Olin College iCal Generator:
Creating iCals for the Spring 2019 Semester

Kristen Behrakis, Athmika Senthilkumar

December 13, 2018

1 Project Overview

The goal of this project is to create automated iCals for the Olin College Spring 2019 Events Calendar and Course Offerings. Users can access our project via a website. Here, they will choose to be emailed iCals for the Events Calendar and/or for courses that they will be taking.

MVP

Website where a user enters their email and is sent iCals for the Spring Events Calendar.

Phase 2

MVP with the additional feature of supporting iCals for Spring Course Offerings. A user will indicate a course title on the website and be sent iCals for the given course.

2 The Process

Front-end

2.1 Creating the Landing Page

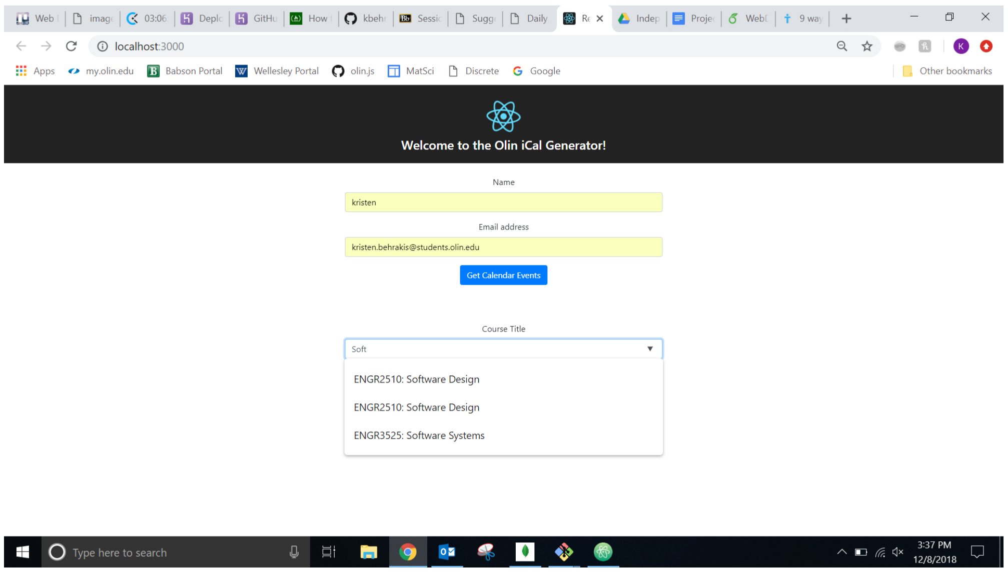
For this project, we used a JavaScript library called ReactJS to build our User Interface. React utilizes components, which are similar to functions. Each component is sent a props variable as input and outputs an interface constructed from those props. The main benefit of this format is the re-usability.
This application is bootstrapped with create-react-app. The landing page includes a form where users can enter an email address. Once the email has been entered, the user then decides if they want to be sent the Events Calendar or a Course Calendar.

Getting Events Calendar

If the user chooses a button called "Get Events Calendar," the entered email is extracted. A call containing this email information is sent to the back-end (see the Constructing iCals section below). A few moments later, a pop-up box will confirm if the email was successfully sent.

Getting Course Calendar

Although the Events Calendar email is standardized, the Course Calendar emails are a bit more complex. Each user can request iCals for one of 49 courses offered in the Spring semester. For convenience, we have implemented a dropdown box with an autocomplete feature for choosing a course (see the Figure below).


Welcome to the Olin iCal Generator!

Name: kristen

Email address: kristen.behrakis@students.olin.edu

Get Calendar Events

Course Title: Soft

ENGR2510: Software Design

ENGR2510: Software Design

ENGR3525: Software Systems

Figure 1: Example of the autocomplete feature.

This dropdown box must be filled with all offered courses immediately when the page loads. Thus, we make a call to the back-end to extract all courses from our database. A function to accomplish this, called componentDidMount(), is invoked during the initialization of the DOM.

Once the user chooses a course, the email is extracted along with the chosen course. A call containing this information is sent to the back-end (see the Constructing iCals section below). A few moments later, a pop-up box will confirm if the email was successfully sent.

Because the course titles can sometimes be long, we have also given the user the option to type in their preferred title for the course. If nothing is entered, the title will default to the full course title.

2.2 Auto-complete

We experimented with a few different UI components that allow users to input classes. We tried implementing a checklist but the web-page look very clustered. After that we a implemented dropdown box, but we realized that it will take too long for the users to find classes, so we switched to a text input field that has autocomplete. The autocomplete was implemented using the dataoption html component. User testing showed that this method works was intutive and efficient.

Back-end

2.3 Storing and Pulling Data

Getting Access to the Data

One of the first steps of this project was getting access to the Spring 2019 Events Calendar and the Course meeting times. All of the information is hosted publicly online.

The Events Calendar webpage is coded in JavaScript. The data we want is held in a table, and we extracted it by using Excel. We then saved the extracted data as a csv file. We also looked quite a bit into web-scraping to bypass the use of Excel. We found a promising article around data scraping websites. Given more time, we believe this would be a solid option to explore.

The Course Calendar information is also available in a pdf format. This information was also extracted via Excel and stored in a csv format.

Populating the Database

We used MongoDB to store the data. MongoDB is a free, open-source database that stores data in JSON-like documents. We constructed a database called mydb with 2 collections, one for the Event Calendar and one for the Course Calendar. A mongoDB collection is the RDMS (relational database management system) equivalent of a table. Collections store documents that are not in the same structure.

For each line in the CSV files, we are adding an entry to the appropriate collection in the database.

Querying the Database

We can query the MongoDB collections by using methods such as find(). The find method allows us to search for all entries that match a given field name. For example, we can query the events calendar database to find all entries that contain the words "Olin Monday." This would return a list of all the matching entries.

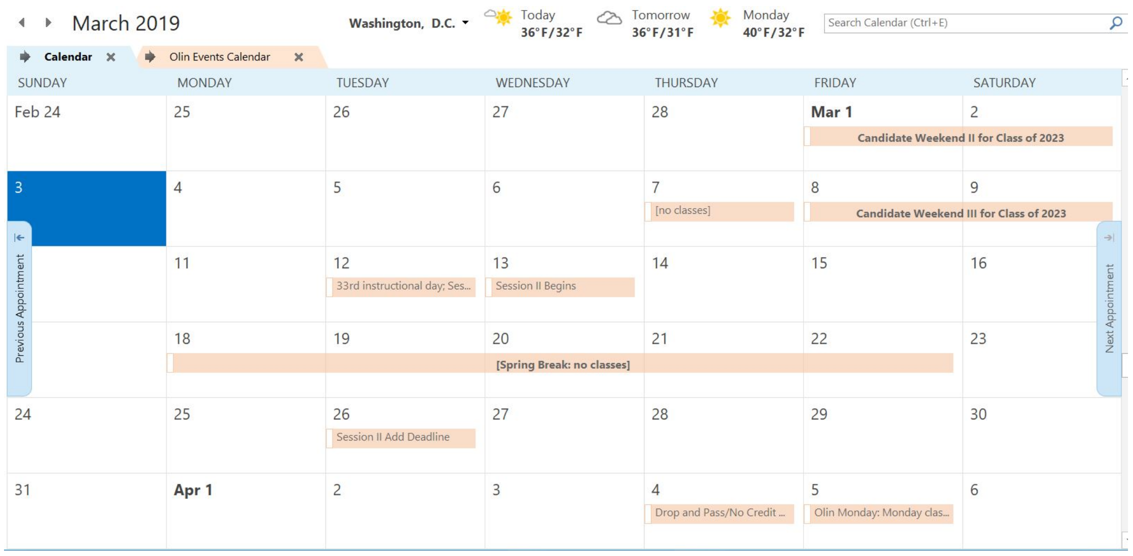
One significant issue we ran into with database populating and querying were timeout issues. JavaScript is a synchronous language, meaning that only one operation will be in progress at one time. Before we could query the database, we had to ensure that it had finished populating. This led to the need for timeout statements, which basically force there to be some set amount of time before the next line of code executes.

2.4 Constructing iCals

To create iCals, we are using npms ical-generator. Using this code allows us to create iCals with start/end times, summaries, descriptions, locations, etc. There is also functionality to set a repeating event as well as excluding specified dates.

iCals for Events Calendar

Because there are so many unique events in the Olin Events Calendar, creating iCals for each of them would result around 30 iCal attachments just for the Spring semester. For a better user experience, we decided to encompass all of these events into one calendar. The users could then open this new calendar in their emails and overlay it onto their main calendar. This is shown in the Figure below.

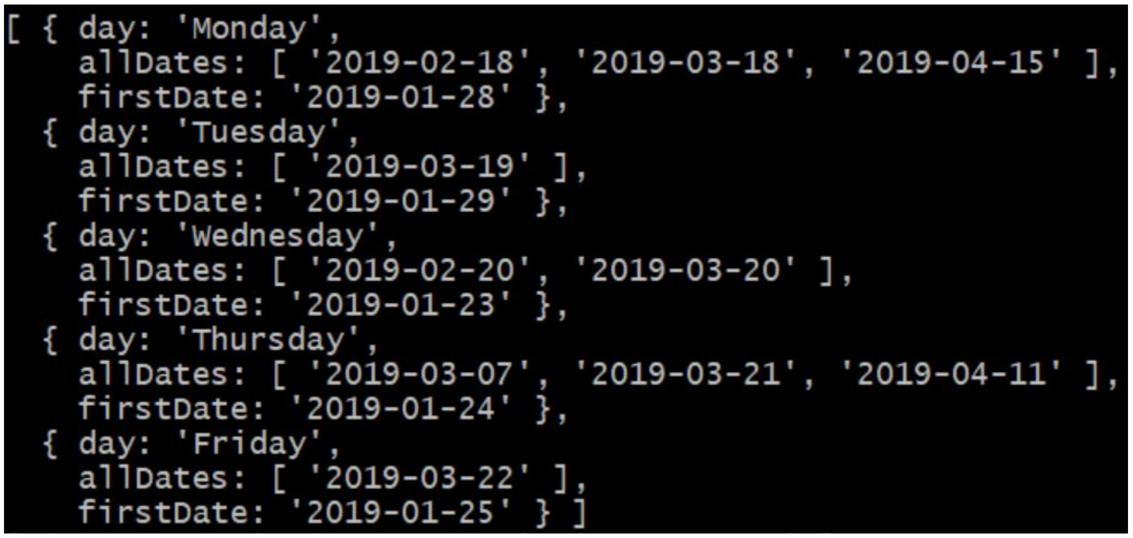


SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
Feb 24	25	26	27	28	Mar 1 Candidate Weekend II for Class of 2023	2
3	4	5	6	7 [no classes]	8 Candidate Weekend III for Class of 2023	9
	11	12 33rd instructional day; Ses...	13 Session II Begins	14	15	16
	18	19	20 [Spring Break: no classes]	21	22	23
24	25	26 Session II Add Deadline	27	28	29	30
31	Apr 1	2	3	4 Drop and Pass/No Credit ...	5 Olin Monday: Monday clas...	6

Figure 2: Olin Events are stored in a separate calendar. The calendar can be overlayed onto the users existing calendar.

iCals for Course Offerings

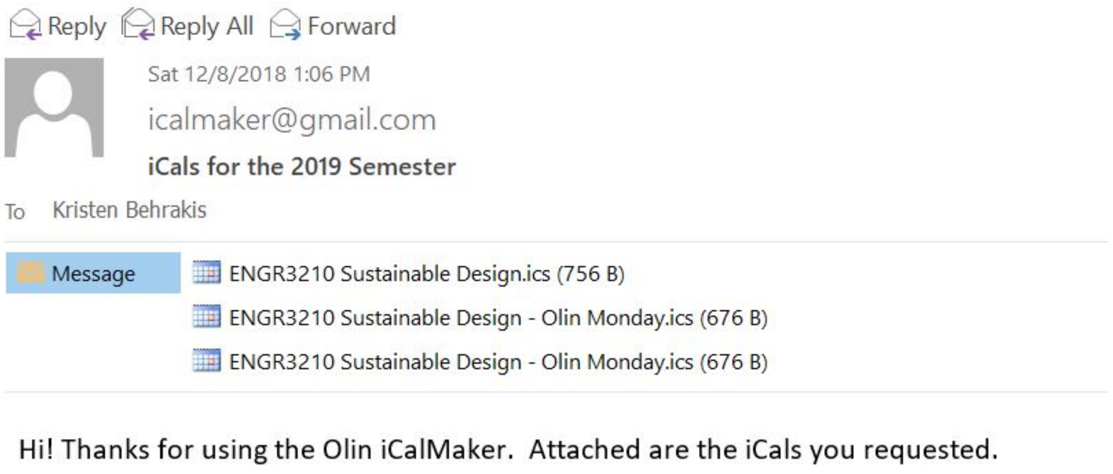
For the course offerings, we used the repeat functionality of the iCal generator. If a course meets on Tuesdays and Thursdays, for example, we can create a repeating event for those days. Originally, we created a list of all the class meeting times during the semester. We then created iCals for each date in the list. This, however, resulted in many iCal attachments to the email. We decided to use the repeat feature to improve the user experience. Using this approach, there are not nearly as many iCal attachments.
If there are no class meetings on one of these days, however, we will use the exclusion functionality. To find the days to exclude, we went through the database and created a data structure to hold the excluded dates. See the Figure below for an example of the data structure.



```
[ { day: 'Monday',
    allDates: [ '2019-02-18', '2019-03-18', '2019-04-15' ],
    firstDate: '2019-01-28' },
  { day: 'Tuesday',
    allDates: [ '2019-03-19' ],
    firstDate: '2019-01-29' },
  { day: 'Wednesday',
    allDates: [ '2019-02-20', '2019-03-20' ],
    firstDate: '2019-01-23' },
  { day: 'Thursday',
    allDates: [ '2019-03-07', '2019-03-21', '2019-04-11' ],
    firstDate: '2019-01-24' },
  { day: 'Friday',
    allDates: [ '2019-03-22' ],
    firstDate: '2019-01-25' } ]
```

Figure 3: The data structure containing days to exclude. These excluded days are Olin Mondays or days off.

If a class meets on a Monday, however, it will also need to meet on any Olin Mondays. In this case, we create special iCals for these Olin Monday dates. See the Figure below.


Reply Reply All Forward

Sat 12/8/2018 1:06 PM

icalmaker@gmail.com

iCals for the 2019 Semester

To Kristen Behrakis

Message

ENGR3210 Sustainable Design.ics (756 B)

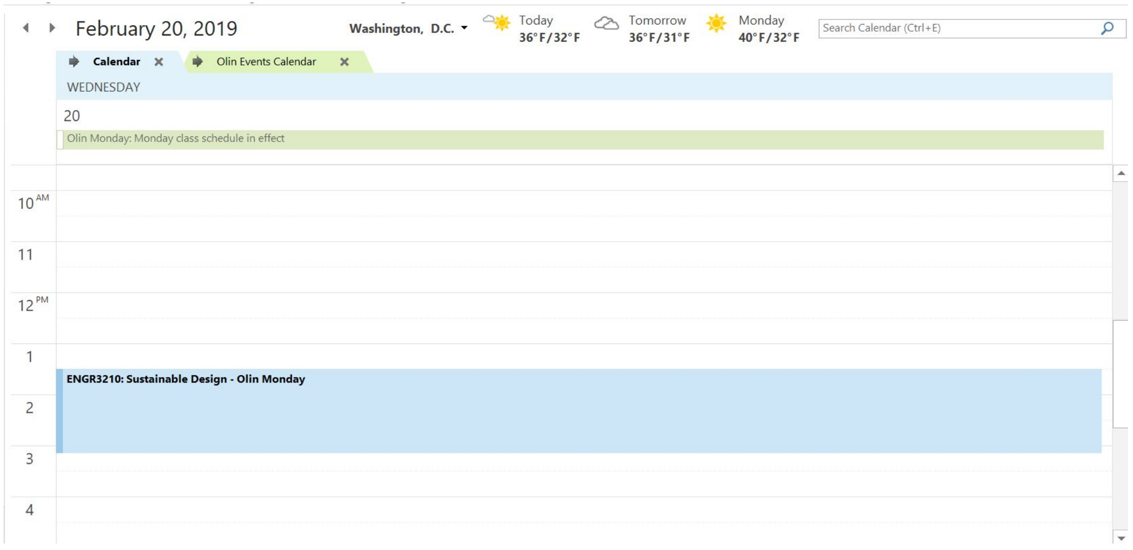
ENGR3210 Sustainable Design - Olin Monday.ics (676 B)

ENGR3210 Sustainable Design - Olin Monday.ics (676 B)

Hi! Thanks for using the Olin iCalMaker. Attached are the iCals you requested.

Figure 4: If a class meets on an Olin Monday, then there will be special calendar events sent along with the repeating event.

The final result is an email with iCals for all class meeting times during the semester. If a class does not occur because of an Olin Monday or a day off, then a calendar event will not appear. Alternatively, if a course meets on Mondays, it will also meet on any Olin Mondays. This can be seen in the example below, where a course that typically would not meet on a Monday now has a class meeting time due to it being an Olin Monday.


February 20, 2019

Washington, D.C. Today 36°F/32°F Tomorrow 36°F/31°F Monday 40°F/32°F

Search Calendar (Ctrl+E)

Calendar Olin Events Calendar

WEDNESDAY

20

Olin Monday: Monday class schedule in effect

ENGR3210: Sustainable Design - Olin Monday

Figure 5: Sustainable Design does not usually meet on Wednesdays, but a calendar event is shown because February 20th is an Olin Monday.

2.5 Parsing the Data

We wrote functions that converted the format of dates and times in the database into ones that we can format. For example TF-10.30 to 3.30pm will be parsed to convert the 12 hour time into 24 hour time and the days will be recognized and stored. Some classes had multiple meeting times in which case a flag was triggered to ensure that multiple iCal events are sent. A parsing function was also used to convert the dates in the events calendars into a format that can be stored and processed.

2.6 Sending Automated Emails

We know that we can send iCals to each other via an email containing an iCal. Once a user receives an iCal email, they will have the ability to accept or reject the meeting. Many existing websites send automated emails to users, and the same approach will be taken here.

First, we setup a gmail account that we will send all of our email from (username: icalmaker@gmail.com). When Gmail notices that we are sending automated emails from this account, it alerts us to this behavior and does not send the emails. To change this behavior, we had to shut off app block to allow less secure apps to access the email.

There are some existing node modules that can help us with sending automated emails. The one we are using is called nodemailer, which allows us to send email via the SMTP protocol. Nodemailer is implemented on the server side, so when running the application, it is important to run both the server side and the client side. Nodemailer also has some functionality for adding attachments to emails as well as some specific event handling for icals. Once the iCals have been created, we use nodemailer to attach them to emails and send them to the desired recipient. To implement nodemailer, we followed the tutorial on the nodemailer website as well as an article describing how to implement it with React.

3 Hosting the Application

Our app is hosted on Heroku and can be found at: https://olin-ical-generator.herokuapp.com/

Heroku allows us to build and operate our application on the cloud. Heroku has some convenient built-in functionality for deploying our application. We were able to connect Heroku to our Github repository (https://github.com/kbehrakis/WebDev-Project). There are adjustments that need to be made to have a successful deployment, however. Many of these changes were due to the file structure of our code. In particular, we have two folders: back-end and front-end.

1. Adding a package.json to the root

One of the first issues we ran into was Heroku not knowing which dependencies to install. The goal is to install dependencies that are in the package.json files that reside in our 2 folders. However, Heroku will only search for package.json files at the root-level. Thus, we needed to create a new, root-level package.json file that would direct to the internal folders.

package.json:

```
{
  "scripts": {
    "postinstall": "npm --prefix ./front-end install && npm --prefix ./back-end install"
  }
}
```

2. Making two separate deployments

Because we have both a client and a server side, we must create 2 separate Heroku deployments. The front-end deployment will make calls to the back-end deployment. Instead of a GET request to localhost, we will be executing a GET request to the website holding the back-end:

```
// axios.get('http://localhost:3002/listClasses')
axios.get('https://olin-ical-generator-backend.herokuapp.com/listClasses')
```

3. Adding a Procfile

Heroku uses a file called a Procfile, which specifies the commands to execute upon startup. The Procfile should be added at the root level. If we are running the back-end deployment, we include the following in the Procfile:

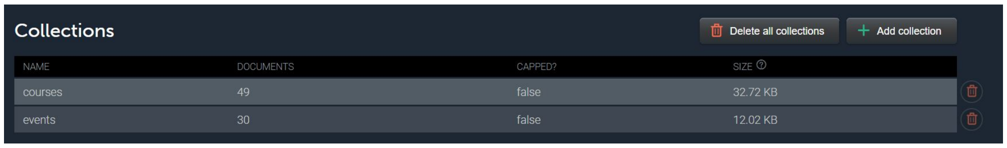
```
web: npm start --prefix back-end
```

If we are running the front-end deployment, we include the following in the Procfile:

```
web: npm start --prefix front-end
```

4. Connecting to a cloud database

Because we are deploying to the cloud, we must also migrate our database. To accomplish this, we used a Heroku add-on called mLab. mLab is a cloud MongoDB service, which we will use to store our 2 collections: courses and events.



NAME	DOCUMENTS	CAPPED?	SIZE
courses	49	false	32.72 KB
events	30	false	12.02 KB

Figure 6: Our cloud database hosted on mLab. Shown are our 2 collections for courses and events.

We access the database by using the URI indicated on mLab along with the user login information.

5. Authenticating Gmail

Although emailing with Gmail worked on localhost, there is an additional step that needs to be taken for Heroku. The Gmail email and password must be stored within the Heroku application, which can be done as follows:

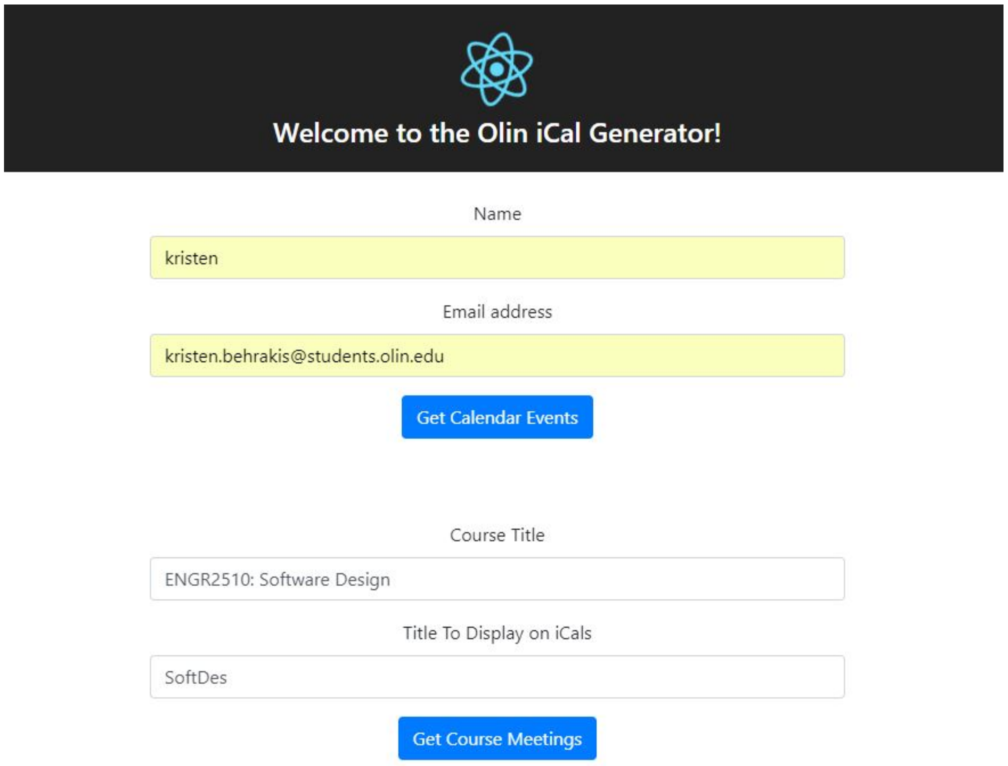
```
$ heroku config:add GMAIL_SMTP_USER=username@gmail.com --app INSERT_APP_NAME
$ heroku config:add GMAIL_SMTP_PASSWORD=yourpassword --app INSERT_APP_NAME
```

6. Preventing application from sleeping

Because we are using a free version of Heroku, our applications will time out after 30 minutes without use. To prevent this from happening, we had to ping the websites occasionally to keep them active.

```
setInterval(function() {
    https.get("https://olin-ical-generator.herokuapp.com/");
    https.get("https://olin-ical-generator-backend.herokuapp.com/");
    console.log("pinging now")
}, 300000);
```

4 The Results



Name

kristen

Email address

kristen.behrakis@students.olin.edu

Course Title

ENGR2510: Software Design

Title To Display on iCals

SoftDes

Figure 7: Image of our web interface.