1. The “Remind Me” application that I plan on creating will be a calendar and time-management application. Users will have full CRUD functionality for events on personal calendars that they can access through their accounts. Users will also be able to set SMS reminders for these events to be sent out at predetermined times.
2. I imagine that while anyone who is able to effectively use a basic computer application can use “Remind Me”—there’s nothing child-unfriendly or elder-unfriendly about the app—the general demographic using the app would be between around 14-65 years old. By high school, most teenagers can effectively use a basic computer app, have cell phones, and would have some responsibility for tracking their own schedules as students. After retirement, adults often have fewer events to track on their schedules. Also, current retirement-age adults may be more uncomfortable using a digital planner compared to their younger counterparts, though this is by no means universal and will change as generations that grew up around computers grow older.
3. I plan on using a calendar API of some kind to show and help effectively store calendar events (both user-generated and general events like widely-observed holidays). I also plan on using a text-message API such as Twilio to handle setting and sending reminders for both user-generated and general events.
4. To create my project, I will need:
    - 4.1. A basic Flask app with a Postgres database and SQLAlchemy
    - 4.2. Models:
        - 4.2.1. A User model (id, username, email, cell phone number, password). At the very least, the cell phone numbers and passwords of users will need to be kept secure. The User model will be associated with the Calendar model in some way—it will be one-to-one for this application, though.
        - 4.2.2. A Calendar model (id, user_id (foreign key to User)). The Calendar model will have a direct one-to-many relationship to the Event model: events = db.relationship(‘Event’, backref=’calendar’)
        - 4.2.3. An Event model (id, calendar_id (foreign key to Calendar), date, time, recur_frequency, has_reminder). If the event does not recur, its recur_frequency will be set to None. If the event has no reminders, its has_reminder attribute will be set to False.
        - 4.2.4. A Reminder model (id, event_id (foreign key to Event), message, time_before_event). If possible, reminders will not be hard-coded to a specific date and time, but will be based off the date and time of their (recurring or one-time) associated event.

4.3. Forms:
   4.3.1. AddUserForm (username, email, cell phone number, password): a form to register a new user. A new Calendar model will be generated upon creation of a new user and associated with that user.
   4.3.2. EditUserForm (username, email, cell phone number): a form to most of an existing user’s details, but not their calendar or password.
   4.3.3. LoginUserForm(username, password): a form to log in a user.
   4.3.4. AddEventForm (calendar_id, date, time, recur_frequency, has_reminder=False): A form to add a new event to an existing user’s calendar. For the sake of simplicity/my sanity, a user cannot add a reminder while creating an event.
   4.3.5. EditEventForm: (date, time, recur_frequency) A form to edit an existing event in an existing user’s calendar.
   4.3.6. AddReminderForm: (message, time_before_event) A form to add a SMS reminder to an existing event in an existing user’s calendar. The event id will be supplied from the URL.
   4.3.7. EditReminderForm (message, time_before_event): A form to edit an SMS reminder to an existing event in an existing user’s calendar. The event id will be supplied from the URL.

4.4. Authentication and authorization: a user should only be able to have CRUD functionality and reminder functionality for their own account, which will be associated with their calendar exclusively.

4.5. A calendar API to assist in storing, retrieving, and displaying calendar data; I’ll have to look into my options to see how exactly it will work.

4.6. A SMS API (looking at Twilio) to assist in sending reminder text messages.

4.7. Routes:
   4.7.1. Basic and Authentication:
      4.7.1.1. @app.route(‘/’): The home page of the app. Will return a redirect to (‘/users/<user_id>/events) if a user is logged in. Otherwise, will return a template welcoming the user to the site and containing links to ‘/register’ or ‘/login’.
      4.7.1.2. @app.route(‘/register’, methods=[“GET”, “POST”]): Will redirect to ‘/users/<int:user_id>’ if a user is already logged in. If not:

4.7.1.2.1. On a GET request, will display an AddUserForm allowing a user to register an account for the site.

4.7.1.2.2. On a POST request, will attempt to register a user to the site with appropriate error handling.

4.7.1.3. @app.route('/login', methods=["GET", "POST"]): Will redirect to '/users/<int:user_id>' if a user is already logged in. If not:

4.7.1.3.1. On a GET request, will display a LoginUser form which will allow a user to log in.

4.7.1.3.2. On a POST request, will attempt to log a user in with appropriate error handling.

4.7.1.4. @app.route('/logout'): If a user is not logged in, will redirect to '/login' with appropriate flashed messages. If a user is logged in, logs the user out and redirects to '/'.

4.7.2. User Management:

4.7.2.1. @app.route('/users/<int:user_id>', methods=["GET", "POST"]): If no user or the wrong user is logged in, will redirect to '/login' with appropriate flashed messages. If not:

4.7.2.1.1. On a GET request where the current user id matches the user id in the URL, returns a template with a user's details (but not their calendar), an EditUserForm for the user, a link to '/events' for the user's calendar, and a button to '/users/<int:user_id>/delete' to delete the user.

4.7.2.1.2. On a POST request where the user is authorized, updates user details with the data from the EditUserForm.

4.7.2.2. @app.route('/users/<int:user_id>/delete', methods=["POST"]): If no user or the wrong user is logged in, will redirect to '/login' with appropriate flashed messages. If not, logs the user out, attempts to delete the user in the URL from the database and redirects to '/'.

4.7.3. Event Management:

4.7.3.1. @app.route('/events', methods=["GET", "POST"]): If no user is logged in, redirects to '/login' with appropriate flashed messages. If a user is logged in:

4.7.3.1.1. On a GET request, shows the user's calendar, hopefully with an interface that looks like a calendar. This calendar should show all events for a range of dates. Also shows an AddEventForm. Each event in the calendar will link to '/events/<int:event_id>' and will have a button that makes a POST request to '/events/<int:event_id>/delete'.

4.7.3.1.2. On a POST request, attempts to add a new event using the data in the AddEventForm with appropriate error handling.

4.7.3.2. @app.route('/events/<int:event_id>', methods=["GET", "POST"]): If no user is logged in or the wrong user for the event is logged in, redirects to '/login' with appropriate flashed messages. If a user is logged in:

4.7.3.2.1. On a GET request, shows details for the event with its id in the URL. Also shows an EditEventForm, as well as another button that makes a POST request to 'events/<int:event_id>/delete', a list of reminders on that event with a button to '/events/<int:event_id>/reminders/<int:reminder_id>/delete' for each one, and a AddReminderForm to add new reminders ('/events/<int:event_id>/reminders', methods=["POST"])

4.7.3.2.2. On a POST request, attempts to update the relevant event using data from the EditEventForm, then redirects to '/events'.

4.7.3.3. @app.route('/events/<int:event_id>/delete', methods=["POST"]): If no user is logged in or the wrong user for the event is logged in, redirects to '/login' with appropriate flashed messages. If a user is logged in, attempts to delete the event with its id in the URL, then redirects to '/events'.

4.7.4. Reminder Management

4.7.4.1. @app.route('/events/<int:event_id>/reminders', methods=["POST"]): If no user is logged in or the wrong user for the event is logged in, redirects to 'login' with appropriate flashed messages. If the correct user is logged in, attempts to add a new reminder to an event and redirects to '/events/<int:event_id>'

4.7.4.2. @app.route('/events/<int:event_id>/reminders/<int:reminder_id>', methods=["GET", "POST"]): If no user is logged in or the wrong user for the event is logged in, redirects to '/login' with appropriate flashed messages.

If the correct user is logged in:

4.7.4.2.1. On a GET request, shows an individual reminder for an event along with an EditReminderForm, and a button to delete the reminder ('/events/<int:event_id>/reminders/<int:reminder_id>/delete').

4.7.4.2.2. On a POST request, attempts to edit the reminder in the URL and redirets to '/events<int:event_id>'.

4.7.4.3. @app.route('events/int:event_id/reminders/<int:reminder_id>/delete', methods=["POST"]): If no user is logged in or the wrong user for the event is logged in, redirects to "/login" with appropriate flashed messages. If the correct user is logged in, attempts to delete a reminder and redirects to '/events/<int:event_id>'