

Chapter 1: Updated System Requirements Document

Problem definition

Eckerd College's process for accepting room draw applications is a lengthy process that can be improved by the use of a web-based system that allows students to apply for their desired dorms online. By having the applications online, this will remove the duration it takes for the different types of applications to be submitted, as well as provide a way for students to request roommates and know the status of that request (whether it has been accepted or denied), as that is currently unavailable. These applications are overseen by the housing director, so having that online will also make it easier for the director to view all of the submissions at once, eventually allowing him the decision to accept or reject an application. Additionally, the procedure for assigning first-year students is flawed, as it consists of a questionnaire that maps roommates together based on few similar responses, as opposed to an evolutionary algorithm that will choose the most compatible roommates, based on their answers to that questionnaire.

How is it solved now?

Room draw is an extensive process that is all managed by hand. Currently, students fill out applications for the dorms that they wish to live in for the new school year. There are different applications for different housing selections: "Omega & Squatting", "Nu, Oberg, & Singles", "Themed, Traditional, and Alta Mar". The availability of these applications depends on the type of housing a student is seeking; therefore, this process takes a little over a month to complete, as applications for each type are available for weeks at a time. Only one person out of a group of roommates is to fill out this form, as they are to write down the information for their roommates on the form as well. Upon completion, the student must submit this form to Housing before the deadline of application submission passes. Students are given the day in which they are to go to Fox Hall and participate in room draw, where they choose their housing. Concerning roommate assignments, freshmen are assigned roommates loosely based off of their answers to a survey. If individuals answer a certain amount of questions similarly, they are paired as roommates.

Challenges to present solutions

- The process of submitting and receiving housing applications is very lengthy.
- Many students request other students as roommates without their knowledge or approval
- Students often forget the dates that applications are available, as well as the deadline.
- Eckerd's procedure for assigning freshmen their roommates is inaccurate and has no defined algorithm to depict the probability of roommates being suitable matches.

What will you do differently?

- Online application forms
 - Open/Close period
- Genetic Algorithm for roommate matching
- View applications that users have submitted
- View all pending roommate requests
 - Allow ability to accept or deny request
- Housing – view all submitted housing requests

Team member contributions

Ray - development of APIs, requirements gathering, code review

Ashley - frontend, web-based design/implementation, documentation

Priorities

The features of this project will be prioritized as such:

1. Online applications
 - a) Applications will be listed by type
 - b) Open/close time periods for applications will be enforced
 - c) Students can submit applications
 - d) Housing can overview all of the applications
2. Roommate assignment for freshmen
 - a) Student answers to questionnaire recorded
 - b) Weighted answers to questionnaire
 - c) Students will be paired based on preferences
 - d) Housing will be able to see the results of matching
3. A 'home page'
 - a) Students can sign in via their Google account
 - b) Students can click a link to housing rules
 - c) Link to view applications
 - d) Link to view requests
 - e) c. Features are hidden if not signed in

Chapter 2: Updated System Requirements Document

GUI

- HTML
- CSS
- JavaScript
- BootStrap
- Vue.js

Concerning the front-end design of the software, HTML will be used for laying out the text and images of the different pages of the website. CSS will be used for formatting and design, while JavaScript will be utilized for incorporating the functionality aspect of the software. As the major functions for the front-end web page will be written in JavaScript, to ease with the development of the design and to increase the level of efficiency when coding the program, we will be using the Vue framework in order to utilize servant-side functionality on the front end. Additionally, Bootstrap's library will be utilized in order to provide neat web design characteristics (such as button and table designs) for the web page.

Server

The software will run on an Entropy server.

Server Languages

- Node, with Express.js and mysqljs
- Axios

Database

- MySQL
- We will use MySQL to store all of the data for this system. We felt that due to the relational nature of the rooms/roommates/submissions/applications/terms it would make most sense to use an actual database. It will also be good practice using an actual database.

Team Members' Coding Experience

Ashley - Java, HTML, CSS, JavaScript, Swift

Ray - JavaScript, RESTAPIs, Java, HTML, Vue, databases

How Our System Works

We have developed a single-page web application that allows users to apply to a specific housing application, view their submissions, as well as view others who have requested them as a roommate. Additionally, a separate page has been created to showcase our first-years roommate pairing algorithm.

1. Code View

Components

```

1 <template>
2 <div>
3 <br>
4 <h1>Viewing Roommate Request</h1>
5 <p>You have received a roommate request from <b>{{request.requester_email}}</b></p>
6 <div>
7   {{request.requester_email}} is requesting you as a roommate. The current status of this request is <b>{{request.request_status}}</b>. In
8 </div><br>
9   <table class="table">
10     <thead>
11       <tr>
12         <th scope="col">Requestee:</th>
13         <th scope="col">Status:</th>
14       </tr>
15     </thead>
16     <tbody>
17       <tr v-for="item in requests" :key="item.submission_id">
18         <td id="sub">{{item.requestee_email}}</td>
19         <td id="stu">{{item.request_status}}</td>
20       </tr>
21     </tbody>
22   </table>
23   <br>
24   <button class="btn btn-info btn-md" v-on:click="respondRequest(1)">Click to Accept</button><br>
25   <button class="btn btn-info btn-md" v-on:click="respondRequest(0)">Click to Deny</button>
26 </div>
27 </template>
28
29
30
31
32
33
34
35

```

```

1 requests: "",
2 request: ""
3 },
4 props: ['requester_email', 'curUserEmail', 'request_id', 'submission_id'],
5 methods: {
6   loadApplications: function() {
7     let vm = this;
8     if(this.$props.curUserEmail == "0"){
9       setTimeout(function(){vm.loadApplications();},1000);
10     } else {
11       axios
12         .get("http://entropy7.nas.eckerd.edu:3000/requests/" + this.$props.request_id, {
13           myEmail: this.$props.curUserEmail
14         })
15         .then(function(response) {
16           vm.$set(vm, "request", response.data[0]);
17         })
18         .catch(function(error) {
19           console.log(error);
20         });
21       axios
22         .get("http://entropy7.nas.eckerd.edu:3000/requestsBySubmissionID/" + this.$props.submission_id, {
23           myEmail: this.$props.curUserEmail
24         })
25         .then(function(response) {
26           vm.$set(vm, "requests", response.data);
27         })
28         .catch(function(error) {
29           console.log(error);
30         });
31     }
32   }
33 }

```

```

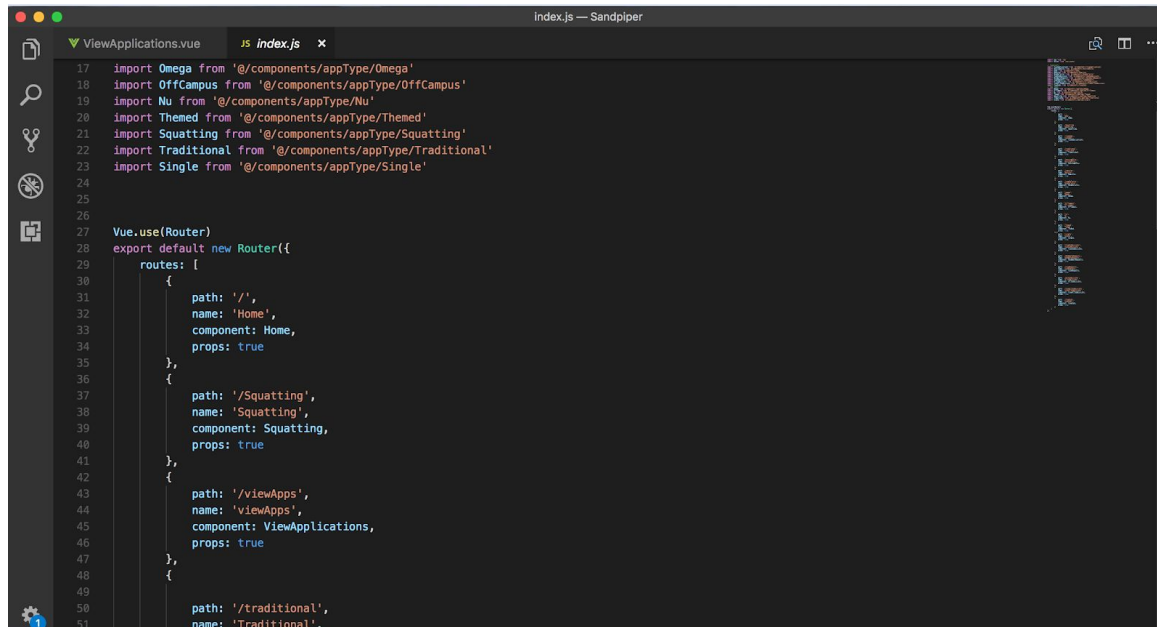
73 .catch(function(error) {
74   console.log(error);
75 });
76
77
78
79
80 respondRequest: function(statusUpdate){
81   let vm = this;
82   let answer = statusUpdate ? "accepted" : "denied";
83   axios
84     .post("http://entropy7.nas.eckerd.edu:3000/updateRequest/", {
85       requestID: vm.$props.request_id,
86       status: answer
87     })
88     .then(function(response) {
89       vm.$router.push({name: 'RoommateRequests'});
90     })
91     .catch(function(error) {
92       console.log(error);
93     });
94 }
95
96 created: function() {
97   this.loadApplications();
98 }
99
100
101

```

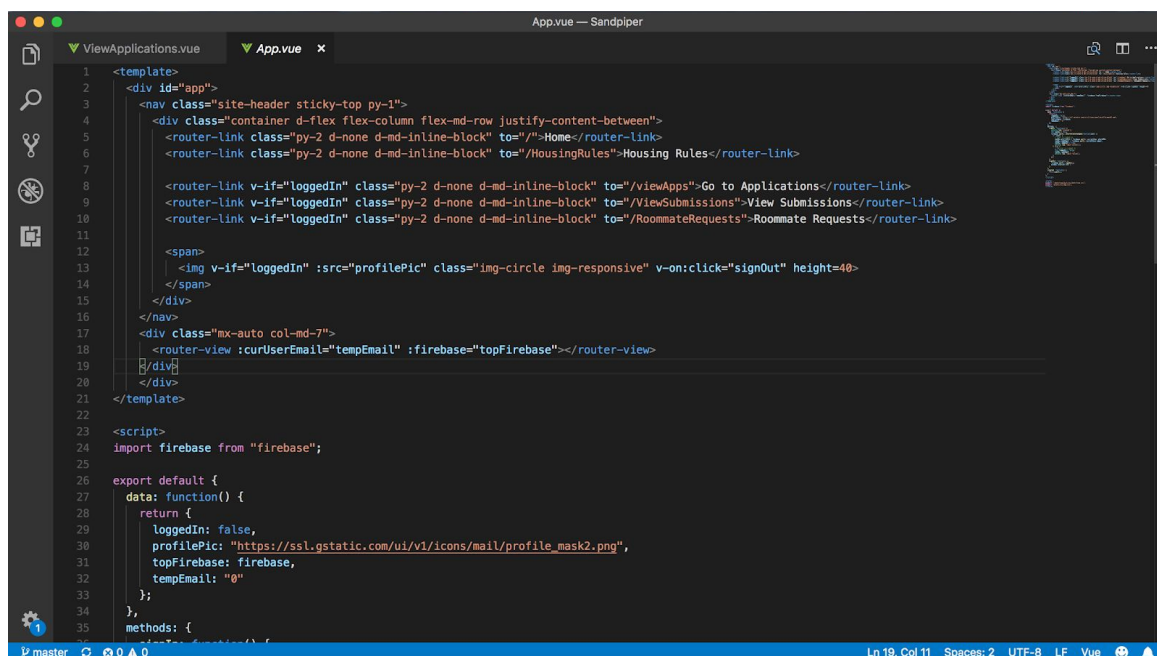
The above screenshots outlines one of the Vue components that was designed, the 'View Requests' component. Components are typically separated into three segments: 'Template' (where the HTML section is written), 'Script' (for scripting languages, or JavaScript using the Vue.js framework in our case), and 'Style' (for CSS design). As our web page is a single page application, which means that the different web 'pages' are loaded on the server-side to eliminate

the need for server requests, we use ‘components’. Each new view that is loaded on the screen as the user navigates via links and buttons is a component.

Routers



```
17 import Omega from '@components/appType/Omega'
18 import OffCampus from '@components/appType/OffCampus'
19 import Nu from '@components/appType/Nu'
20 import Themed from '@components/appType/Themed'
21 import Squatting from '@components/appType/Squatting'
22 import Traditional from '@components/appType/Traditional'
23 import Single from '@components/appType/Single'
24
25
26
27 Vue.use(Router)
28 export default new Router({
29   routes: [
30     {
31       path: '/',
32       name: 'Home',
33       component: Home,
34       props: true
35     },
36     {
37       path: '/Squatting',
38       name: 'Squatting',
39       component: Squatting,
40       props: true
41     },
42     {
43       path: '/viewApps',
44       name: 'viewApps',
45       component: ViewApplications,
46       props: true
47     },
48     {
49       path: '/traditional',
50       name: 'Traditional',
51     },
52   ],
53 })
```



```
1 <template>
2   <div id="app">
3     <nav class="site-header sticky-top py-1">
4       <div class="container d-flex flex-column flex-md-row justify-content-between">
5         <router-link class="py-2 d-none d-md-inline-block" to="/">Home</router-link>
6         <router-link class="py-2 d-none d-md-inline-block" to="/HousingRules">Housing Rules</router-link>
7
8         <router-link v-if="loggedIn" class="py-2 d-none d-md-inline-block" to="/viewApps">Go to Applications</router-link>
9         <router-link v-if="loggedIn" class="py-2 d-none d-md-inline-block" to="/ViewSubmissions">View Submissions</router-link>
10        <router-link v-if="loggedIn" class="py-2 d-none d-md-inline-block" to="/RoommateRequests">Roommate Requests</router-link>
11
12        <span>
13          
14        </span>
15      </div>
16    </nav>
17    <div class="mx-auto col-md-7">
18      <router-view :curUserEmail="tempEmail" :firebase="topFirebase"></router-view>
19    </div>
20  </div>
21 </template>
22
23 <script>
24   import firebase from "firebase";
25
26   export default {
27     data: function() {
28       return {
29         loggedIn: false,
30         profilePic: "https://ssl.gstatic.com/ui/v1/icons/mail/profile_mask2.png",
31         topFirebase: firebase,
32         tempEmail: "0"
33       };
34     },
35     methods: {
36       // ...
37     }
38   }
39 </script>
```

Each component is accessed through the use of ‘routers’. The top screenshot shows the components created for the housing applications. This is a key component to single-page applications, as routers act as the navigational tool to access each component dynamically. Additionally, we use the ‘router-link’ element, which links (or routes) to a particular component, as shown in the bottom screenshot above.

Rest API

```
69 app.get("/students/:email",function(req, res){
70   const sql = "SELECT * FROM student WHERE email = " + connection.escape(req.params.email);
71   dbQuery(req,res,sql);
72 });
73
74 app.get("/applications/:applicationID",function(req, res){
75   const sql = "SELECT * FROM application WHERE app_id = " + connection.escape(req.params.applicationID);
76   dbQuery(req,res,sql);
77 });
78
79 app.get("/submissions/:submissionID",function(req, res){
80   const sql = "SELECT * FROM submission WHERE submission_id = " + connection.escape(req.params.submissionID);
81   dbQuery(req,res,sql);
82 });
83
84 app.get("/requests/:requestID",function(req, res){
85   const sql = "SELECT * FROM request WHERE request_id = " + connection.escape(req.params.requestID);
86   dbQuery(req,res,sql);
87 });
88
89 app.get("/requestsBySubmissionID/:submissionID",function(req, res){
90   const sql = "SELECT * FROM request WHERE submission_id = " + connection.escape(req.params.submissionID);
91   dbQuery(req,res,sql);
92 });
93
94
```

The above screenshot displays the API designed to make requests to the database from the server. In the system, we use these GET requests to retrieve specific user information, typically to print the information on the screen (usually in a table). We also utilize POST requests, which were used when data was submitted, such as when users click the ‘Submit’ button after filling out an application.

SQL Database

```
dbSetup.sql — Sandpiper
ViewApplications.vue dbSetup.sql x
26 app_term INT NOT NULL,
27 FOREIGN KEY (app_term) REFERENCES term(term_code)
28 );
29
30 CREATE TABLE student (
31   first_name VARCHAR(20) NOT NULL,
32   last_name VARCHAR(20) NOT NULL,
33   class_standing INT NOT NULL, # 0 (freshmen) through 3(senior)
34   email VARCHAR(30) NOT NULL PRIMARY KEY,
35   phone VARCHAR(30) # Not sure if we really need this
36 );
37
38 CREATE TABLE submission (
39   submission_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
40   primary_student_email VARCHAR(30) NOT NULL,
41   submission_date DATETIME NOT NULL DEFAULT NOW(),
42   app_id INT NOT NULL,
43   room VARCHAR(30),
44   room_preference VARCHAR(30),
45   sub_status VARCHAR(30),
46   FOREIGN KEY (primary_student_email) REFERENCES student(email),
47   FOREIGN KEY (app_id) REFERENCES application(app_id)
48 );
49
50
51 CREATE TABLE request (
52   request_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
53   requester_email VARCHAR(30) NOT NULL,
54   requestee_email VARCHAR(30) NOT NULL,
55   submission_id INT NOT NULL,
56   request_status VARCHAR(30), # pending, approved, denied
57   FOREIGN KEY (requester_email) REFERENCES student(email),
58   FOREIGN KEY (submission_id) REFERENCES submission(submission_id)
59 );
60
61
```

A SQL database was set up to store entered data. Separate tables stored specific information, such as the ‘student’ table, which held user information (e.g. first and last name), and the ‘submission’ table, which held user data pertaining to application submission (e.g. the student’s email and room preference).

Roommate Algorithm

Roommate Pairing Tool

Paste student data in the format: StudentID, Q1 Value, Q2 Value, etc

Improve Average Compatability:

Welcome to the roommate pairing tool. Please start by entering an even number of student to be paired. Student information should be in the format of student ID followed by comma separate number values.

Ex: 1373719, 110, 110, 110, 50, 60, 10, 110, 50, 50, 110

This page doesn’t connect to the server at all and can be served statically. Essentially all you have to do is paste in a list of roommates, as seen above and it will automatically come up with a near optimal pairing. It works by randomly pairing up the roommates and then randomly swapping them to find better pairings.

2. User View

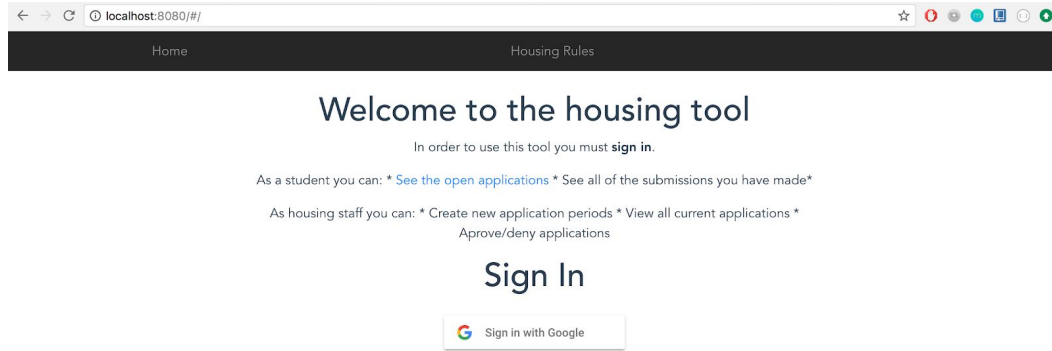


Fig. 1 - Home Page

The first screen that users see upon arrival to our page is the home page. There is also a navigation bar with links to either 'Home' (the current page), or to 'Housing Rules'. Simple HTML is used to create the current formatting and text that is seen on the page. Users are also prompted to 'Sign in with Google', through the use of Firebase. Upon clicking the link to sign in, users will either enter their Google account information, or will be signed in if they are already signed in to Google.

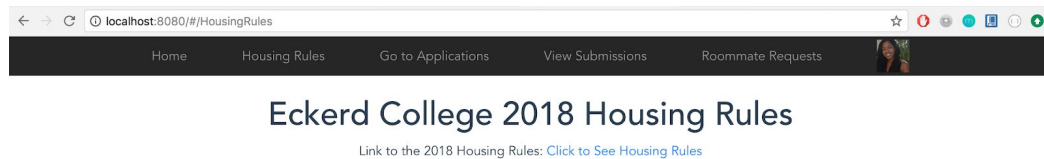
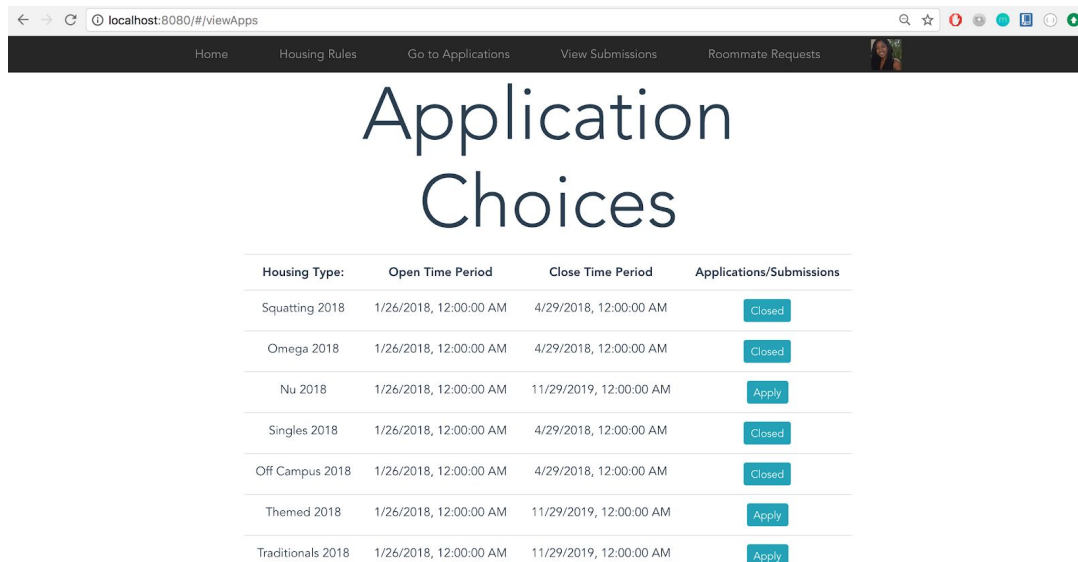


Fig 2. Housing Rules

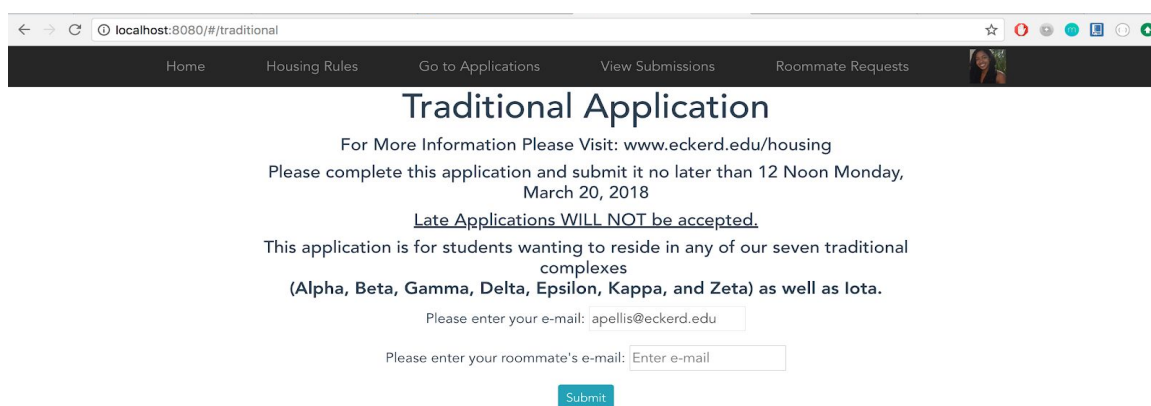
Once the user signs in, their Google account avatar will appear in the rightmost corner of the navigation bar. The 'Housing Rule' component consists of a link that leads the user to an external Google Drive page that contains the 'Housing Rules' document.



Housing Type:	Open Time Period	Close Time Period	Applications/Submissions
Squatting 2018	1/26/2018, 12:00:00 AM	4/29/2018, 12:00:00 AM	Closed
Omega 2018	1/26/2018, 12:00:00 AM	4/29/2018, 12:00:00 AM	Closed
Nu 2018	1/26/2018, 12:00:00 AM	11/29/2019, 12:00:00 AM	Apply
Singles 2018	1/26/2018, 12:00:00 AM	4/29/2018, 12:00:00 AM	Closed
Off Campus 2018	1/26/2018, 12:00:00 AM	4/29/2018, 12:00:00 AM	Closed
Themed 2018	1/26/2018, 12:00:00 AM	11/29/2019, 12:00:00 AM	Apply
Traditionals 2018	1/26/2018, 12:00:00 AM	11/29/2019, 12:00:00 AM	Apply

Fig. 3 Application Choices

The ‘Application Choices’ page contains a table that lists all of the currently available applications that students can apply for, as well as buttons for each one. The availability of applications depends on whether it is within the ‘Open Time Period’. If so, the button for the application will say ‘Apply’ and can be selected; otherwise the button will say ‘Closed’ and will remain static. Upon clicking the ‘Apply’ button, users will instantly be navigated to the relative component for the particular housing type that they are applying to. For example, if a user selects Nu, they will be sent to a /Nu component that has the application for Nu housing.



Traditional Application

For More Information Please Visit: www.eckerd.edu/housing

Please complete this application and submit it no later than 12 Noon Monday, March 20, 2018

Late Applications WILL NOT be accepted.

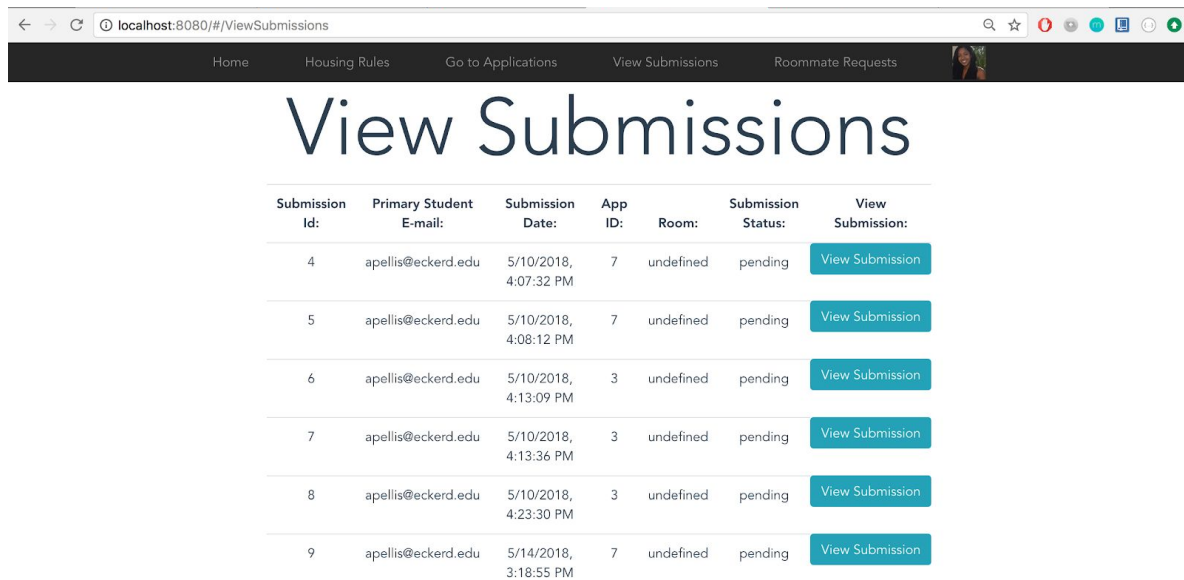
This application is for students wanting to reside in any of our seven traditional complexes
(Alpha, Beta, Gamma, Delta, Epsilon, Kappa, and Zeta) as well as Iota.

Please enter your e-mail:

Please enter your roommate's e-mail:

Fig. 4 Application for Selected Housing Type

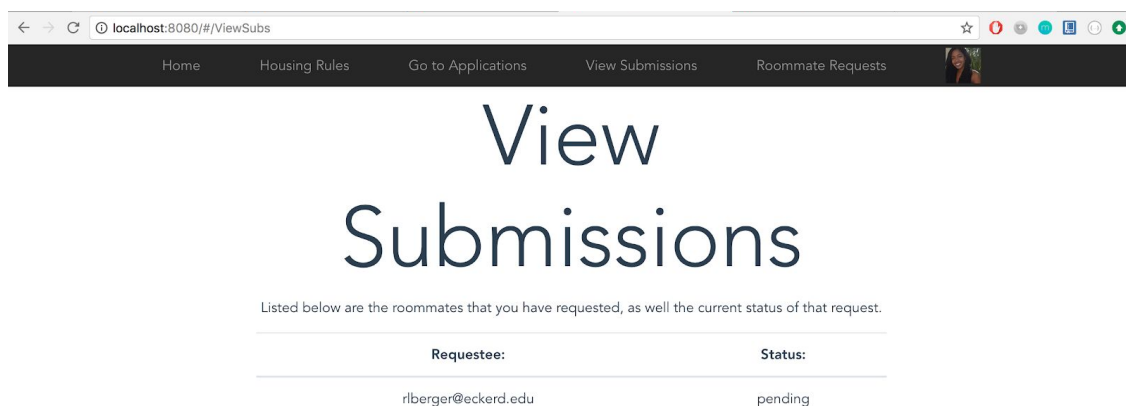
Upon clicking the ‘Apply’ button, users are taken to a component for their selected housing application. Here, the users must fill out text fields (and often checkboxes); Vue.js plays a key role here, as we use a feature called ‘v-bind’ which is a form of data binding. As users enter their information, v-bind is used to capture their information and use it in certain functions (such as sending their information to the database on the server). Each application requires users to enter their preferred roommate(s). Once complete, the user clicks the ‘Submit’ button.



Submission Id:	Primary Student E-mail:	Submission Date:	App ID:	Room:	Submission Status:	View Submission:
4	apellis@eckerd.edu	5/10/2018, 4:07:32 PM	7	undefined	pending	View Submission
5	apellis@eckerd.edu	5/10/2018, 4:08:12 PM	7	undefined	pending	View Submission
6	apellis@eckerd.edu	5/10/2018, 4:13:09 PM	3	undefined	pending	View Submission
7	apellis@eckerd.edu	5/10/2018, 4:13:36 PM	3	undefined	pending	View Submission
8	apellis@eckerd.edu	5/10/2018, 4:23:30 PM	3	undefined	pending	View Submission
9	apellis@eckerd.edu	5/14/2018, 3:18:55 PM	7	undefined	pending	View Submission

Fig. 5 View Submissions

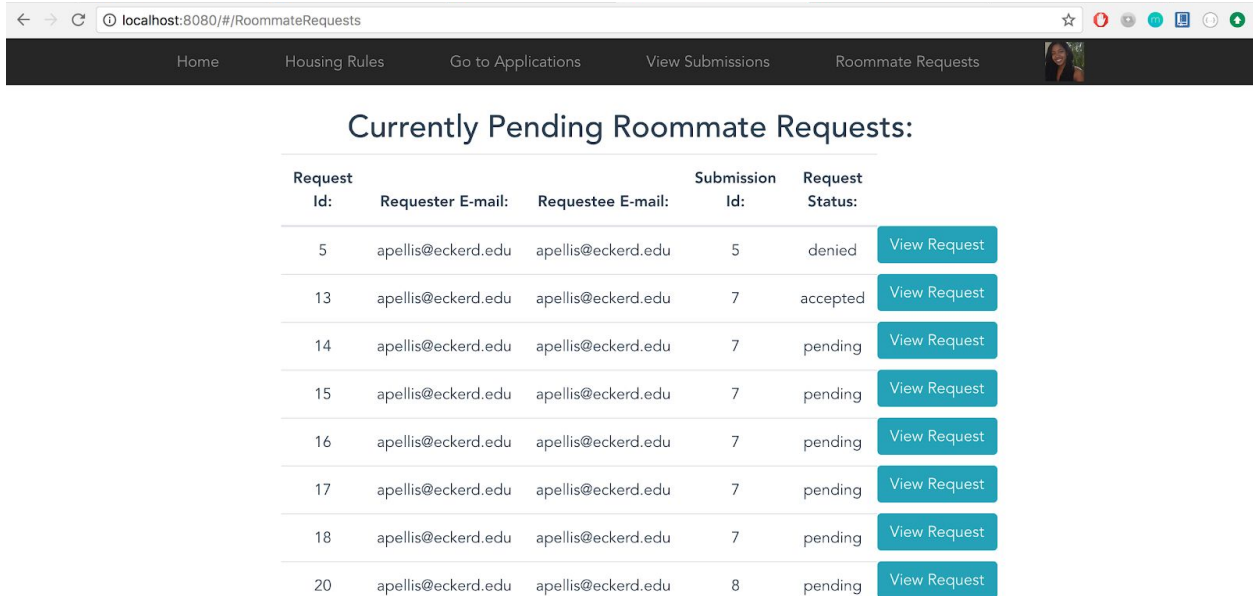
Upon clicking ‘Submit’, users are immediately sent to the ‘View Submissions’ component, which is a table that lists all of their application submissions, the date and time of submission, the application ID associated with the submission, their desired room, the status of that submission, as well as a button titled ‘View Submission’ that allows the user to view the specifics of each submission (meaning the requested roommates and the status of that request).



Requestee:	Status:
rlberger@eckerd.edu	pending

Fig. 6 ViewSubs

The 'ViewSubs' page is an extension of 'View Submissions'; here, users are able to view the roommates that they have requested when they were filling out the application. Additionally, the user is also able to see the status of that request. 'Pending' means that the requested roommate has not accepted or denied the request, 'Accepted' means that the requested roommate has accepted the request, and 'Denied' means that the requested roommate has denied the request.



Request Id:	Requester E-mail:	Requestee E-mail:	Submission Id:	Request Status:	
5	apellis@eckerd.edu	apellis@eckerd.edu	5	denied	View Request
13	apellis@eckerd.edu	apellis@eckerd.edu	7	accepted	View Request
14	apellis@eckerd.edu	apellis@eckerd.edu	7	pending	View Request
15	apellis@eckerd.edu	apellis@eckerd.edu	7	pending	View Request
16	apellis@eckerd.edu	apellis@eckerd.edu	7	pending	View Request
17	apellis@eckerd.edu	apellis@eckerd.edu	7	pending	View Request
18	apellis@eckerd.edu	apellis@eckerd.edu	7	pending	View Request
20	apellis@eckerd.edu	apellis@eckerd.edu	8	pending	View Request

Fig. 7 Roommate Requests

'Roommate Requests' lists all of the other persons that have requested the user as a roommate. Listed in the table are the request ID, the requester e-mail, requestee e-mail (always the current user), submission ID, request status (which shows whether the user has accepted, denied, or has not yet responded to a request), as well as a navigational button to view each individual request. 'View Request' leads to another component, which outlines the details of each request and provides the user with the opportunity to give their response.

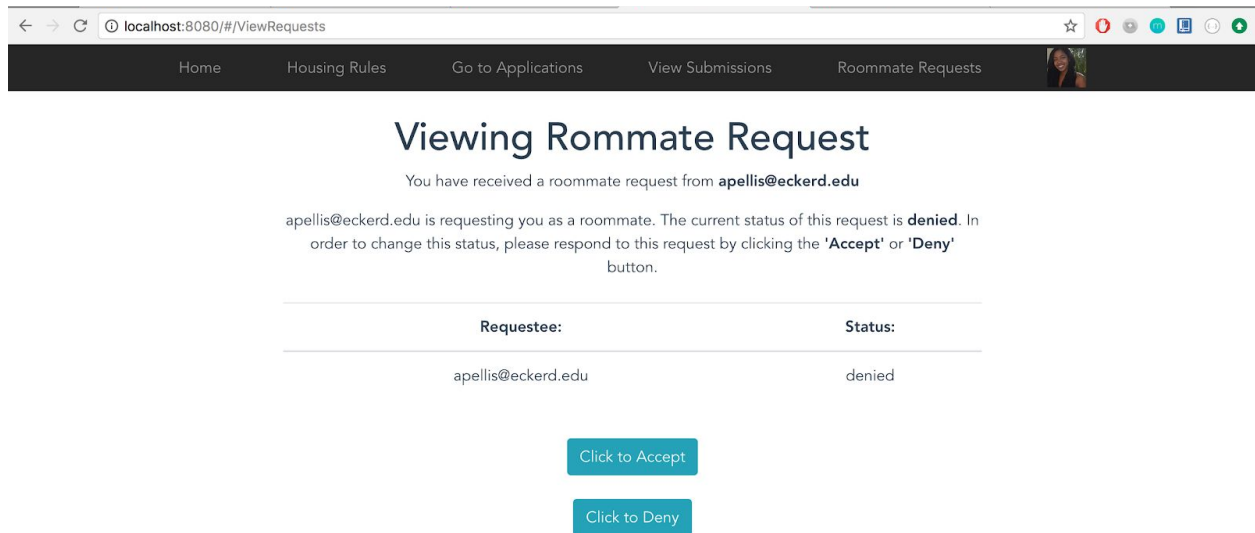


Fig. 8 View Requests

The 'View Requests' component shows the user who is requesting them, and the current status of that request. Users can either choose to accept or deny the request by clicking 'Click to Accept' or 'Click to Deny'. Once the user accepts or denies them, the status is changed to reflect that decision.

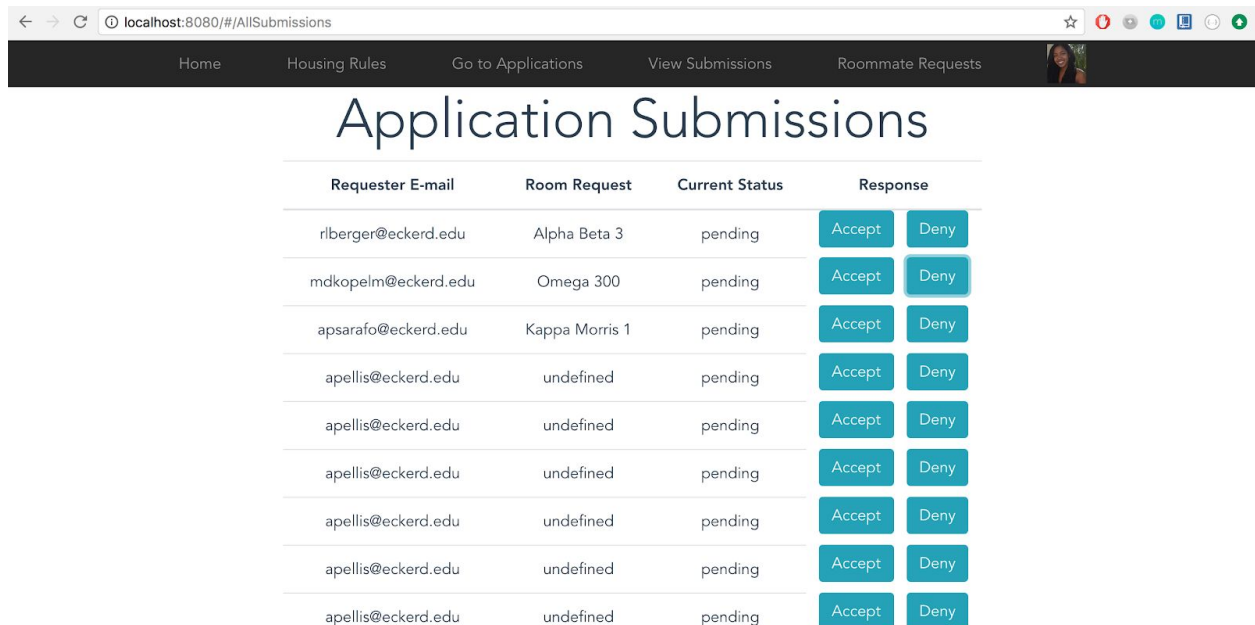


Fig. 9 All Application Submissions

Ideally, this component is a way in which housing will be able to see all of the applications that have been submitted thus far, including the user that has made the request and

the room that they are requesting. They would also have the ability to immediately accept or deny submissions.

Chapter 3 – Lessons Learned and Future Plans

Throughout the course of the semester, working on this project has been a truly great learning experience – not only related to technical, software-related information but also about how we function as teammates within the context of this project. We found that categorizing the work as either frontend or backend, and then assigning someone to each category as most suitable for this project. Specifically, Ray worked on the backend portion, while Ashley worked on the frontend; however, there was a lot of communication in order to ensure that both members were always aware of what the other person was doing and how it helped with the overall coding and implementation of the software.

Working with a teammate who has a completely different schedule often resulted in both concurrent and asynchronous updates to the code, as a lot of the work was split up and worked on independently. However, this worked out well for us, as we were able to work harmoniously even when we did not always have the time to physically work on the code together. Much time was devoted to ensuring that each member's code was suitable and functioning, in order to ensure that the other group member was able to work off of what was previously updated. This was not a problem, especially due to the extremely frequent updates to the github folder that allowed us to immediately acknowledge any changes that the other teammate has made. Even after separate updates, we made sure to inform the other when a git pull should be made to update the code. Overall, communication was a key component to progressing in our project, and we believe that we did well in that aspect.

While we immediately learned the importance of communication and productivity to ensure the success of our code, there were also many technical things that were learned. Each teammate was able to learn new aspects of code that we had otherwise been unfamiliar with. Particularly, Ashley was unfamiliar with most of the backend related work, as she had never been exposed to server-side maintenance. Additionally, features concerning the frontend, such as the Vue.js framework, were also new to her; thus, a great deal of time was spent learning how Vue worked, as it was heavily used in the code. On the other hand, Ray was able to refine his skills on building backend APIs and learning more about using vue on a larger scale. He also realized the reason why tools like graphql are becoming popular (they basically help you avoid making a bunch of rest API endpoints).

In regards to future plans, we both agree that in order to truly make our system implementable by the school, we would have to add more functionality to the webpage. This would consist of allowing the housing staff to actually be able to accept or deny the submitted applications, as well as potentially exporting all of the recorded data to an Excel spreadsheet for future use. Eventually, we would like to be able to sell this system to other colleges, once it is polished.