SlimeLine - Sprint Plan 1

KEY:
Bora - green
Ronith -
Nicolas - yellow
Mohan - red
Jack - blue

At the end of your team's sprint planning meeting, the team needs to turn in a sprint plan. This document needs to be typewritten (or the team needs to use a web-based agile planning tool and provide the TA/tutor access to the tool to view the project) and have the following elements:

- **Heading:** Document name ("Sprint {number} Plan"), product name, team name, sprint completion date, revision number & revision date.

- **Goal:** Short, 1-2 sentence description of the high-level goal(s) for the sprint. • **Task listing, organized by user story:** This section lists the user stories, in priority order from most important (top) to least important (bottom). Within each user story, there needs to be a list of tasks required to implement the user story, along with the time estimate for each tasks (preferably less than or equal to 6 ideal hours). This should look like:

  User story 1 ("As a {user role}, I want {goal} [so that {reason}]")
  Task 1 description (time estimate for task 1)
  Task 2 description (time estimate for task 2)
  ...
  Task N description (time estimate for task N of user story 1)
  Total for user story 1: XX hours

Foundational steps, making website, user auth, basic posts

1. As a guest, I just want to see what's new
    a. Task 1 - create a simple dynamic web page
        i. Google cloud run as the webserver
    b. Task 2 - create premade posts in html to be served dynamically
        i. Watch youtube tutorials
        ii. Setup basic template: name of club/event, timestamp, etc
    c. Task 3 - just have regular user auth (display that you are logged in)
        i. Watch youtube tutorials
        ii. Prompt to proceed as guest, student, or official ucsc clubs/events account
            1. Roles: student, club
            2. Guest becomes default
2. As a club owner, I want to have an official club account to make events/info posts (includes post modifications/deletions)

a. Task 1 - implement user auth so that clubs can post
b. Task 2 - ensure that only special emails have access to “Make a post” button
c. Task 3 - “Make a post” button work → posts some text to the database, the webserver can then dynamically serve these raw posts for users to see “what’s new”
    i. Click a button, form pops up, fill out form to create post (watch YT tutorial)
    ii. Post deletion

- **Team roles:** Give a listing of all team members. Next to the team member, list their role(s) for this sprint. Assign each person to at least one role (for example, this role might be "Developer"). This looks like:

  Team member 1: role 1 {, role 2, role 3}
  Team member 2: role 1 {, role 2, role 3}
  ...
  Team member N: role 1 {, role 2, role 3}
- **Initial task assignment:** A listing of each team member, with their first user story and task assignment. This should look like:

  Team member 1: user story, initial task
  Team member 2: user story, initial task
  ...
  Team member N: user story, initial task

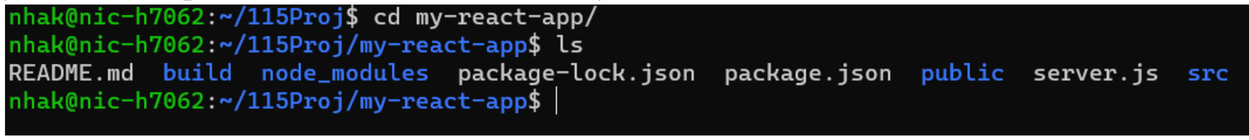
- **Initial burnup chart:** A graph giving the initial burnup chart for this sprint and is labeled as such with sprint number and project name and is located in the lab.

- **Initial scrum board:** Also known as a task board, the scrum board is a physical board and labeled as such with sprint number and project name and located in the lab. This board has four columns, titled user stories, tasks not started, tasks in progress, and tasks completed. Index cards or post-it notes representing the user stories and the tasks for this sprint should be placed in the user stories, tasks not started, and tasks in progress columns. Tasks associated with a user story should be placed in the same row as the user story.
- **Scrum times:** List at least the three days and times during the week when your team will meet and conduct Scrum meetings. Also, indicate which of these meetings will have the TA/tutor visit as arranged with the TA/tutor. It is expected the TA/tutor will visit during the Scrum meeting during your lab time.

Note that if the team ended up modifying its release plan during sprint planning, submit an updated release plan document also with the sprint plan.

Last modified: 10/15/13 adapted from materials for cmps171

{Nicolas} Infrastructure setup:

1. Sudo apt install npm
    a. Install npm which is the javascript package installer
2. npm install react react-dom express
    a. React for front-end, express view engine to render React objects
    b. https://www.npmjs.com/package/express-react-views
3. npm install body-parser
    a. https://www.npmjs.com/package/body-parser
4. npm install axios
    a. Needed to integrate react
5. npm init -y
    a. In the desired directory, run this to set up the environment
6. npx create-react-app client
    a. curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
    b. Used for “nvm install 14” and “nvm use 14”
    c. Node 14 is needed for the npx command to run
    d. If get error nvm command not found, then close terminal, open new one
7. From step 6, there should now be a directory called client
    a. Cd into that directory
    b. Run: npm run build
        i. Creates a build directory
8. Inside your client directory, create a file called server.js
    a. See heading Server.js
9. Inside your src directory “.../client/src”
    a. Edit the App.js file
    b. See heading App.js
10. Ensure your terminal path is now in the client directory
    a. 

```
nhak@nic-h7062:~/115Proj$ cd my-react-app/
nhak@nic-h7062:~/115Proj/my-react-app$ ls
README.md  build  node_modules  package-lock.json  package.json  public  server.js  src
nhak@nic-h7062:~/115Proj/my-react-app$
```

    b. Now run node server.js
    c. You can now go to localhost:5000 and you will see a button that you can push
        i. It was made with React!

Server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');


const app = express();

// Parse incoming JSON data
app.use(bodyParser.json());

// Serve static assets from the React app's build directory
app.use(express.static(path.join(__dirname, 'build')));

// Serve the React app's index.html at the root path
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'build', 'index.html'));
});

// New route for handling GET requests to the root path
app.get('/', (req, res) => {
    res.send('Hello from the server!'); // Or any response you want
});

app.get("/foo", (req, res) => {
    res.send("You've requested the foo page.");
});



const port = process.env.PORT || 5000;
app.listen(port, () => {
    console.log(`Server listening on port ${port}`);
});
```

App.js

```
import React, { useState } from 'react';

function App() {
  const [showHello, setShowHello] = useState(false);

  return (
    <div>
      <button onClick={() => setShowHello(true)}>Say Hello</button>
      {showHello && <p>Hello!</p>}
    </div>
  );
}

export default App;
```