


CS673 Software Engineering
Team 1 - UNO
Plan.ly
Project Proposal and Planning

<u>Team Member</u>	<u>Role(s)</u>	<u>Signature</u>	<u>Date</u>
George Wright	Team Leader		2/16/2021
Aysha Zenab Kenza	Security Leader	<u>Aysha Zenab Kenza</u>	<u>2/16/2021</u>
Vibhu Bhatia	Backup Team Leader	<u>Vibhu Bhatia</u>	<u>2/17/2021</u>
Karen Sommer	Design and Implementation Leader		02/16/2021
Zach Schandorf-Lartey	Configuration Leader	<u>Zach S-Lartey</u>	<u>2/16/21</u>
Chris Kulig	Requirements Leader	<u>Christopher Kulig</u>	<u>2/17/2021</u>
Matt Dowding	Quality Assurance Leader	<u>Matthew Dowding</u>	<u>2/17/2021</u>

Revision history

<u>Version</u>	<u>Author</u>	<u>Date</u>	<u>Change</u>
0	Group	2/17	Initial Version
1	Group	3/9	Increased estimation for essential features, added estimation for next iteration, refined non-functional requirements, added comment about automated testing
2	Group	4/7	Updated metrics, test spreadsheets, risk register, estimates/plan for next iteration

3	Group	4/26	Final metrics/test updates for iteration 3
---	-------	------	--

[Overview](#)

[Related Work](#)

[Proposed High level Requirements](#)

[Functional Requirements](#)

[Nonfunctional Requirements](#)

[Security requirements - OAuth](#)

[Django](#)

[React](#)

[MongoDB](#)

[Management Plan](#)

[Process Model](#)

[Objectives and Priorities](#)

[Risk Management](#)

[Monitoring and Controlling Tools and Mechanisms](#)

[Timeline](#)

[Quality Assurance Plan](#)

[Metrics](#)

[Standard](#)

[Inspection/Review Process](#)

[Testing](#)

[Defect Management](#)

[Configuration Management Plan](#)

[Configuration items and tools](#)

[Change management and branch management](#)

[Code commit guidelines](#)

[Integration and deployment plan](#)

[References](#)

[Glossary](#)

1. Overview

Team 1 will be creating an Event Planning tool called Plan.ly for our CS673 project. The tool will allow any user to create an account, create events, track event details, invite others (if they are not yet registered, they will receive an email invitation to register), assign tasks, and

post messages to a message feed. The tool's envisioned userbase is anyone--professionally or otherwise--needing to manage the details surrounding event planning and preparation. We envision our tool will add the most value to larger events involving 20+ attendees where professional event planners are not involved. However, we hope our tool can be used by any and all event planning parties with a minimal UI and covers their functional needs.

Sample use cases: holiday potluck dinner; Thanksgiving with extended family; sports team BBQ.

Plan.ly will be developed using modern Python and Javascript web technologies, with separate codebases for the front- and back-end components. We will host it using a free plan on Heroku; this allows for scaling out as needed to support increased demand.

2. Related Work

There are many event planning websites and applications currently available. The most related services found to date include Facebook[1] events, Eventbrite[2], and Zkipster[3]. Currently, Facebook offers the most similar experience as it has a light, simple to use feel, and is geared toward non-professional users. Most other event/planning services are focused on more complex features like visual seating charts, ticketing, and guest check-in – that is to say, they are more focused on the professional planner.

The area where Plan.ly has an advantage is in the subset of features for controlling “To Do’s” related to the event that guests can contribute to. While messages can be sent through Facebook there is no structured mechanism for the host to keep track of action items or for guests to respond to them.

Additionally, when compared with professionally focused applications, this project has an advantage as it is (currently) free. While we intend to maintain some core functionality that exists in many of the competitor applications our primary user/customer has not indicated a need for things like custom email templates or event check-in. Please see the below table for insight into comparable features across various products.

Product Offering	Eventbrite	Facebook Events	zkipster
Free/Paid	Free, Paid % of ticket revenue	Free	Paid
Focus	Professional, Casual, Other	Casual, Other	Professional
Target Market	All event types	FB Users	Luxury/Corporate/Professional
Event Types	Live Event, Catch-all	Catch-all, Ticketed via Eventbrite	Luxury, fashion, non-profit, professional sports, arts, entertainment
Complexity	High	Low	Medium
Competitor Fit	Medium/High	High	Low
Features			
Guest List/Check-in	Yes		Yes
Seating Chart	Yes		Yes
free promotion	within Eventbrite ecosystem	Yes	
Online Invitation	Yes	Yes	Yes
Multi Day Planning	Yes	Yes	Yes
Ticketing	Yes		
Social Media Integration	Yes	Yes	
Structured Subtask management			
Built in platform commenting	Yes	Yes	

3. Proposed High level Requirements

a. Functional Requirements

The following [link](#) contains all the functional requirements we created in PivotalTracker. We built our stories with a title that describes the purpose of the story to be as intuitive as possible. For the description, the team used the format: As (a role), *I want to* (action), *so that* (value).

Our functional requirements are classified into essential, desirable, and optional features. For every story, we have provided a rough estimation in terms of required person-hours.

i. Essential Features

The essential features are marked as such in PivotalTracker and are stories that support our basic application functionality. In the following table, are the essential features with initial time estimates a person will need to complete implementation. (person-hours estimation).

Essential feature	Person-hour estimation
Event owner - complete list ownership	6 hours
Manage Event User Roles	2 hour
Sign up	4 hours
Event Owner ability to create Invite List	4 hours
View my private event	8 hours
Create a private event	13 hours
Receive Account Creation Email Notification	12 hours
Log in	4 hours
Event owner - Manage event to-do lists	6 hours
Email notification of event invitation	2 hours

ii. Desirable Features

The desirable features are the requirements that are nice to have. The ones the team wants to have. Below you can see the list of the desirable features.

User stories - Desirable features
Go To Event From Feed
Event Owner -- Create new task lists
Event Owner -- Edit Event Description
Event Owner -- Event Co-owner(s)
Event Invitee - ownership over tasks
Event invitee - Manage to-do list items
Change Password
Manage Contact Information
History private events
Delete User
View feed of events
Delete private event
Update my private event

iii. Optional Features

The optional features are the features that are cool to have--these are the ones the team will do if there is time to implement them. In this [link](#), it is important to highlight that we created labels (optional, essential, and desirable) for filtering in an easier way the stories.

b. Nonfunctional Requirements

I. **Security:** *Passwords of application user accounts need to meet minimum complexity requirements.*

II. **User Interface:** *The application should be usable on a variety of devices and be built responsively.*

As a user of this application, I want all content to be shown on the screen with no left-to-right scrolling required if I am on a 1080x1920 display so that I do not get frustrated by useless scrolling.

As a user using a mobile device, I want the application to be re-rendered appropriately based on the device I am using so that I can use Plan.ly on the go.

- III. **Complexity** : *The application flow should require as few clicks as possible to navigate through the screens available to each user.*

As a website user, I want to be able to reach all screens I am authorized to use within 6 clicks so that I do not get confused while using the application.

- IV. **Scalability**: *The application should be able to handle a large number of records in the database across all entities.*

As a Plan.ly product manager, I want the application to be capable of storing information for at least 10,000 events, 10,000 users, 100,000 event invitations, 50,000 task lists, and 100,000 tasks so that my application can scale through its initial roll-out amongst pilot users.

4. Management Plan

a. Process Model

UNO will be using a modified SCRUM model [4], effectively working in one week sprint cycles, with our Sunday meetings being both our planning and review meeting. We won't have retrospectives or daily standups, but the emphasis will remain on rapid testing and iteration. Any urgent issue requiring immediate attention will be dealt with over Slack.

b. Objectives and Priorities

- i. Deploy all essential features in a working codebase tested to 100% acceptance.
- ii. Ensure that user authentication mechanisms are in adherence to industry standards and do not risk compromising user login/password information.
- iii. Design and implement a minimal user interface with a consistent look and feel across all screens.
- iv. Adhere to best practices for Quality Management including use of GitHub repository, documentation of test cases, and reliance upon CI/CD pipeline.
- v. Encourage the learning process by having team members commit code both in areas where they have prior experience and in areas that are new.

c. Risk Management

There is a broad range of experience across our team when it comes to technical abilities implementing Django and React. For this reason, a big risk is that some of the more time-intensive user stories will fall to a subset of team members to implement.

Our team will be managing this risk by:

- i. Hosting workshops for less experienced team members to become familiar with frameworks and the deployment process
- ii. Assigning the most crucial/essential tasks to experienced engineers in earlier iterations of the project. This will leave smaller/easier tasks for newer engineers as the project goes on.

Our risk management sheet can be found at the following [link](#).

d. Monitoring and Controlling Tools and Mechanisms

We will use the following tools to facilitate group communication and monitor the project progress.

- i. PivotalTracker Link: <https://www.pivotaltracker.com/n/projects/2487101>
- ii. Slack Link: BUMETCS673S21, channel: #project-group-1
- iii. Github Link: <https://github.com/BUMETCS673/BUMETCS673S21T1>
- iv. Zoom meeting Link: <https://bostonu.zoom.us/j/2024473071>
- v. Weekly meeting time:
 1. Primary: 2 PM, Sundays
 2. Secondary, if needed: 9 PM, Mondays

e. Timeline

Iteration	Functional Requirements (E/D/O)	Tasks	Estimated/real person hours	Presentation Recording Link (5-10 minutes)
1	<ul style="list-style-type: none"> Register Login Create/delete event View events 	<ul style="list-style-type: none"> Setup mongoDB Select/implement UI toolkit Finalize design Decide/implement security protocol 	125 hours across team of 7 people/ 214.5 actual	N/A

		<ul style="list-style-type: none"> Initial version of end-to-end tests written 	person hours	
2	<ul style="list-style-type: none"> Create to-do list (CRUD) Edit event details (CRUD) Send email notifications (registration and invitation) 	<ul style="list-style-type: none"> Integrate email service 	125 hours across team of 7 people	N/A
3	<ul style="list-style-type: none"> Task lists Invitee management Finish open stories from iteration 2 	<ul style="list-style-type: none"> Define all new APIs Refactor event search behavior 	125 hours across team of 7 people	N/A

5. Quality Assurance Plan

a. Metrics

- i. The objective of this section is to define what metrics we will be using, how we are going to keep track of them, and how we are going to analyze them to improve. This section has two types of metrics: product metrics and process metrics. For each of the metrics, we included product complexity (LOC, # of files, # of classes, # of methods, etc.) cost (in terms of man-hours), defect and defect fix rate, etc.

Metric ID	Metric Type	Metric Description	Process Improvement
M-1	Product	Open bug count	Analyze bugs for trends; communicate to developers for continuous improvement.
M-2	Product	Closed bug count	This metric will be an indication of the stability of the product. If it increases, additional time will be spent on debugging rather than creation of new code.

M-3	Process	Stories accepted	This metric will serve as an indication of team productivity. If it decreases, additional time will be spent bringing open stories to completion before new stories are started.
M-4	Process	Rejection rate	Use this metric to gauge quality management. If the rejection rate of stories increases, analyze sources of bugs for trends.
M-5	Product	Number of screens	Use number of screens metric as a gauge for complexity of product as project continues.
M-6	Process	Testing coverage	Use testing coverage metric to be most efficient in adding of unit tests to codebase.

ii. Results (to be completed at the end of each iteration)

Metric ID	Metric Description	End of Iteration 0	End of Iteration 1	End of Iteration 2	End of Iteration 3
M-1	Open bug count	0	0	3	12
M-2	Closed bug count	0	0	4	10
M-3	Stories/bugs accepted	0	1	11	24
M-4	Rejection rate	0	0	0	0
M-5	Number of screens	0	5	6	6
M-6	Testing coverage	0	4	13	24

b. Standard

The development branch will have set up and explanation documentation for developers to read when they branch off of it.

Code must adhere to standards outlined by the pylint linter (PEP-8) [7], for Python code, and eslint [9], a JavaScript linter that will be configured to follow Airbnb's JavaScript standards [8].

Code should be thoroughly commented so that functions describe their purpose, making it easier for reviewers to look over pull requests.

c. Inspection/Review Process

Our team will use a CI/CD pipeline that does not allow code to be merged to the main branch without going through a pull request process. The pull request will have the team leader, backup team leader, and configuration leader added as reviewers by default. Only after the pull request is reviewed (ideally in a group setting) will the code be pushed to the main branch for deployment to Heroku.

At a minimum, our team's expectation is that at least one unit test is included per function that is being merged to the main branch. Commits that do not meet this standard are subject to rejection from the pull request.

d. Testing

Our team will be adhering to the following SCRUM standards for quality management throughout our project.

- i. Requirement Analysis: make sure the requirements are clear, consistent, complete, traceable, and testable. This way they prevent possible software defects and facilitate upcoming test design activities.
- ii. Test Design: design test cases or checklists covering software requirements. Test cases outline conditions, test data, and test steps needed to validate particular functionality, and state the expected test results.
- iii. Execute Testing: starts at the unit level, when the development team performs unit testing. In its turn, the test team takes over at the API and UI levels. Manual test engineers execute the designed test cases, submitting found defects in a defect tracking system, while test automation engineers use a selected framework to execute automated test scripts and generate test reports.
- iv. Report Defects: After each test if run any bugs are logged and reported. Each issue gets a priority level from urgent to low, which the development team then resolves based on time and who is available.
- v. Re-Test: Once the found defects are fixed, functionality in question is retested and a regression testing is performed to make sure that bug fixes neither

broke the related functionality nor made it different from that specified in the requirements. Once verified the bug may be 'closed' in the bug tracking list.

- vi. Release Testing: The QA team must perform build verification testing to ensure the build is stable. If the test passes, then modified tests are run, and a report is generated at the end.

It is the responsibility of each individual developer to author and implement their own automated unit test cases each time they are working on stories from the backlog. End-to-end testing will be managed and carried out by the Quality Assurance Lead on an ongoing basis (i.e. weekly). A subset of the end-to-end tests will be selected for automation and simulated with Selenium. From the end-to-end testing process, we will develop a list of defects that will be tracked and closed in PivotalTracker using "Bug" stories.

The Quality Assurance Leader will be responsible for updating the following two documents on an ongoing basis as stories are implemented:

- i. [End-to-End Test Cases](#): This document will keep an inventory of all functionality tests that are used on an ongoing basis to ensure the app provides all required functionality. Additional columns will be added each time tests are carried out. If failures emerge from this testing, they will be entered as "Bug" stories in PivotalTracker.
- ii. [Requirement Testing Coverage](#): This document will map all test cases to the requirements/stories they exercise in order to ensure all stories are being adequately tested. The list will be based on a story export from Pivotal, and the Quality Assurance leader will be responsible to identify those test cases that exercise the functionality specified in each story.

e. Defect Management

- i. After first QA testings are run any bugs occurring need to be logged, communication then needs to occur to those developing and those bugs should be worked on, testing will then go to the retesting phase to check for any new issues / clear the initial bugs. As bugs show up they should be entered into pivotal and the QA Leader is responsible for filling out and keeping track of the bugs

6. Configuration Management Plan

a. Configuration items and tools

- i. Items:
SPPP, Meeting Minutes, Progress Report Spreadsheet, Iteration 0 Presentation
- ii. Tools:

Git[11], Github[10], Github Actions[12], Pivotal Tracker

b. Change management and branch management

Our team will use the following Github branching structure:

```
|---main
|
|---development
|  |
|  |---feature/feature-name
|  |---fix/fix-name
|  |---branch-name
|
|---fix/fix-name
```

A description of each of the above branch types is as follows:

- i. Main: release branch. Most stable version of project, should always be ready for deployment
- ii. Development: stable working branch, Merges into main
- iii. Feature/branch-name: working branch for a specific feature. Merges into development
- iv. Fix/fix-name: Hotfix working branch for development. Merges into development
- v. fix/fix-name (from main): hotfix for any deployed releases. Merges into main

In order to ensure effective quality management, the iteration process that each engineer will follow is:

- i. Pull from deployment branch
- ii. Create a new feature/* branch from deployment
- iii. When feature is completed, create pull request to merge feature branch into deployment
- iv. Once pull request is approved, it will be merged into deployment branch
- v. At the end of each iteration (before the presentation), deployment will be merged into main

We will accomplish change management through the use of pull requests; all promotions of code to the main branch are required to be approved using a pull request. The pull request will be the opportunity to ensure the team's standard of at least one unit test per function is adhered to in all submitted code.

c. Code commit guidelines

For code being pushed, functions should have comments that describe their purpose, making it easier for reviewers to look over pull requests.

Pull requests should outline what has been added/changed in the branch being merged

Branches should adhere to the naming conventions outlined in section b.

d. Integration and deployment plan

Team 1 will be using Github Actions to manage CI/CD. This will enable the following automatic safeguards and processes:

- i. Every pull request made to the deployment branch will automatically add George, Zach, and Vibhu as reviewers.
- ii. Pull requests will be automatically linted with pylint and eslint and if the code being reviewed doesn't pass, the pull request will fail, which will allow the author of the pull request to go back and update the code in the request

Our team will use the Sunday before each iteration presentation to merge our deployment to the main branch which will be deployed to Heroku [13] with each update. This will allow three days for any hotfixes before the presentation. The main branch will not have any merges during this three day period, unless they are fixed.

7. References

Related work resources as follows:

- [1] *Facebook*. (2004). Facebook. <https://www.facebook.com/>
- [2] *Eventbrite*. (2006). Eventbrite. <https://www.eventbrite.com>
- [3] *zkipster: Online Event Management Software For Event Planners*. (2020, December 18). Zkipster. <https://www.zkipster.com/>
- [4] Braude, E. J., & Bernstein, M. E. (2016). *Software Engineering*. Amsterdam University Press.
- [5] A. (2012). *OAuth 2.0 Authorization Framework*. Auth0 Docs. <https://auth0.com/docs/protocols/protocol-oauth2>
- [6] *Django introduction - Learn web development | MDN*. (2021, February 9). Django Introduction. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>
- [7] L. (2019). *Pylint - code analysis for Python | www.pylint.org*. Pylint. <https://www.pylint.org/>
- [8] A. (2021). *airbnb/javascript*. Airbnb Linting Standard. <https://github.com/airbnb/javascript>
- [9] *Eslint*. (2012). Eslint. <https://eslint.org/>
- [10] *GitHub: Where the world builds software*. (2008). GitHub. <https://github.com>

- [11] *Git*. (2008). Git. <https://git-scm.com>
- [12] *Features • GitHub Actions*. (2008). GitHub Actions. <https://github.com/features/actions>
- [13] *Cloud Application Platform | Heroku*. (2007). Heroku. <https://www.heroku.com>
- [14] MongoDB. (2009). *The most popular database for modern apps*. <https://www.mongodb.com/3>
- [15] Baer, E. (2021). *What React Is and Why It Matters*. O'Reilly Online Learning. <https://www.oreilly.com/library/view/what-react-is/9781491996744/ch01.html>

8. Glossary

CI/CD: Combined practices of continuous integration and either continuous delivery or continuous deployment.

Linter: Lint, or a linter, is a static code analysis tool used to flag programming errors, bugs, stylistic errors, and suspicious constructs

Git: Git is a software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.

SCRUM: Scrum is a framework for project management that emphasizes teamwork, accountability, and iterative progress toward a well-defined goal