Space Calendar

System Architecture

EECS 581 - Team 22

Sam (Kiara) Grimsley, Audrey Pan, Ella Nguyen,

Hart Nurnberg, Yuwen (Reeny) Huang, Lauren D'Souza

Synopsis

A Python-based website that imports and exports user calendars, analyzes questionnaire preferences, and suggests optimal time slots for new events.

Architecture Description

Front-End

The front-end of the scheduler program is a web page rendered by Django's templating engine and served by a web server. Django handles the server-side logic and generation of the web page content, which forms the front-end delivered to the user's browser.

The front-end serves as the user interface for collecting preferences, creating events, and importing/exporting calendars. It is built using standard web technologies (HTML, CSS, and minimal JavaScript) and uses local browser storage and cookies to store user data. The front-end provides three main features for the user:

1. User preference entry: the user fills out a questionnaire about their time management preferences, including what time of day they like to study, what time of day they like to do physical activity, and more.
2. Importing and exporting calendars: the user must import their current schedule as a .ics file for the back-end to later use. After their new schedule has been created, they can export it, again as a .ics file.
3. Creating new events: the user can create events they'd like to add to their calendar. The user enters the name of the event, the category (e.g. study, workout, etc.), the length of the event, and an optional set date or time range.

The user interface connects to the back-end using HTTP requests. GET retrieves the new schedule file from the back-end, and POST sends preferences and new events to the back-end.

Back-End

The back-end of the scheduler program is implemented using Django. Since there is no database, the back-end mainly serves as the logic engine of the program. All info is passed to the back-end as JSON payloads from the front-end. The back-end serves three main purposes:

1. Input processing: after Django handles the receiving of the user's existing calendar through a POST request, the back-end Python program parses the file and converts it into data for the program to use. It also parses the user preferences into usable data, also received through a post request.
2. Scheduling logic: the program scans for open time slots from the user's existing calendar. It then applies a simple rule-based algorithm to apply the user's preferences to the

schedule. Some example constraints it may apply include not scheduling study sessions in between classes and scheduling workouts for before 9pm.

3. Calendar generation and exporting: the program uses the calendar module, ics, to generate the new calendar. It puts all existing and new events into the calendar, which is then sent to the front-end through a POST request.

User preferences are stored using persistent cookies. No other data is stored persistently, hence no need for a database.

Data Flow

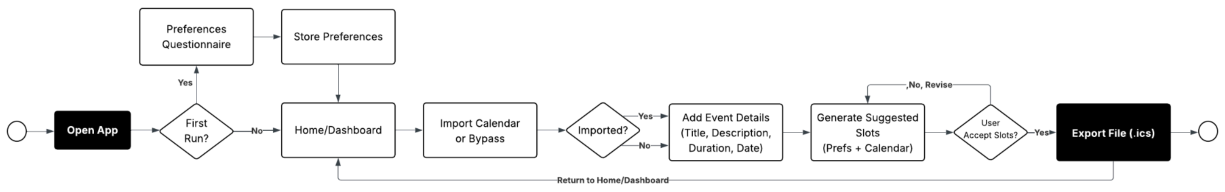

Preferences Questionnaire
Store Preferences
Yes
Open App
First Run?
No
Home/Dashboard
Import Calendar or Bypass
Imported?
Yes
No
Add Event Details (Title, Description, Duration, Date)
No, Revise
Generate Suggested Slots (Prefs + Calendar)
User Accept Slots?
Yes
Export File (.ics)
Return to Home/Dashboard

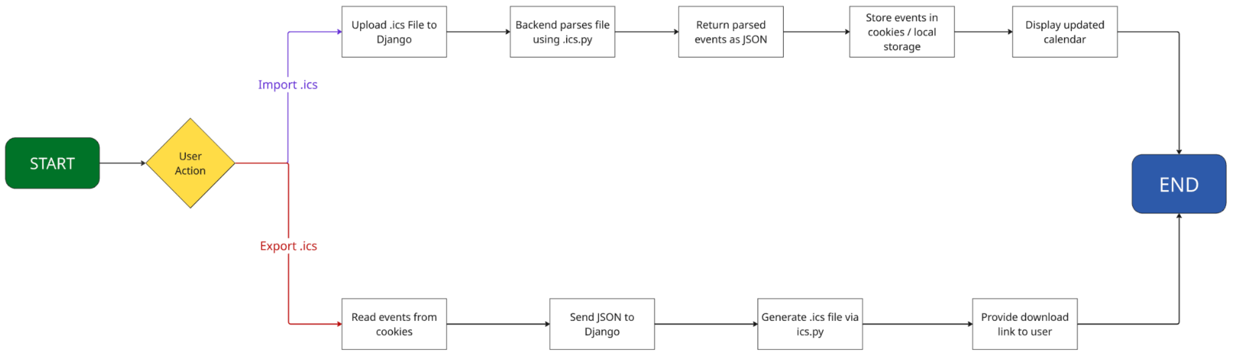
Figure 1. MVP UI Flow UML Diagram

When the user first opens the app, the system checks whether they have existing preferences stored. If not, they are guided through a preferences questionnaire to specify their available days, preferred working hours, and buffer times. After setup, users arrive at the Home/Dashboard, where they can choose to import an existing calendar file or skip that step to start fresh.

Once a calendar is imported, the system parses the .ics file and identifies all busy blocks. The user can then add new events through a simple input form specifying details such as the event title, duration, and optional date or time range. Based on the imported calendar and the stored preferences, the algorithm searches for blank spaces that can accommodate the new event. The MVP uses a basic first-fit, prioritizing any available slot that satisfies the required duration and time constraints.

The user is presented with a list of suggested time slots and can accept or reject them. If they choose to modify their inputs, the system reruns using the updated parameters. Once satisfied, the user can export the new schedule as a combined .ics file, which merges both their original calendar events and any newly generated ones. This file can then be re-imported into their preferred calendar client, such as Google Calendar or Apple Calendar.

Import/Export .ics File



START
User Action
Import .ics
Upload .ics File to Django
Backend parses file using .ics.py
Return parsed events as JSON
Store events in cookies / local storage
Display updated calendar
Export .ics
Read events from cookies
Send JSON to Django
Generate .ics file via ics.py
Provide download link to user
END

Figure 2. Import/Export .ics File Workflow

The import/export workflow allows users to move calendar data between their local files and the web app without relying on a database. When the user chooses to import .ics option, they upload a calendar file that is sent to the Django backend. Django processes the file using the ics.py Python library, which reads the standard iCalendar (.ics) format and extracts event details such as the title, date, start and end times, and duration. The parsed events are converted into JSON and returned to the browser. The front end then stores this event data in persistent cookies or local storage. This ensures that it is available even if the page is refreshed or reopened later. After storing the information, the interface updates to show the user's imported events on the calendar view.

For the Export .ics path, the browser reads locally stored events and sends them back to Django as a JSON payload through a POST request. The backend will use the ics.py library, but this time, it generates a new .ics file that includes all user-created or modified events. Once the file is created, Django provides a downloadable link so the user can save or re-import into their personal calendar application (e.g., Google Calendar or Apple Calendar).

Website State Description

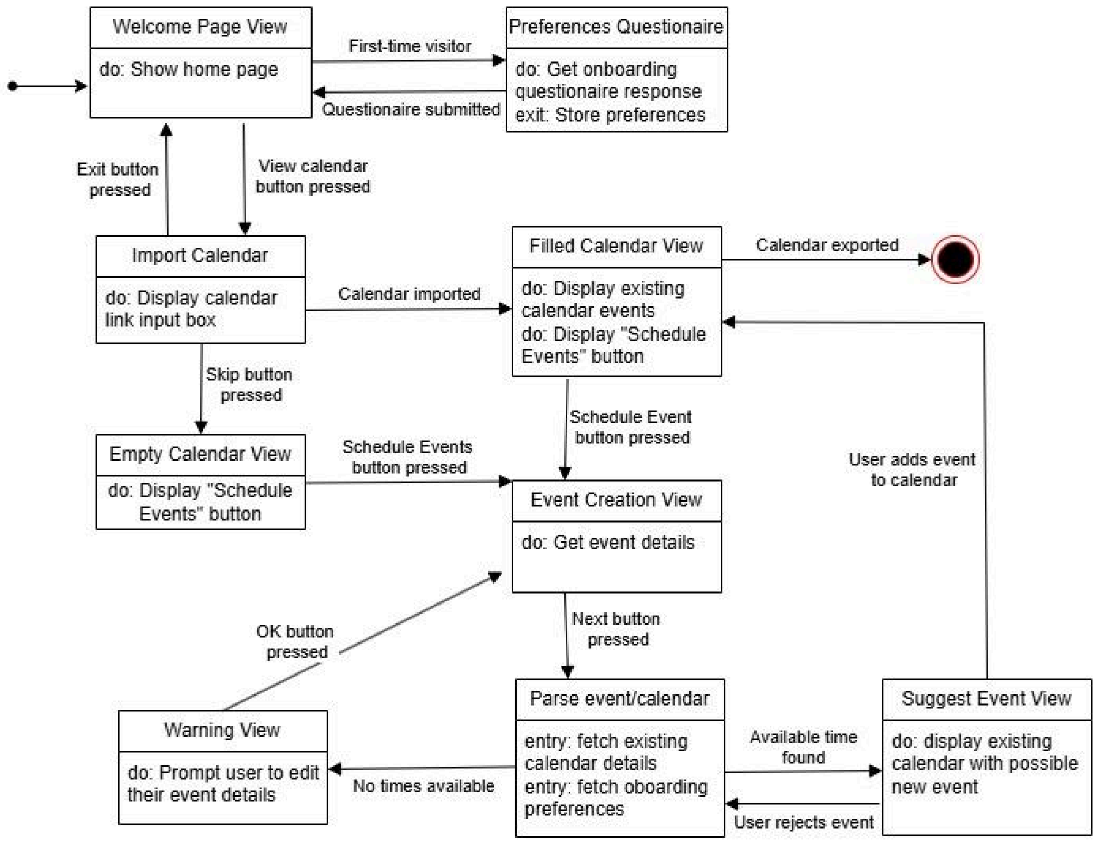

Welcome Page View
do: Show home page
First-time visitor
Questionaire submitted
Preferences Questionaire
do: Get onboarding questionaire response
exit: Store preferences
Exit button pressed
View calendar button pressed
Import Calendar
do: Display calendar link input box
Calendar imported
Filled Calendar View
do: Display existing calendar events
do: Display "Schedule Events" button
Calendar exported
Skip button pressed
Empty Calendar View
do: Display "Schedule Events" button
Schedule Events button pressed
Schedule Event button pressed
User adds event to calendar
Event Creation View
do: Get event details
OK button pressed
Next button pressed
Warning View
do: Prompt user to edit their event details
No times available
Parse event/calendar
entry: fetch existing calendar details
entry: fetch oboarding preferences
Available time found
User rejects event
Suggest Event View
do: display existing calendar with possible new event

Figure 3. State Diagram of Website Front-End

The state diagram describes how the website changes as the user interacts with it to view their calendar and schedule events. The base state of the website is the “Home” page, which is where users will automatically be directed. If this is our first recorded instance of the user visiting the page, we then display the onboarding questionnaire to obtain their general preferences, as described in the Data Flow section.

Once the user wants to view their calendar, they can press a “View Calendar” button to import their calendar. This import process is described in Figure 2, however it is an optional step. Depending on if the user imports their calendar, they will either be redirected to an Empty Calendar View state or a Filled Calendar View state. Both of these states will allow the user to press a “Schedule Events” button, however only the Filled Calendar View state will allow the user to export the calendar as an .ics file.

Upon selection of the “Schedule Events” button, the user will be prompted to enter more details about the event, such as the duration or requested days and frequency, in the Event Creation state. These details will then be compared against the user’s existing calendar and preferences. If no event can be created, then a warning saying as much is displayed in the Warning View state. Then, the user moves back to the Event Creation state to edit their details. Otherwise, an event time is displayed in the Suggest Event state where the user can accept or reject the event, as described in the Data Flow section. If rejected, the website will re-parse the calendar availability with the events details to suggest a new event, as described in the Data Flow section. If accepted, the event is added to the user's calendar and the user moves back to the Filled Calendar View state where the calendar can be exported (refer to Figure 2) or new events can be added.