Plan.ly

CS 673
Team 1 - UNO
Iteration 2

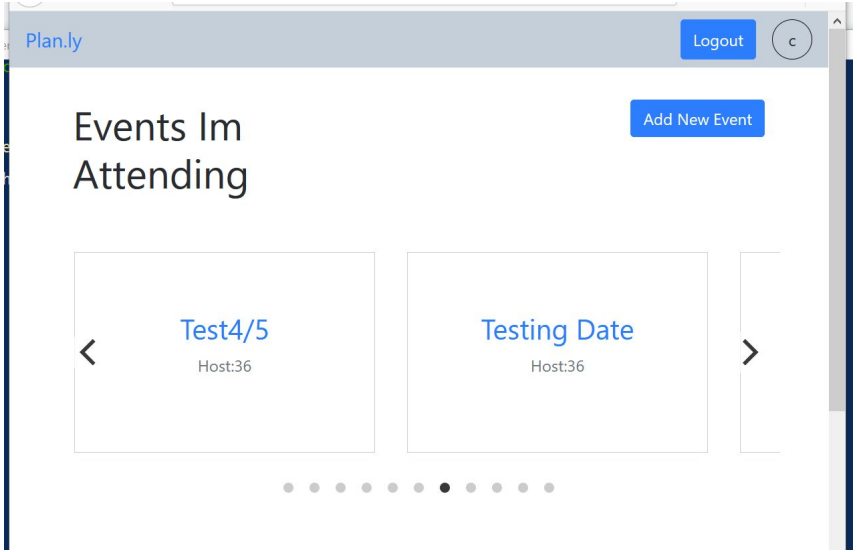
Team Overview

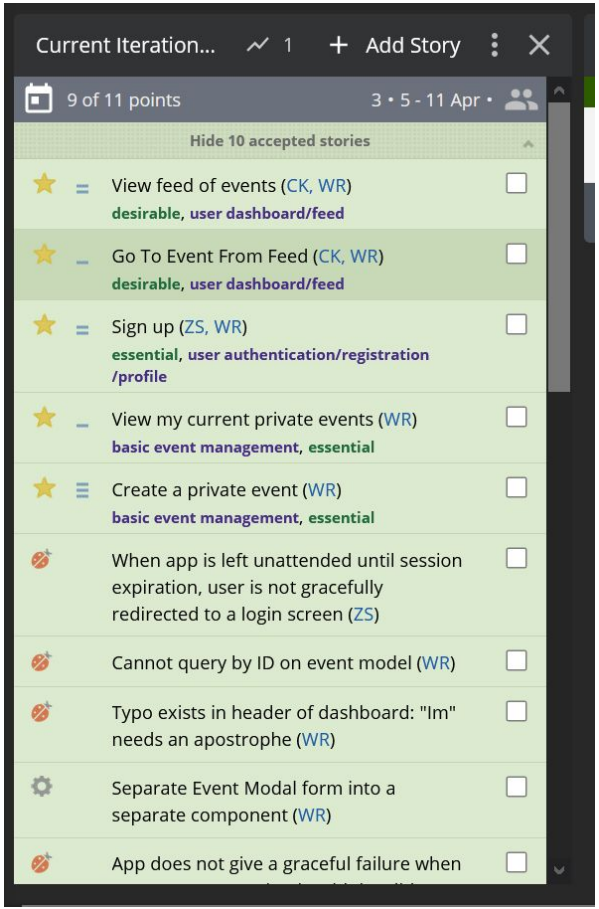
- Team **UNO**
- Working to build Plan.ly, an event planning tool

Name	Role
George Wright	Team Leader
Vibhu Bhatia	Backup Team Leader
Aysha Zenab Kenza	Security Leader
Karen Sommer	Design/Implementation Leader
Zach Schandorf-Lartey	Configuration Leader
Chris Kulig	Requirements Leader
Matt Dowding	Quality Assurance Leader

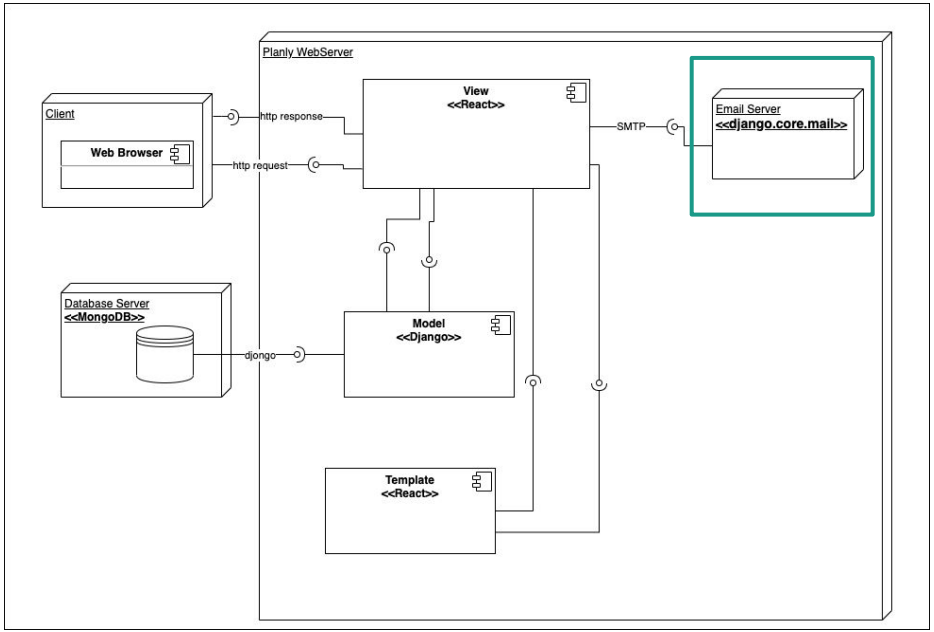
UNO

Iteration 2 User Stories





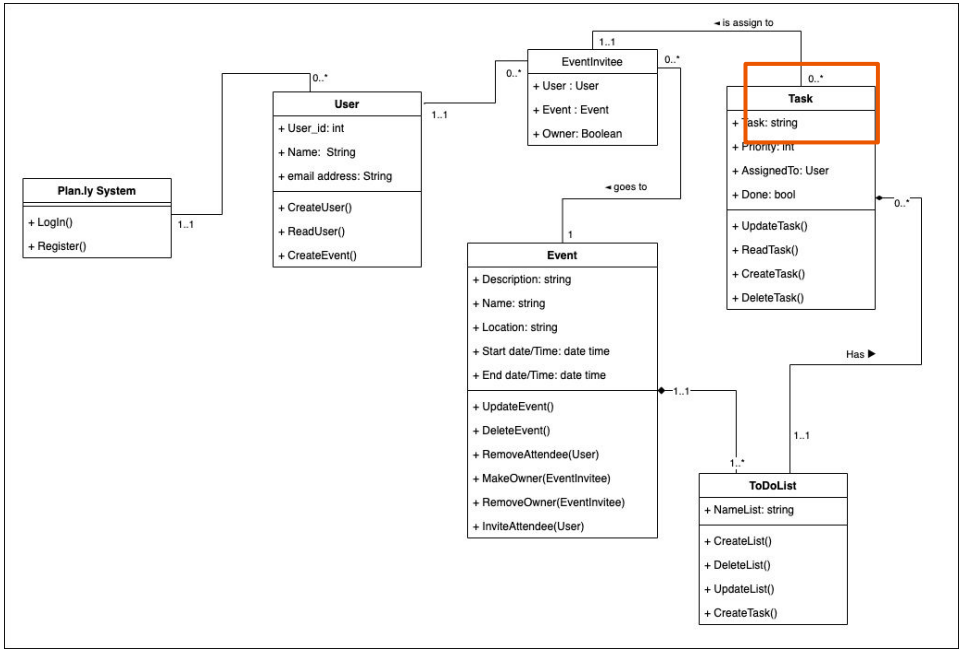
Design -Changes



Component Diagram

UNO

Design -Changes

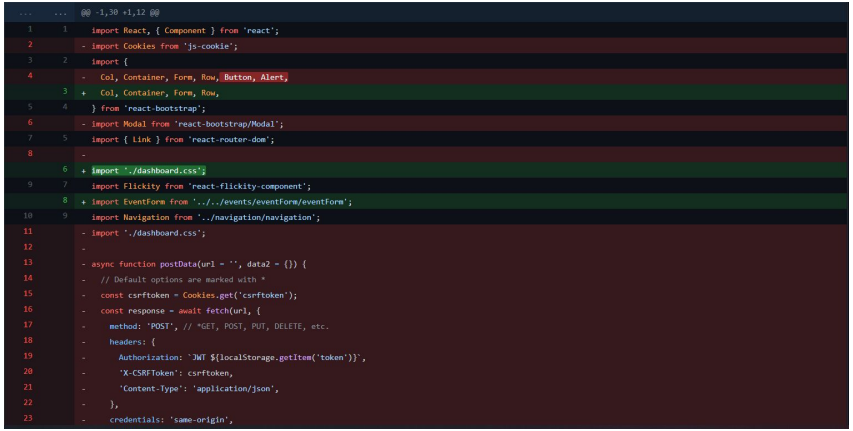


Class Diagram

UNO

Implementation: Refactoring

We needed to refactor dashboard.js and move the event creation form modal to its own component so that it could be reused. All of the methods related to event creation were moved to eventform.js and it was set up to handle editing events as well



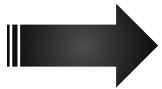
```
@@ -1,30 +1,12 @@
 import React, { Component } from 'react';
-import Cookies from 'js-cookie';
 import {
-  Col, Container, Form, Row, Button, Alert,
+  Col, Container, Form, Row,
 } from 'react-bootstrap';
-import Modal from 'react-bootstrap/Modal';
 import { Link } from 'react-router-dom';
-
+import './dashboard.css';
 import Flickity from 'react-flickity-component';
+import EventForm from '../../events/eventForm/eventForm';
 import Navigation from '../navigation/navigation';
-import './dashboard.css';
-
-async function postData(url = '', data2 = {}) {
-  // Default options are marked with *
-  const csrftoken = Cookies.get('csrftoken');
-  const response = await fetch(url, {
-    method: 'POST', // *GET, POST, PUT, DELETE, etc.
-    headers: {
-      Authorization: `JWT ${localStorage.getItem('token')}`,
-      'X-CSRFToken': csrftoken,
-      'Content-Type': 'application/json',
-    },
-    credentials: 'same-origin',
```

Event form

Implementation: Refactoring

Before

Getting all stored users' emails in the template component



After

Accessing to all users' emails in the view component

- Security
- Scalability

Testing: Non Functional Requirements

- Security
- User Interface
- Complexity
- Scalability

Unit testing

For unit testing, we have made use of the Django built-in unit testing mechanism in tests.py. In this file, we have checks to verify that we can create and delete model records with the expected successes and validation exceptions being returned by the back-end. Example tests include:

- Checking to make sure that the id field of the User model cannot be assigned by a developer; only the back-end
- Testing that Event Invitee records can be created and tied to an event/user using the respective ID attributes of each
- Testing behavior when task records are given extremely long names/descriptions (exceeding the validation requirements)
- Checking that no ValidationError is raised when an Event record is created with all required field filled in correctly.

System testing

System Tests. Our system testing approach relies upon Selenium, which we have imported through its Python module. The system testing is designed specifically around the requirements we have stored in PivotalTracker: in the spreadsheets referred-to below, we specifically map each test case to the requirements it exercises. We aim to automate each and every single one of our test cases. By doing so, we will ensure they can be quickly run, and serve as a valuable tool to quickly test changes to our app and ensure nothing has regressed. Samples of system test cases include (but are not limited to):

- Sign up for a new Plan.ly account and ensure you can login
- Attempt to login with credentials known to be invalid and ensure a warning is given
- Sign in, navigate to an Event page, and attempt to edit the event; ensure the details are updated on the event page once “Save” is clicked on the edit form

Testing Metrics

Metric ID	Metric Description	End of Iteration 0	End of Iteration 1	End of Iteration 2
M-1	Testing coverage	0	4	13
M-2	Unit tests written	0	0	7
M-3	Unit tests passing	0	0	7
M-4	System test written	3	4	6
M-5	System test simulated	0	4	6
M-6	System tests passing	0	4	6

Management & Metrics

- As implementation has gotten more intense (with more code being written), we have instituted an addition group meeting with more ad-hoc meetings between
- Focus of the additional meeting is usually refactoring - this has to be communicated widely so that developers are all up-to-speed on any changes to code architecture.

Metric ID	Metric Description	End of Iter 0	End of Iter 1	End of Iter 2
M-1	Open bug count	0	0	3
M-2	Closed bug count	0	0	4
M-3	Stories/bugs accepted	0	1	11
M-4	Rejection rate	0	0	0
M-5	Number of screens	0	5	6
M-6	Testing coverage	0	4	13

Lessons Learned

- if time will allow, try to have a design review after every iteration.
- It’s easy to design a test to pass but it’s more difficult to design a good test that is full proof.
- In a project, it’s more important to be technology agnostic. Need to select the best solution that works for you.
- Learning can be exponential in pair programming.
- Modularizing user stories into small/incremental parts is hard - stories that seem small can have more to them than what is anticipated.
- Git version control is a gift to programmers
- Need to Budget expectations and complexity of features. Always work on small incremental changes that can be deployed. Overwork can lead to degraded quality of codebase, frustrations and if working in group - lack of communication.

Thank you

Questions?