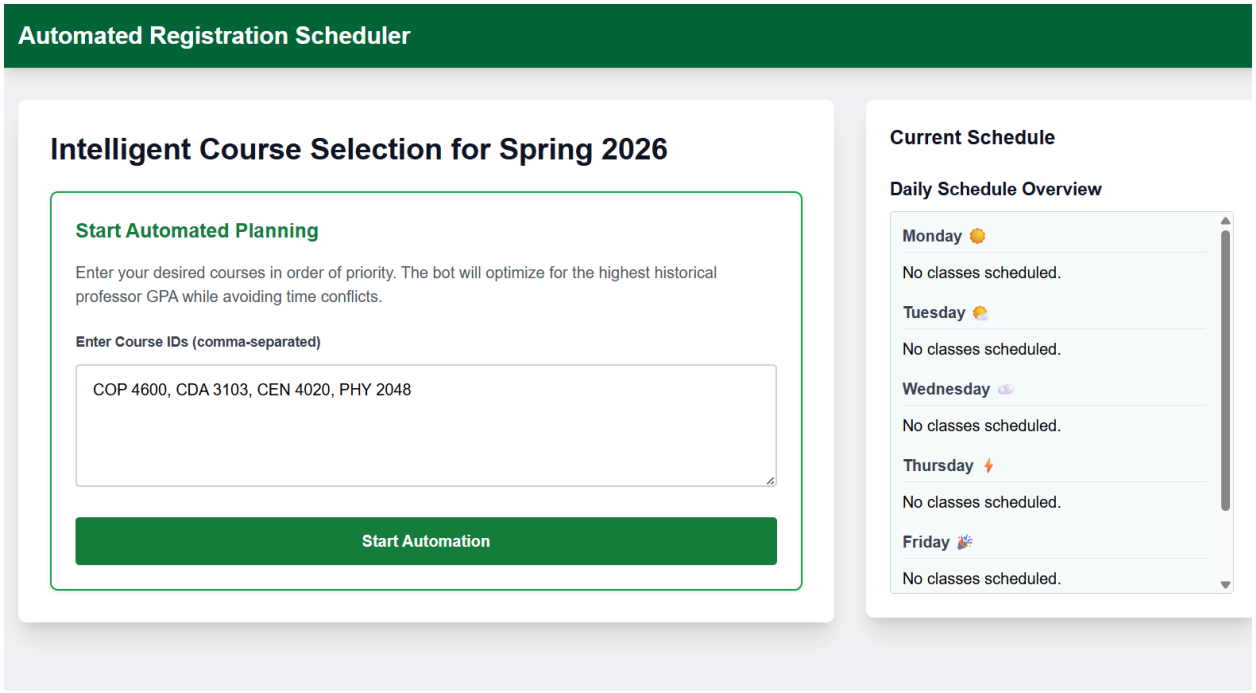
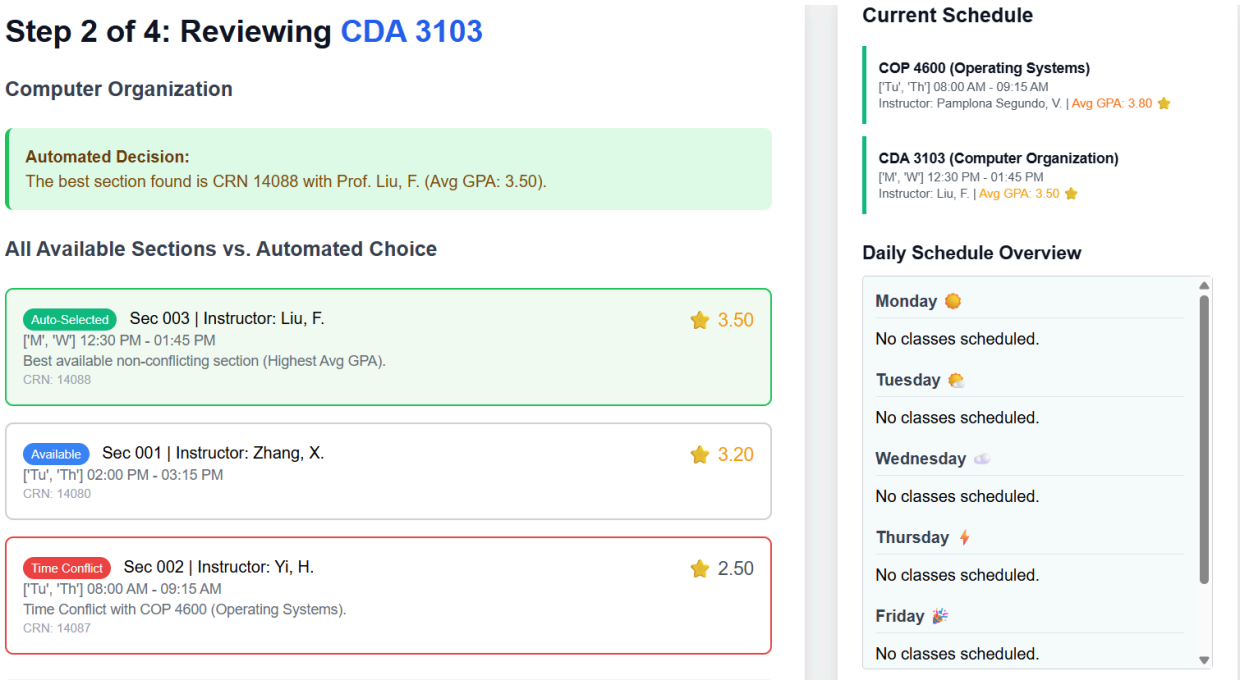
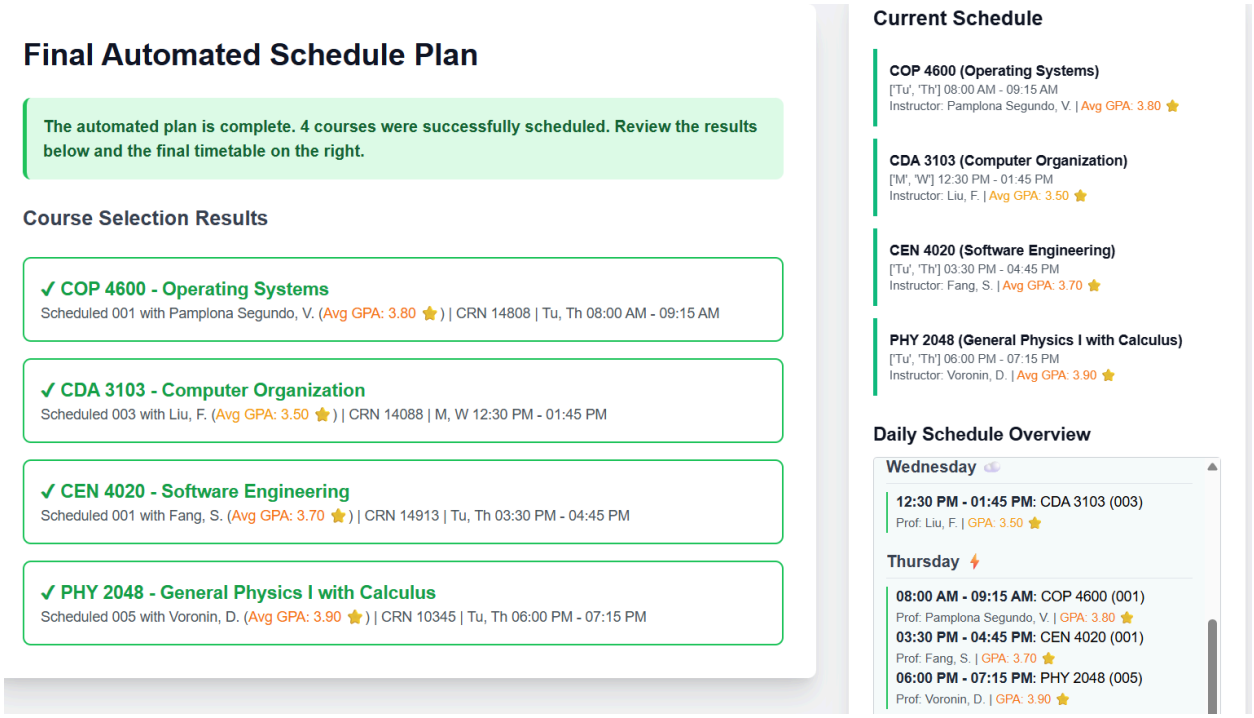
1) User is at the Course Schedule Page and simply enters the the course numbers



2) The program looks at each prof available for the course, compares the prof with the USF infocenter data, selects the best non-conflicting(w/ previous class) prof with highest gpa.



3) It prioritizes the courses entered first, meaning if there is a time conflict between the course entered first and the course entered second, it will select the best prof (by avg gpa from infocenter) and then it will select the best non-conflicting prof from course 2 and this continues until the last course. It finally displays a proposed plan -



Code snippets - to mimic actual registration system

```
COURSE_OFFERINGS = {
    "COP 4600": {
        "title": "Operating Systems",
        "sections": [
            {"crn": "14808", "section": "001", "days": "TuTh", "time": "08:00 AM - 09:15 AM", "instructor": "Pamplona Segundo, V.", "status": "Open", "enrolled":
            {"crn": "14809", "section": "002", "days": "TuTh", "time": "09:30 AM - 10:45 AM", "instructor": "Zhang, Y.", "status": "FULL", "enrolled": 40, "capaci
            {"crn": "14810", "section": "003", "days": "MW", "time": "09:30 AM - 10:45 AM", "instructor": "Karthik, R.", "status": "Open", "enrolled": 30, "capaci
            {"crn": "14811", "section": "004", "days": "MW", "time": "02:00 PM - 03:15 PM", "instructor": "Smith, A.", "status": "Open", "enrolled": 20, "capacity
        ]
    },
    "CDA 3103": {
        "title": "Computer Organization",
        "sections": [
            {"crn": "14080", "section": "001", "days": "TuTh", "time": "02:00 PM - 03:15 PM", "instructor": "Zhang, X.", "status": "Open", "enrolled": 35, "capaci
            {"crn": "14087", "section": "002", "days": "TuTh", "time": "08:00 AM - 09:15 AM", "instructor": "Yi, H.", "status": "Open", "enrolled": 25, "capacity"
            {"crn": "14088", "section": "003", "days": "MW", "time": "12:30 PM - 01:45 PM", "instructor": "Liu, F.", "status": "Open", "enrolled": 28, "capacity"
        ]
    },
    "CEN 4020": {
        "title": "Software Engineering",
        "sections": [
            {"crn": "14913", "section": "001", "days": "TuTh", "time": "03:30 PM - 04:45 PM", "instructor": "Fang, S.", "status": "Open", "enrolled": 30, "capacit
            {"crn": "14914", "section": "002", "days": "MW", "time": "06:00 PM - 07:15 PM", "instructor": "Anderson, J.", "status": "FULL", "enrolled": 40, "capac
        ]
    },
    "PHY 2048": {
```

Build the schedule -

```python
def build_daily_schedule_data(schedule_data, user_courses):
    """
    Organizes scheduled sections into a dictionary grouped by day and sorted by time.
    """
    # Daily schedule keys 1-5 for Mon-Fri
    daily_schedule = {d: [] for d in range(1, 6)}
    online_schedule = []

    for course_id, section in schedule_data.items():
        # Add priority index for reference
        section['priority'] = user_courses.index(course_id) + 1

        if 'Online' in section['days']:
            online_schedule.append(section)
            continue

        # Calculate start minutes for sorting
        section['start_minutes'] = time_to_minutes(section['start_time'])

        for day_abbr in section['days']:
            day_key = DAY_MAP.get(day_abbr)
            if day_key:
                daily_schedule[day_key].append(section)

    # Sort classes within each day by start time (in-place)
    for day_key in daily_schedule:
```

Checks for conflicts -

```python
def check_conflict(new_section, time_slot_map):
    """Checks for time conflicts with already scheduled classes."""
    if 'Online' in new_section['days']:
        return None

    start_min = time_to_minutes(new_section['start_time'])
    end_min = time_to_minutes(new_section['end_time'])

    if start_min is None or end_min is None:
        return None

    for day_abbr in new_section['days']:
        day_key = DAY_MAP.get(day_abbr)

        if day_key and day_key in time_slot_map:
            for existing_class in time_slot_map[day_key]:
                existing_start = time_to_minutes(existing_class['start_time'])
                existing_end = time_to_minutes(existing_class['end_time'])

                # Check for time overlap: (StartA < EndB) AND (StartB < EndA)
                if start_min < existing_end and existing_start < end_min:
                    return existing_class

    return None
```