Team Name: Moon Jelly
Navid Shamszadeh, Qi Guo, Aidan Glina, Jingjing Yu
Emails: aidanglina@ucsb.edu qi_guo@ucsb.edu nshamszadeh@ucsb.edu jingjingyu@ucsb.edu
Link to Project Demo: https://www.youtube.com/watch?v=hPkrs3oFK8s&feature=youtu.be
Github: https://github.com/nshamszadeh/moon-jelly,
TravisCI: https://travis-ci.org/nshamszadeh/moon-jelly
Slack: https://moon-jelly.slack.com/messages

Jelly Plan

Vision Statement

The goal of this project is to create a personalized scheduler web app for groups of physicians. There are several different ways that physicians can be employed. One such way involves multiple physicians of one specialty coming together to form a “group” who contract their services out to hospitals/surgery centers. This method of organization often involves different physicians filling different positions at different hospitals on different days. Scheduling which doctors do what position at which hospital is tricky and time consuming. Current scheduling techniques involve inputting every doctor into a spreadsheet by hand or using general scheduling programs that are difficult to customize for different individual groups. The first method is time consuming for busy physicians and the second method is not optimized to the specific needs of the group.

Moon Jelly scheduler aims to be a solution to this issue by being an easy to use, time efficient way to schedule physicians optimally. Groups will be able to customize the number of doctors they have working each day and the number of positions that need to be filled at different facilities. An algorithm will take into account the stressfulness of each position and create an optimized schedule that spaces out stressful work positions while giving every member of the group roughly the same amount of every position. This results in a better working environment for the group while ensuring equal participation. The algorithm will also account for the needs of specific doctors. For instance, the schedule will consider if certain doctors don’t work at certain hospitals or if certain doctors specialize in certain procedures. These cases can change on a doctor by doctor basis. Through this app doctors will be able to request for certain days to be shorter work days or days off. We will also prioritize clean and intuitive user interface that can be easily used by doctors that are not so tech savvy.

This project will result in a web app that can be used by real physicians. A group of physicians based in Santa Cruz CA has already agreed to test the project. We will work with them to create a product that fulfills their needs in a real world environment. This will result in
their group of physicians getting a customized scheduling system while our project group learns about development cycles in a real customer/developer relationship.

We plan to implement this scheduler through a web app hosted by Heroku. We will utilize the built in database that comes with Heroku web apps to store physician data safely. We will build the app primarily with python-flask and access the database with sqlalchemy. Our first goal is to learn the basics of web development and set up a basic app with the ability to access a database (Sprint 1). From there we will move on to creating a basic scheduling algorithm and creating an intuitive user interface (Sprint 2). After this we will optimize the scheduling algorithm and perfect the account system allowing physician group members to create requests (Sprint 3).

**User Stories:**

*each link is to a user story, each user story has its own column with many tasks below it*

**User story 1:** As a team member, I will be able to use and contribute to making a web app through heroku

**Link 1:** https://trello.com/c/dJx15bye

**User story 2:** as a developer, I can interact with the data base, (add/ delete, etc)  

**Link 2:** https://trello.com/c/rFEODNn7

**User story 3:** As a registered user, I am able to log in with my information

**Link 3:** https://trello.com/c/2Ln4YrT4

**User story 4:** As an administrator, I should be able to add and delete new members

**Link 4:** https://trello.com/c/dJx15bye

**User story 5:** As an administrator, after I have input the necessary data, I can log in and check a formatted schedule and check added members.

**Link 5:** https://trello.com/c/yiJ0DC2l

**User story 6:** A user can view a schedule that fills variable number of spots every day

**Link 6:** https://trello.com/c/yDSPTwJ0

**User story 7:** A user can get a schedule that includes special cases (like a certain specialist always in the top 3 spots, certain people not working in certain spots and the more days between the same shift the better)

**Link 7:** https://trello.com/c/ctRpmWcg
**User story 8:** As a user, the user interface should be intuitive and compelling to me as well as easy to understand

**Link 8:** [https://trello.com/c/r0Bd3xxW](https://trello.com/c/r0Bd3xxW)

**User story 9:** as an administrator, I can input the number of people i want working each day, and can input which people I want working on each day

**Link 8:** [https://trello.com/c/3GdEU2QN](https://trello.com/c/3GdEU2QN)

**User story 10:** As a user, after I can log in to the schedule, I can export my schedule as a CSV file.

**Link 10:** [https://trello.com/c/JDHvFKeX](https://trello.com/c/JDHvFKeX)

**User story 11a:** As a user, I can directly message the administrators.

**User story 11b:** As an administrator, I can easily see the messages from users and delete them. (these were grouped together as they both pertain to message functionality)

**Link 11:** [https://trello.com/c/JuqX4tQX](https://trello.com/c/JuqX4tQX)

**Challenges:**

Over our ten week development cycle our group encountered several challenges that impeded our progress. One of our group members dropped the course midway through sprint one. This was our greatest challenge and made us re-evaluate the goals of the sprint as well as our longer term intentions with the project. We concluded that with a group of four we should focus on having fewer, more robust user stories rather than many features with weaker functionality. This manifested itself with our group focusing on the scheduling input/algorithm and making a strong user framework. By building a strong basis of our project we were able to incorporate more functionality by making a very flexible scheduling system by integrating the user class and the scheduling input and algorithm in different ways.

Learning the basics of web development was much slower than we thought. We had to pick a framework, database and cloud platform to host the web app. Our group lost some time exploring spring api, a java framework we decided was not right for our project. This started us back at square one with python flask which we ultimately used. We were not able to immediately start working on our project as we needed to learn the basics of these technologies. This combined with losing a group member and the coursework of other classes made our progress in sprint 1 and the beginning of sprint 2 slow. However during sprint 2 our group started to become more adept at and comfortable with the various aspects of the project we were focusing on and real progress started to be made.
Schedule Creation Sequence Diagram
Github Commits:
Left out some inconsequential ones

AidanGlina (Aidan):
Create blank
Create travis_ci.travis.yml
test grid
added an html link
Update app.py
commit to deletebutton\testbranch pls
Update requirements.txt
Update _navbar.html
dynamic files
added None to globals
sunday works with the database
displays users all days
everything's in a database now
fixes index error on schedule page if haven't gone thru make
a working add function for testing purposes
handling the edge case for schedule with no users in database
added a basic 'day' class, set it up for qi to start the algorithm
added 2 more weeks
adding some sorting functionality
makes 3 weeks. edge cases :
        debug with prints
added export to csv
can send messages to admins
messages work
Merge pull request #41 from nshamszadeh/newmessageAdmin
making it hard for the user to go wrong
Merge pull request #44 from nshamszadeh/useredgecases

nshamszadeh (Navid):
Initial commit
Create LICENSE
Update LICENSE
Update README.md
On branch navbranch
Update test_sample.py
On branch navbranch
Merge pull request #8 from nshamszadeh/navbranch
Update app.py
On branch navbranch
On branch navbranch
Changes to be committed
On branch navbranch
Merge pull request #22 from nshamszadeh/navbranch
On branch navbranch
Merge pull request #23 from nshamszadeh/navbranch
Update app.py
Changes to be committed
Changes to be committed
Your branch is up to date with 'origin/master'
Merge pull request #38 from nshamszadeh/newnavbranch
Merge pull request #40 from nshamszadeh/csvout

lucqiiiii (Qi):
Add files via upload
Merge master.
Remove Python temp files.
Merge pull request #1 from nshamszadeh/about-page
Update app.py
styling
linking
done
Migration for login DB changes.  
Oops. Another migration...
Merge pull request #11 from nshamszadeh/grid
bit update
another migration
Merge pull request #12 from nshamszadeh/lastcall
schedule
Merge pull request #14 from nshamszadeh/dynamic2
some migration
trying the alg
done
Merge branch 'master' of https://github.com/nshamszadeh/moon-jelly in...
add some stuff
slider
sorting
conflict resolve
Merge pull request #28 from nshamszadeh/final
styling the table
migration!!!!
beautifully done
Merge pull request #34 from nshamszadeh/endgame
migration

jingjingyu123 (Jenny):
DATABASE_URL
migration
schedule
login
contact
Merge pull request #5 from nshamszadeh/jennyjennyfish
Merge pull request #16 from nshamszadeh/jennyfish
Update requirements.txt
Update app.py
Update requirements.txt
Merge pull request #21 from nshamszadeh/jennyfish3
Update requirements.txt
Merge branch 'jennyfish3' of https://github.com/nshamszadeh/moon-jelly ...
C++
Merge pull request #24 from nshamszadeh/jennyfish3
Update app.py
Merge pull request #29 from nshamszadeh/jennyfish7
Update _logged_navbar.html
Merge pull request #39 from nshamszadeh/jennyfish8
Update test_sample.py
Update requirements.txt
Update .travis.yml
Update test_sample.py
Update app.py
Sprint Retrospective:

Sprint 1:
- **When the user logs in with the wrong email or password, nothing showed up:** when the user logs in the wrong way, an error message shows up
- **The webpage looks very ugly:** we use css to beautify our webpage and upload beautiful images
- **Let users have their own password:** we set up a new function: when the administrator adds a user, an email would be sent to the user and he or she can change the password, and only the user knows his or her own password
- **We want to remove some users that the hospital doesn't need from the database:** we create a “delete” function. With this function the administrators can delete the users that they don’t need by typing the user’s email.
- **Sign up limitation:** We want only one administrator for a hospital. And the administrator can add doctors. So we only allow for one person—the administrator—to sign up. Other doctors can only log in their account.

Sprint 2:
- **Website styling:** We completely restructure our website layout, add in features such as slider, jumbotron, footer, and contact so that it is a functional website.
- **Ways to upload different versions of code:** We now write and test our code on different branches, push and merge them after we resolve conflicts and agree on the changes.
- **Administrators and other doctors have different privileges:** only administrators should be allowed to add or remove users and create schedules. Regular users can only view the schedule. Administrators can elevate other users to have administrator privileges now.
- **Schedule sorting algorithm:** We restructure our data layout to optimize our algorithm as well as implement dozens of helper functions to follow the actual scheduling rules.

Sprint 3:
- **Website navbar layout:** We separate and utilize the navbar for different role of the doctors so that it’s easier to manage and use our website. Admins and regular users have different navbars showing different links.
- **Password Reset:** we create a “forgot password” function. When the user forgets his or her password, he or she can click “forgot password,” and enter their email. The user would then receive an email with a link to a password reset page.
- **Schedule Output:** We add in the ability to make and display three week long schedules at a time so that the schedule manager can save more time by scheduling less often.
- **Schedule Styling:** We separate the first three calls, post calls and regular calls with different colors to show off different priorities of each position.
- **Admin Messages:** Users can send messages to administrators. Administrators can send and view messages. This allows for users to discuss schedule changes or conflicts directly on the platform.
- **CSV Export:** After a schedule is made, users can choose to have the schedule emailed to them as a CSV in case they want to view or edit it in a spreadsheet software like Excel.
- **Scheduling Edge Cases:** Various edge cases such as inputting only one doctor or not inputting any cardio specialists are accounted for so that the website does not throw an internal server error. These edge cases do not reflect anything a real physician group would input, but making sure the website is fully functional is still important.

**Tests:**

**Selenium IDE & Travis CI Test**

In the project, we mostly used Selenium IDE to test our web app functionality. We included Travis CI Tests to test the database functionality.

**How to use Selenium IDE**

1. Download a selenium IDE from firefox
2. Open it from the website(firefox)
3. Click "create new project" to create a new project to test the app
4. Enter the URL of the project
5. Record the website (click the button on the website or enter some information)
6. Save the project
7. Open the project and click the right button to play test
8. It will automatically go to that URL play the test
9. If the test is wrong, the step would become red. And a red warning will show up saying "tests failed." If the test is right, it will turn green and shows “test completed successfully”

**Selenium IDE Tests**

Log_In_Test(1-3): Log in with the right account: go to “create schedule” page
- Log in with the wrong account: show “invalid account” message
- Log in with the wrong password: show “invalid password” message
Sign_Up_Test1: users can enter email and password to sign up
Sign_Up_Test2: if users enter the wrong email, they cannot sign up

Add_User_Test1: Add one user and the “users page” would show the user successfully.
Add_User_Test2: Add one user that has the same email as the existing user, there would be a message showing the user cannot be added.

CreateSchedule_Test1: Click “Create Schedule” and insert the # of working days for each doctor. Then insert the name of doctors, after that we would see the
CreateSchedule_Test2: If no people working on a specific day, an error message would pop up.
CreateSchedule_Test3: If enter the same name of doctors in one day, and error message would pop up

SeeSchedule_Test1: After log in, both the administrators and other doctors can see the schedule.

SeeAboutPage: Users can see the “about page” shown on the web page.

SeeTeam_Test: Jump to the team member section, which is for introducing team

Home_Test: After clicking it, users from any page can return to home

SeeUsers_Test: When clicking this button, we are able to see our users(only administrator can do this)

ForgotPassword_Test: When the user forgot the password, he can click “forgot password” and he will go to another page
Travis CI Test
Travis CI test is written mainly for testing databases and functions not involved with the flask web framework. The first test we run is when an instance of the user class is created. When a new user is created, the new user is given the next available id number and the number of users in the database increments by 1. This user is made with a default password and an email that was input. Users are emailed a link to set their custom password. Log in is tested next. When we log into our account with the proper user email and password, we can log in successfully.
After that, we test for adding a user. If we add one user, the number of users in the database would add one. In addition, we also test for logout function. The tests are uploaded in github and they are called “test_samply.py.” Additional debugging and edgecase testing was carried out using the debug configuration of flask along with print statements and try/catch statements.

**Test Conclusions**
As is mentioned above, the tests of each function have all been listed specifically. Every function has at least one tests. Some functions even have four to five tests. The only function that we haven’t tested is the “Export as CSV file” function. In this function, we are able to download our schedule as a csv file in our own computer. And we can edit the CSV file in our local excel chart. We have downloaded and tested it for many tests. However, we haven’t figured out a way to test it using either Selenium IDE or Travis CI.

### Missing/Remaining Features and Functionality:

The primary feature we hope to implement in the future is the incorporation of vacation times in the scheduling algorithm. Our client has told us that including vacation times in the scheduler would be the final user story we would need to fulfill in order for his physician group to use the scheduler in their practice. Additionally we would like to implement post call requests for scheduled doctors. A web form in which doctors indicate what day they would like to be on post call can then transfer that information to the scheduling algorithm. The algorithm could then use that information to schedule each doctor to best fit their requested days. Of course conflicting days and multiple doctors requesting the same day are edge cases that would have to be dealt with.

Vacations and post call requests both would enhance the scheduling algorithm, making it more complicated and precise in its functionality. Improving the algorithm itself is another feature we would like to implement in the future, and the previously mentioned features correspond directly with this idea.

Finally, with the ability to download the schedule as a csv, we would like to also implement uploading a csv schedule. The idea here is that administrators can use Jelly Plan to generate a schedule, download it, and then edit the csv to fit their needs if the generated schedule is not ideal. Administrators should then be able to upload the edited csv, in which the Flask api would be used to display the new schedule on the website.
Appendix

Technologies used: Flask microframework, Postgresql, Heroku, Selenium IDE, and Git

Burn Down Charts