushed: 'Fixed login bug.'")
```

4. What's the most important thing that needs to happen between you and your client? How have you done so in regards to your user stories in project 2?

The most important thing that needs to happen between us and our client is clear communication. With clear communication, we can establish and clarify requirements throughout development, and also gather feedback. We demonstrated this through our user stories by writing detailed user stories describing their purpose and use cases, prioritizing user-centered strategies, and improving features as we build the project and recognize flaws.