TeamPlayer -

Software Requirements Specification

**Table of Contents**

Product Description…………………………………………………………………………………………….2-4

User Interface Diagrams………………………………………………………………………………………5-9

Use Cases…………………………………………………………………………………………………..……10-15

Process Description……………………………………………………………………………………….…16-19

**I. Product Description**

**What is your product?**

“TeamPlayer” is a group based task manager. Deployed as a web app, users can create and join groups where they can schedule task and manage bills between everyone on their teams. TeamPlayer’s task management system is focused on a monthly main calendar where all groups’ tasks and bills can be created, viewed, and modified. The calendar can be viewed through a filtering system to help the users manage their individual tasks. These tasks can be dateless, where it is viewed separately from the calendar, and even repeat weekly. They can also automatically rotate between members after they are completed. Another main feature of TeamPlayer is the flexible bill management system, where groups can split and manage their bills between the users in the teams.

**Target audience?**

We have rescoped our target audience past roommates to all groups who need a better way to organize themselves, especially task and bills between group members. For example, our product can be used in project/event planning groups to organize each individuals’ tasks, or in groups of friends who always seem to be owing each other money. With Team Player, users can view their events as well as events of other members in their group, and create events involving other people.

**What problem does it solve?**

This application solves communication problems created by poor scheduling, poor reminders, and unorganized finances between groups of people. It will help groups reduce the time needed to communicate with multiple people, thereby decreasing misunderstandings on who is assigned what task and when. It will also help keep people accountable by letting TeamPlayer keep track of money deadlines and totals people owe each other.  Overall, it will help cleanly facilitate group bills and tasks by providing a clean interface to view and assign tasks.

**What alternatives exist, and what are their strengths and weaknesses?**

Alternatives to these problems already exist. A representative application, HomeSlice, manages a social network between roommates to help with communication. HomeSlice seeks to provide incentive for good behavior and transparency in group dynamics. They provide a whiteboard to help with communication between roommates, a supplies manager to help with who needs to buy what and how much is left, and a chores and bills manager to help facilitate tasks and finances. Their application provides a clean interface for a small community, being roommates, to manage basic tasks.  Its advantages are that it provides push notifications to notify community members of tasks they need to do, and provides a clean interface for users to add responsibilities. However, users have complained that the application does not provide the ability to submit one bill for multiple members. Additionally, there’s no way to snooze events.

Another cliche alternative is shared calendars, like Google Calendar. It provides an event based approach in a calendar format. Its advantages include already being widely used, and an integration with other Google software like Gmail. You can create a shared calendar, as well as subscribe to multiple calendars (e.g. a personal one). However, it is weak for groups because it is only task based--there is no concept of “bill management.” It also does not allow for tasks that do not have a specific due date.

Another competitor specific to group management is Asana. Asana allows users to set up task notifications for a group and for members in their group. Their application’s advantages are that it’s very flexible to allow different team’s needs, and its ability to comment on tasks allows for accountability and communication between members. However, their product does not contain specific bill splitting functionality, and it is not a free software when the group is larger than 15 people.

**How will your system be different from others?**

There are a few applications similar to TeamPlayer, such as HomeSlice, Google Calendar, and Asana. However, our application will differentiate itself by providing a calendar view of events to facilitate time-dependant tasks. While HomeSlice provides a lot of services for roommates to organize their group, it does not provide a calendar view that could make the application difficult for paying bills on time or completing tasks on time. Furthermore, HomeSlice specifically targets roommates as a user base and not just groups who need to organize events. Seeing as our application works with general tasks and bills and not specifics like supplies, we can reach a wider audience.

Our application will differentiate itself from Google Calendar by being geared toward groups rather than events. We will differentiate between items such as tasks, bills, events, and todo’s (tasks without a due date). We will also provide a richer set of features such as calculating who owes whom, cycling tasks between a group, and allowing repeat events.

Our application will differentiate itself from Asana with cyclical and rotating tasks. Additionally, our application’s bill functionality will add more functionality to our product. Furthermore, Asana is extremely flexible and allows for calendar views but it’s not the default. Our product will set itself apart by prioritizing the calendar view, allowing functionality for tasks, and allowing functionality for bills.

**What are its major features?**

Main Features:

* view all tasks and bills on the calendar
  + users can filter these events by group
* task creation for individual, group, subgroup
  + tasks contain a name, description, and members assigned to it
  + able to assign each task a due date, which will make the task appear on the calendar
  + able to assign each task no due date, which will appear on your todo list
  + able to set tasks as repeating (happening on specific days of the week) or cyclic (going through each member in the group)
* bill input and bill splitting
  + acts like a task, with specific users, name, description, and due date
  + will have a total cost and ability to split bill evenly amongst members of the bill
  + members can be assigned specific amount they owe on the bill
  + has the same repeating and cyclic functionality of tasks.
* view running list of user’s bills and payments
  + can view bills for selected groups users are a part of
  + can see how much they owe another member

    Stretch Features:

* email notifications
  + notifications could be sent on the day of an event to remind the user what events they have due that day
* group management
  + Currently, groups can be created and new members can be added by anyone in the group, but can be extended to management permissions
* Snooze/Move events
  + Currently, the creator of an event must remove, and create a new event. However, we discussed the ability for anyone whom the task applies to “snooze” the event. This may require additional thought, as the task often dictates when it can be snoozed. For example, a user probably does not want to snooze a rent payment.

**What are its non-functional requirements?**

Our application will require that the user display is up to date when other users have entered new information. This means that our application will need to require consistency and accuracy among databases and updates from other users.  Additionally, since our product will require each user to have their own username and password to access their calendar, our product needs to require security, password hashing and other related measures to protect data.

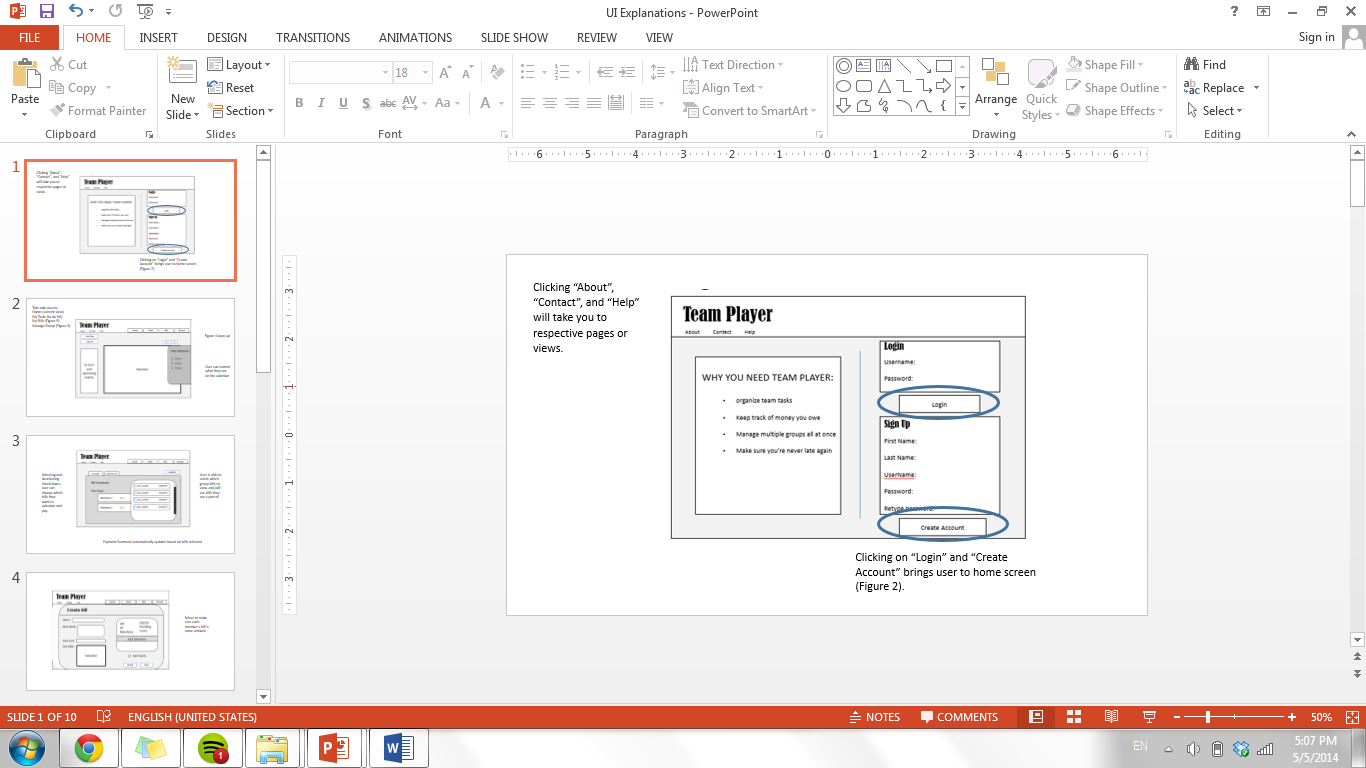
Another non-functional requirement would be performance with a short response time, especially in the calendar and bill viewing interfaces. Selecting and deselecting to filter tasks and bills, as well as calculating payment totals, may need to be efficient.

**What external documentation will you provide that will enable users to understand and use your product?**

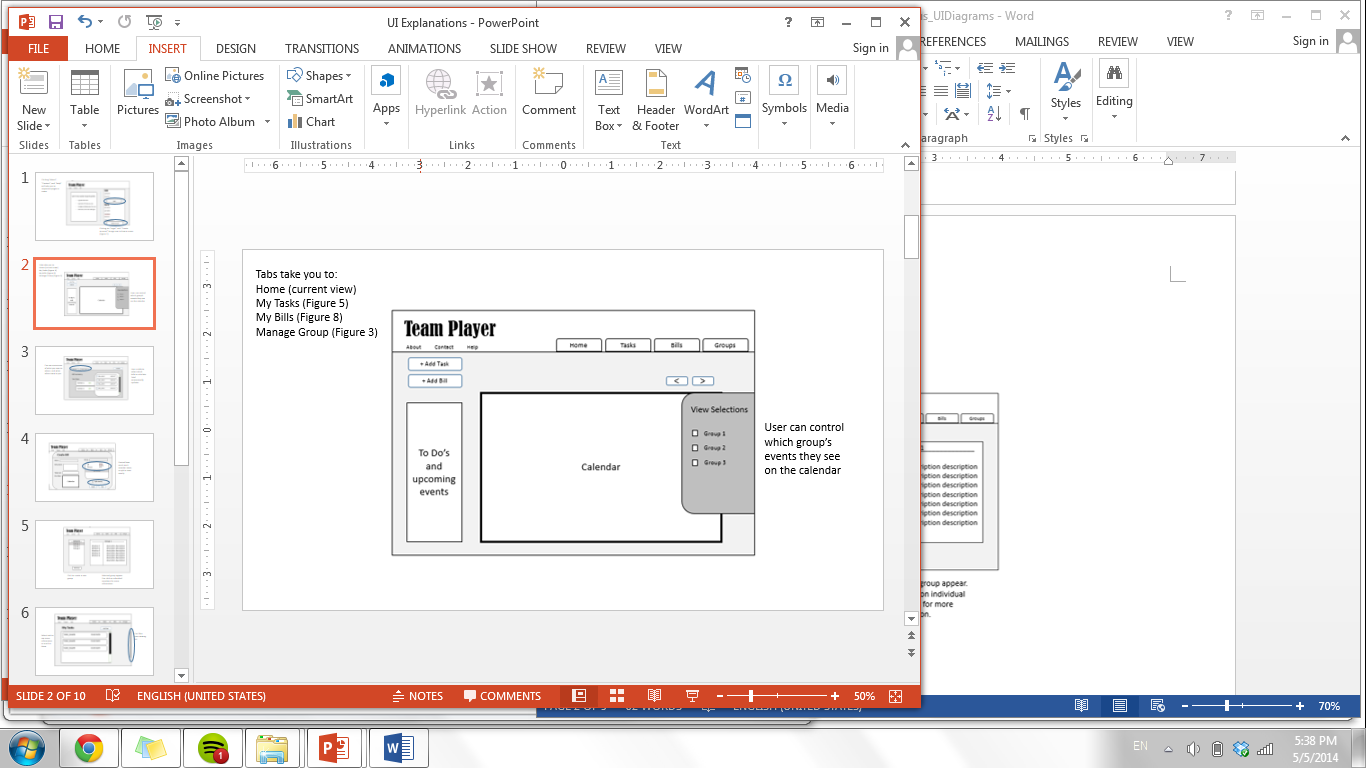
Our product will provide new users an optional step-by-step tutorial on the features of our product when they log in for the first time. From then on, the user will be able to find a help/FAQ guide if they run into problems.  Our main targets for understanding our product will be on the different functions, so when a user first uses one of the functions they will be informed how it work.

**II. User Interface Diagrams**

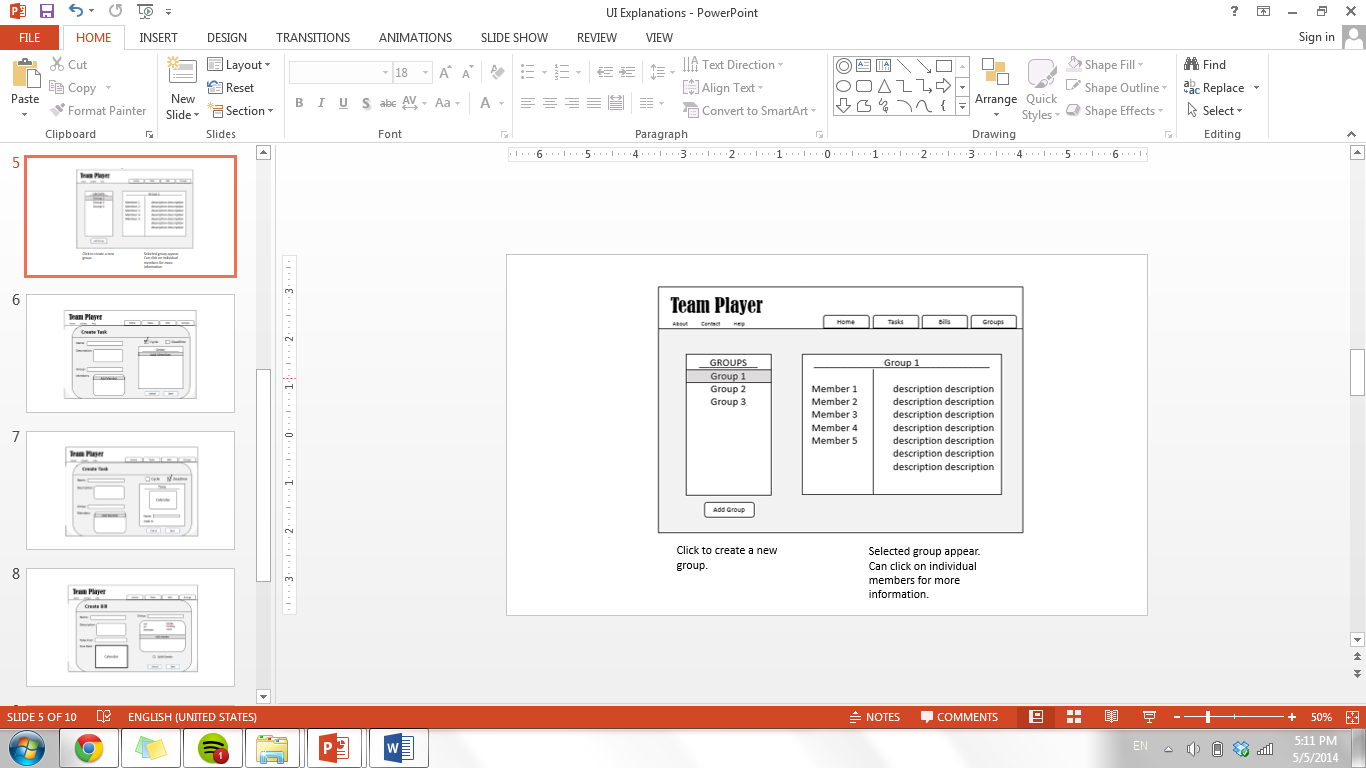
**Figure 1. Login View**



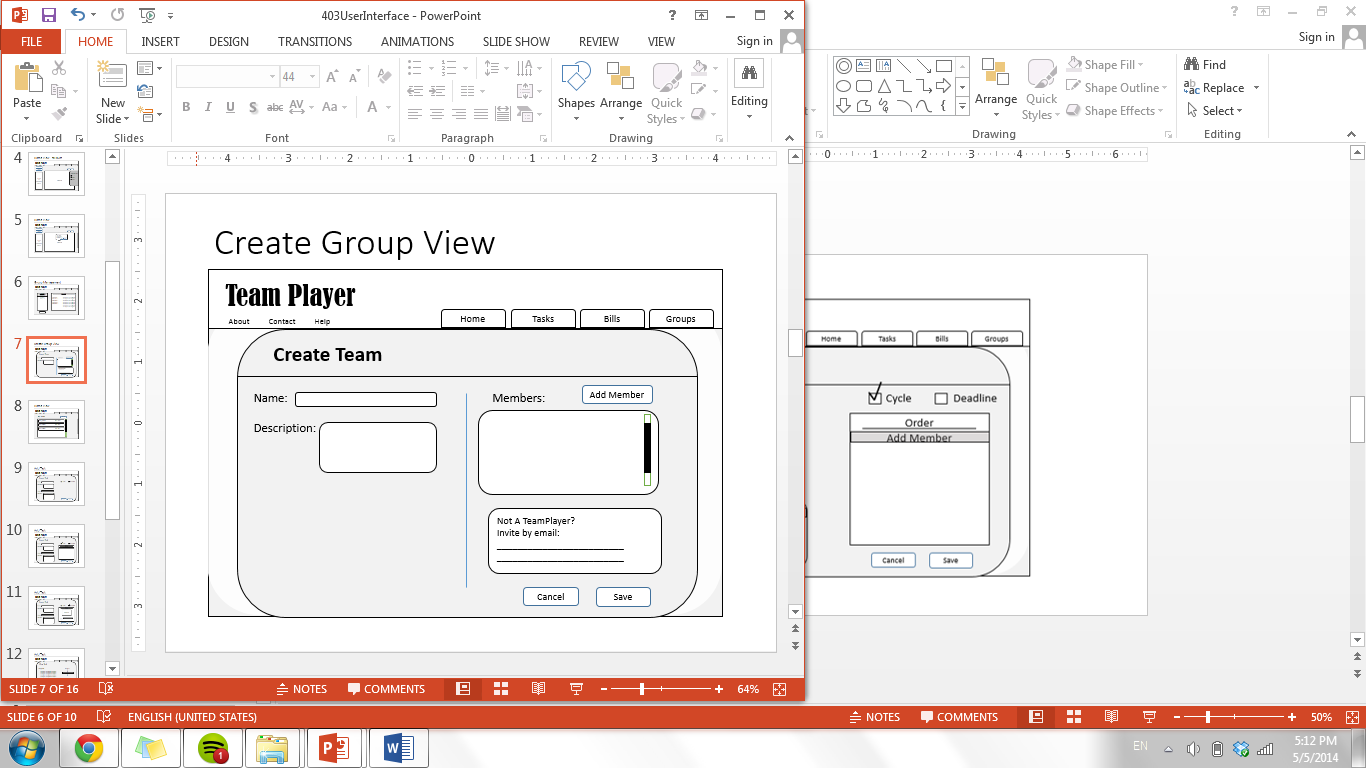
**Figure 2. Home View**



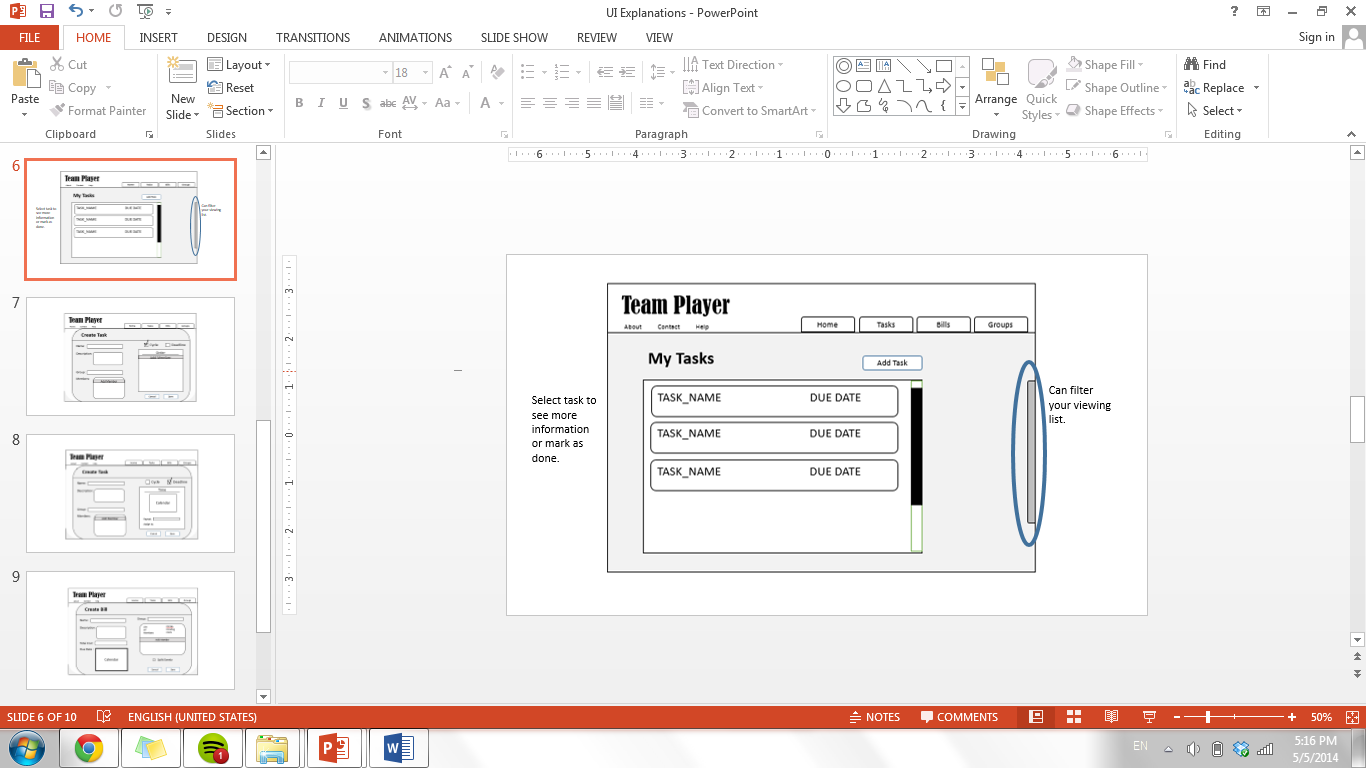
**Figure 3. Group Management View**



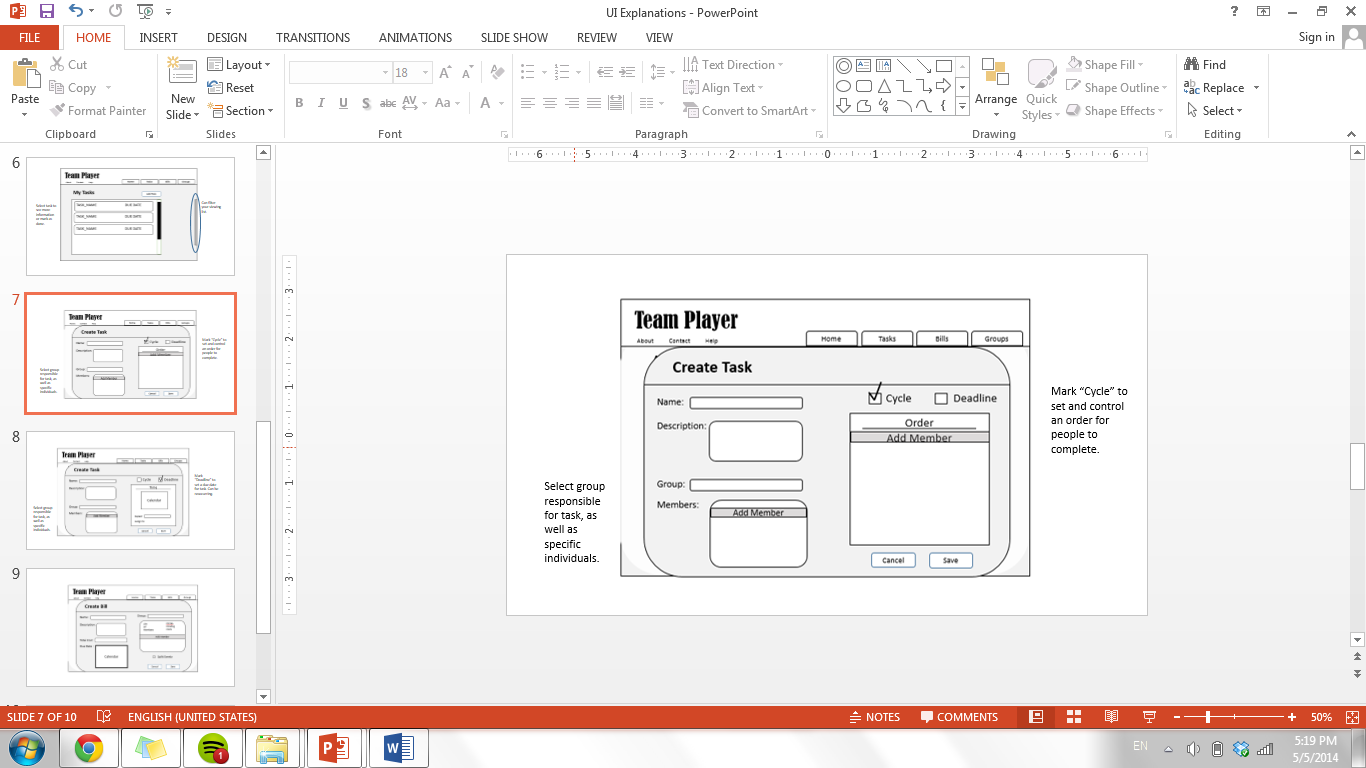
**Figure 4. Group Creation/Edit View**

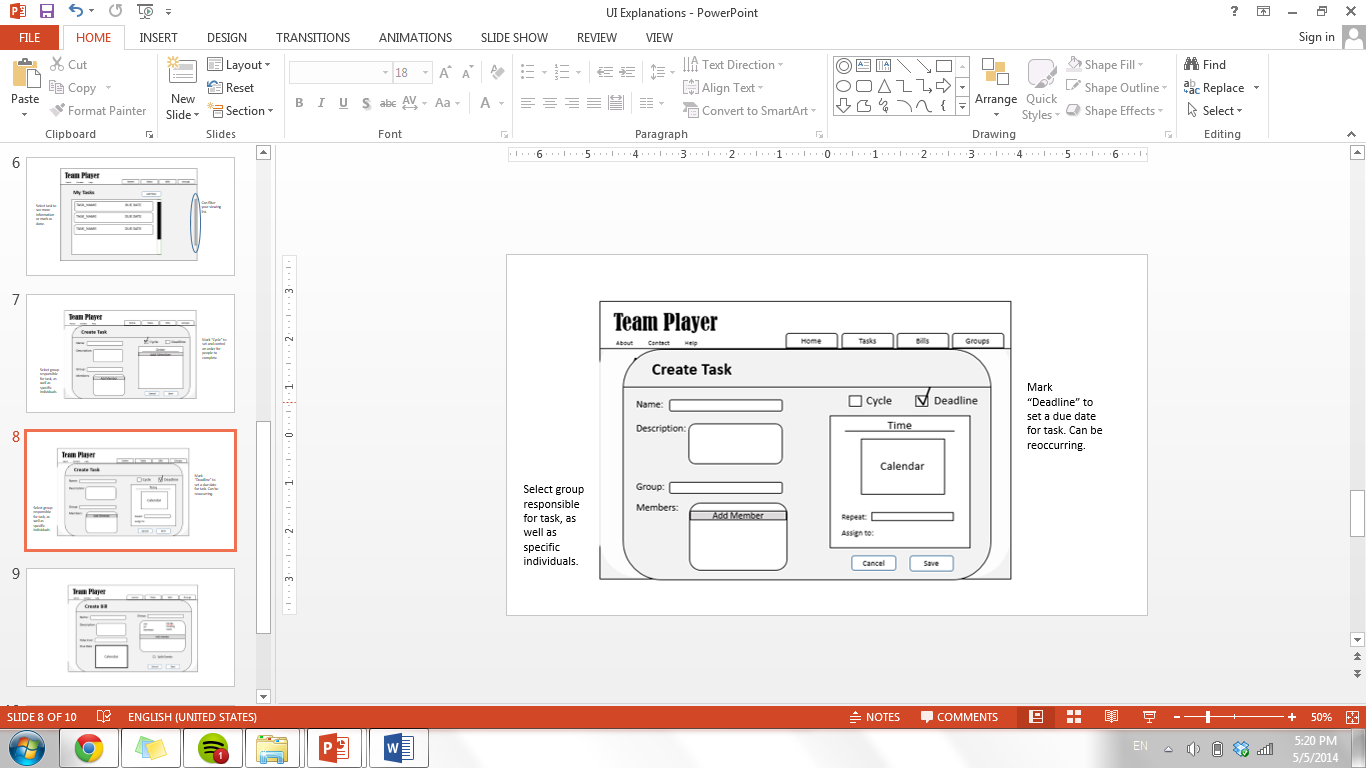


**Figure 5. All Tasks View**

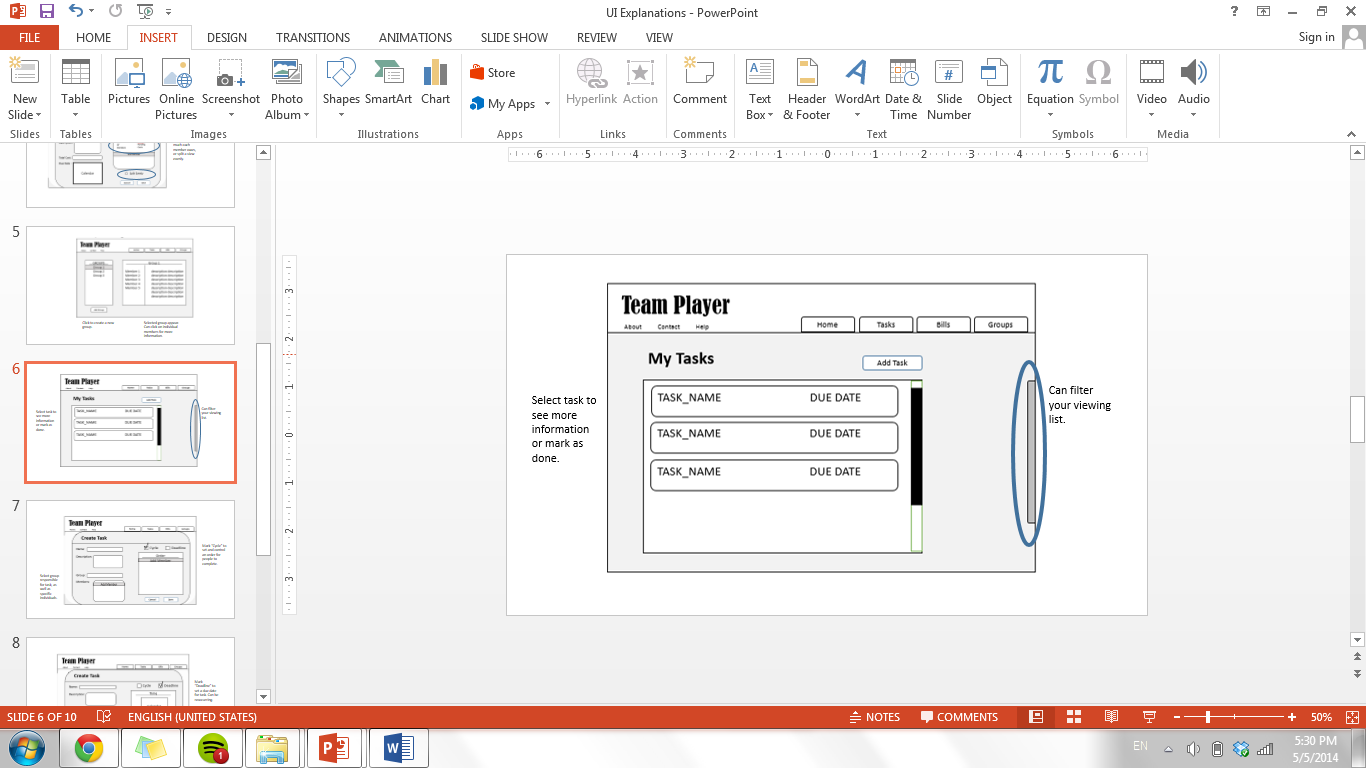


**Figure 6. Task Addition View**

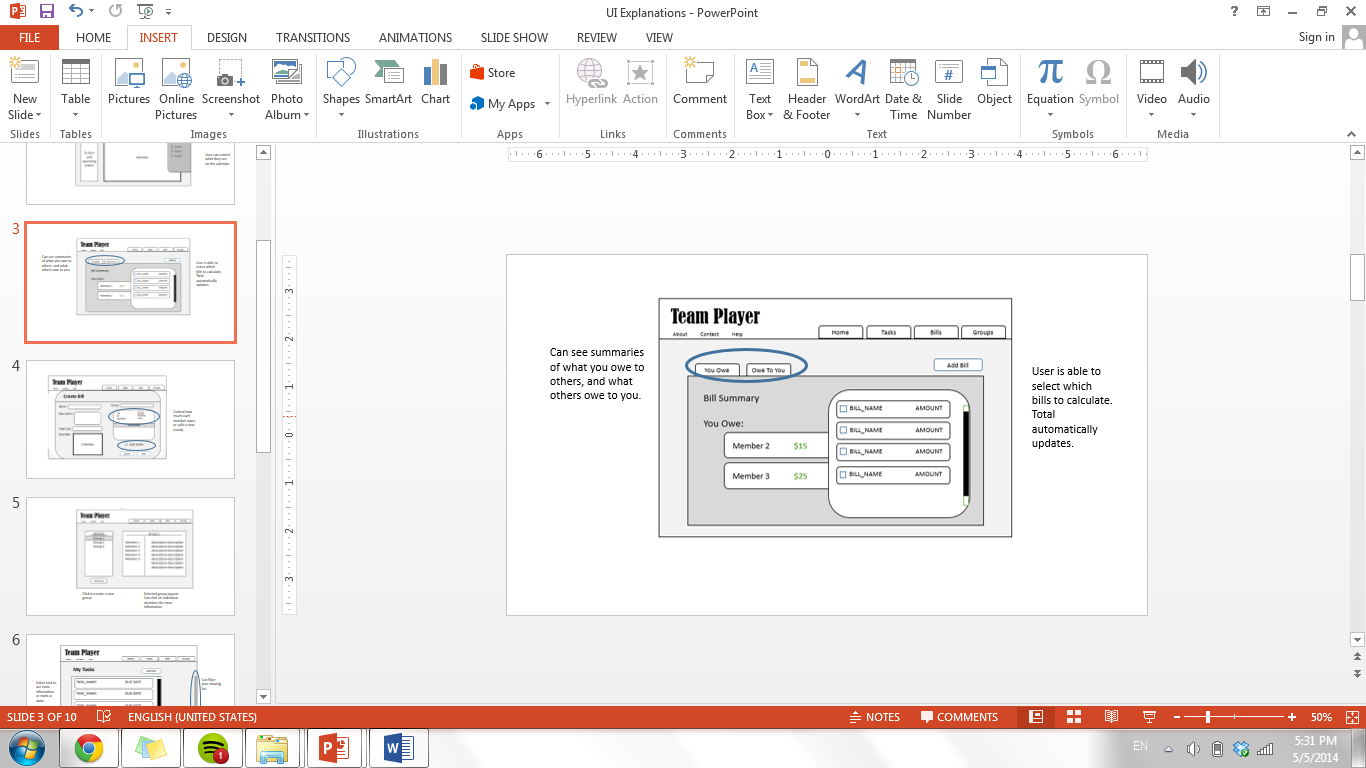




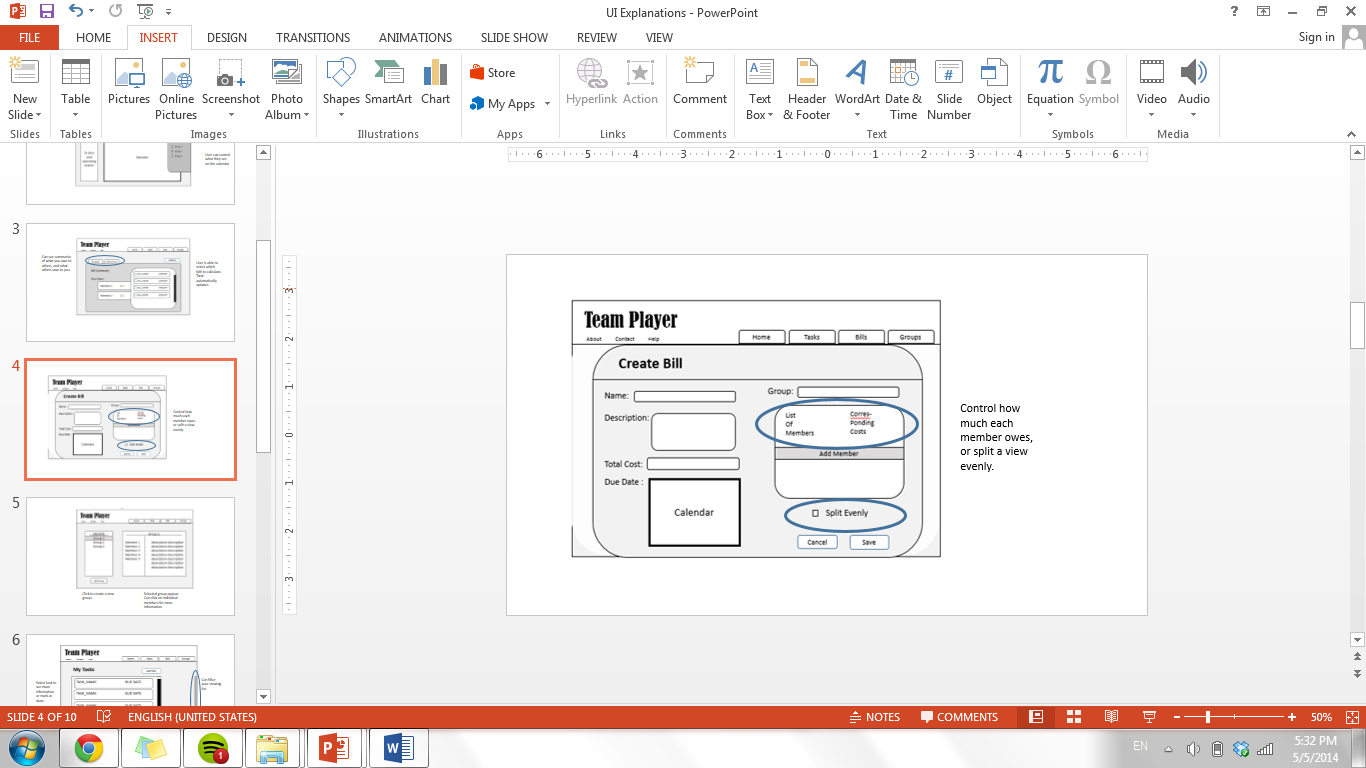
**Figure 7. Task Summary View**



**Figure 8. User Bills View**



**Figure 9. Bill Creation View**



**III. Use Cases**

**Use Cases Considered**

The following use cases are the most important ones and they cover the important scenarios.

* Login/Create Account

Every user must go through the Login/Create Account process to use this application.

* Home Page View

The home page is the first view after the user logs in. It contains summaries of user’s information and it will be displayed every time a user is logged in.

* Group Management

Since our app is a group based task and bill management application, joining and creating groups is an important use case as it is how our task and bills are organized.

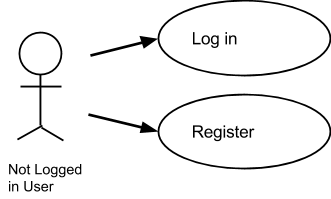
* Task Management and Bill Management

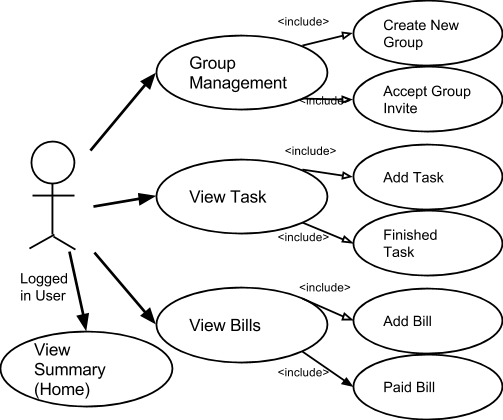
Task and bill management are the two core functionality of the application and will be frequently used.

**Failure handling**

The most frequent failure that would occur would be cause by concurrency issue: when two users of a group modified the same information. The general idea of solving this problem is to accept the first submitted change, and notify the other users that the information has been modified. If an unexpected failure happens, we would ask the user to provide feedback on what happened. We will then try to reproduce the failure, and start to troubleshoot to fix the problem.

**Use Cases Diagrams**

**

**

**Detailed Use Cases**

**Use Case: Login**

Actor: Not logged in User

Precondition: User is not logged in

Triggers: User, who have not logged in, come to the webpage

Success Condition: The user logged in and get redirected to home page

Failure End Condition: The user failed to login, back to login/register page

Use Case Basic Flow:

1) The user comes to the product website

2) The system displays the login input box and the create account input box

3) The user input his/her username and password in the login input box then click login

4) Login successful: the user becomes a “logged in user” and the system displays the application home page

Alternative Flow:

1. In step 3), if the user input a wrong username-password combination:

2. The system displays an error message and ask the user to try again

3. Back to step 2)

**Use Case: Create Account**

Actor: Not logged in User

Precondition: User is not logged in

Triggers: User, who have not logged in, come to the webpage

Success Condition: The user creates an account get redirected to home page

Failure End Condition: The user failed to create an account , back to login/register

Use Case Basic Flow:

1) The user comes to the product website

2) The system displays the login input box and the create account input box

3) The user input his/her name, email, and desired password in the create account input box then click “Create Account”

4) An account is successfully created: the user becomes a “logged in user” and the system displays the application home page

Alternative Flow:

1. In step 3), if the user input an email address that has been used before:

2. The system displays an error message and ask the user to try again

3. Back to step 2)

**Use Case: View Summary (Home)**

Actor: Logged in User

Precondition: User is logged in

Triggers: User come to the webpage or clicked on “Home” Tab

Success Condition: The user see the summary of tasks and bills in all his/her groups

Failure End Condition: Page fails to load

Use Case Basic Flow:

1) The user come to the webpage or clicked on “Home” Tab

2) The system displays a summary of user’s tasks list, a calendar that contains tasks and bills information on all the user’s group, and a notification bar displaying the most recent news

3) The user may continue to use case “Group Management”, “View Tasks” or “View Bills”

**General Use Case: View Group/Group Management**

Actor: Logged in User

Precondition: User is logged in

Triggers: User clicked on “Group Management”

General Use Case Basic Flow:

1) The user selected the “Group Management” Tab

2) The system displays the current groups that the user is in, and displays groups that the user being invited to

3) The user may choose to continue to a sub use case

**Sub Use Case: Create New Group**

Actor: Logged in User

Precondition: User is logged in

Triggers: User clicked on “Create New Group”

Success Condition: The user defined group is created

Failure End Condition: Nothing changed from the start

Use Case Basic Flow:

1) The user selected “Create New Group” on Group Management page

2) The system displays a popup window, prompt for user input on group name, description, and invite group members

3) The user inputs desired group name, associated description, invite members to the group, and clicked “Add”

4) The popup window disappears and the system displays the Group Management page, with the new group created displayed

5) The system stores the new group information

6) Step 1-5 may be repeated

Alternative Flow:

1. in step 3), if the desired group name is already a group name of one of the user’s groups:

2. The system displays “Group Name Exist” message when user clicks “Add”

3. The user clicks “OK” and changed a group name not used before

4. Continue to Step 4)

**Sub Use Case: Accept Group Invite**

Actor: Logged in User

Precondition: User is logged in

Triggers: User clicked on “Accept Invite”

Success Condition: The user join the group inviting him/her

Failure End Condition: Nothing changed from the start

Use Case Basic Flow:

1) The user checked the checkboxes next to the groups inviting him/her then clicked “Accept Invite”

2) The system displays a success message and display the Group Management page with the inviting groups moved to the “My groups” section

Alternative Flow:

1. in step 1), if the user did not check any group then clicked “Accept Invite”:

2. The system displays an error message and ask the user to try again

3. Back to step 1)

**General Use Case: View Task**

Actor: Logged in User

Precondition: User is logged in

Triggers: User clicked on “My Tasks”

General Use Case Basic Flow:

1) The user selected the “My Tasks” Tab

2) The system displays the list of tasks, with brief info of each, needs to be done by the user

3) The user may choose to continue to a sub use case

**Sub Use Case: Add Task**

Actor: Logged in User

Precondition: User is logged in

Triggers: User clicked on “Add Task”

Success Condition: The user defined task is created

Failure End Condition: Nothing changed

Use Case Basic Flow:

1) The user selected “Add Task” on “My Tasks” page

2) The system displays a popup window, prompt for user to select group that the task belongs to, name, date and description of task, and members responsible for the task

3) The user input the requested information

4) The system displays a preview of what it will look like on the calendar

5) The user select “Add”

6) The pop up window disappears and the system displays the “My Task” page

7) The system stores the new task information

8) Step 1 - 6 may be repeated

Alternative Flow:

1. in 5), if the user put in a task name already exist on that day:

2. The system displays an error message and ask the user to try again

3. Back to step 2

**Sub Use Case: Finish Task**

Actor: Logged in User

Precondition: User is logged in

Triggers: User clicked on a task name

Success Condition: The user defined task is marked as “Finished”

Failure End Condition: Nothing changed

Use Case Basic Flow:

1) The user selected one of the tasks from his “My Tasks” list

2) The system displays a popup window, displaying detailed information about the task including task name, description, participating members, and date.

3) The user clicks “Finished”

4) The pop up window disappears and the system displays the “My Task” page

5) The task will be marked as “finished”

6) Step 1-5 may be repeated

**General Use Case: View Bills**

Actor: Logged in User

Precondition: User is logged in

Triggers: User clicked on “My Bills”

General Use Case Basic Flow:

1) The user selected the “My Bills” Tab

2) The system displays the list of bills related to the user, with brief info of each and a bill summary

3) The user may choose to continue to a sub use case

**Sub Use Case: Add Bills**

Actor: Logged in User

Precondition: User is logged in

Triggers: User clicked on “Add Bills”

Success Condition: The user defined bill is created

Failure End Condition: Nothing changed

Use Case Basic Flow:

1) The user selected “Add Bill” on “My Bills” page

2) The system displays a popup window, prompt for user to select group that the bill belongs to, name, date, amount and description of bill, members involved and which member paid

3) The user input the requested information

4) The system displays a preview of what it will look like on the calendar

5) The user select “Add”

6) The pop up window disappears and the system displays the “My Bill” page

7) The system stores the new bill information

8) Step 1 - 6 may be repeated

**Sub Use Case: Paid Bill**

Actor: Logged in User

Precondition: User is logged in

Triggers: User clicked on a bill name

Success Condition: The user defined bill is marked as “Paid”

Failure End Condition: Nothing changed

Use Case Basic Flow:

1) The user selected one of the tasks from his “My Bills” list

2) The system displays a popup window, displaying detailed information about the bill including bill name, description, participating members, amount and date.

3) The user clicks “Paid”

4) The pop up window disappears and the system displays the “My Bills” page

5) The bill will be marked as “Paid”

6) Step 1-5 may be repeated

**IV. Process Description**

**Software Toolset**

We decided to make our project a web-based application, because our team is most familar with web languages, and web is cross-platform (i.e. we do not need to worry about iOS vs Android). We also acknowledged that building well-designed UIs, which will be a large factor over our competitors, is easiest with web languages. Here are the tools and the explanations on why they were chosen:

* *HTML/CSS/JavaScript*
  + The backbone of the frontend of any web application. We will be using these heavily to design our website.
* *JQuery*
  + This is a JavaScript library that will simplify our frontend code - t provides helpful functions on Ajax requests, basic animation, selecting elements in the DOM, etc.
* *Bootstrap*
  + This is a JavaScript and CSS library originally from Twitter. We will be using many of their default styles and boilerplate code to help aid our design of the website. For example, it is not terribly easy to implement a professional looking button or form in HTML/CSS, but Bootstrap’s provided CSS will help us make the site more professional.
* *AngularJS*
  + This is a MV\* framework for frontend heavy applications. We will use it as both a “frontend PHP” (e.g. HTML with control flow), and to easily inject the data from the backend.
* *Ruby on Rails*
  + We will use Ruby on Rails as our backend framework. The backend will provide the overall architecture of the site, as well as handle querying the database, and returning the formatted data to the front end.
* *sqlite3/PostgreSQL*
  + Our data will be very relational, so our ruby on rails backend will use these databases to store data, sqlite3 for development and PostgreSQL for production. Then, we will query it through the ruby functions. The data will solely come from user input, as the application is driven around group task management.
* *Git*
  + This will function both as our version control and bug tracking system. Git is a fairly intuitive and popular version control system. Its bug tracking system is visual and clear.
* *Heroku*
  + We will be using Heroku to deploy our app. It has Ruby on Rails already installed, so will simplify the deployment process.
* *Jenkins*
  + This is a continuous integration system used to deploy automated daily testing on our product, and send emails when builds have failed.

**“Off The Shelf” Software Components**

The main UI challenge will likely be implementing the calendar view. We first discussed Google Calendar, but unfortunately, that would allow users to modify their calendar without going through our system. That would immediately render our backend inconsistent with their view, which is a problem we do not want to have. We then came across a JavaScript calendar implementation (<http://arshaw.com/fullcalendar>) that allowed more control over user access. We probably will use this as the base of our calendar, but adding in the variety of features we discussed. We will now use an Angular UI Calendar (<http://angular-ui.github.io/ui-calendar/>) which uses FullCalendar and comes with open source files we can use.

Other than that, our frameworks (Angular and Rails) and libraries (JQuery), will help prevent us from needing to reinvent the wheel. After getting past the learning curve, these should dramatically reduce the amount of code we need to write. This will be especially important in synchronizing the backend and frontend. From scratch, we will likely need to design/code the frontend views (login, manage groups, calendar, todo’s, and bills). On the backend, we will need to design the database architecture, and how to query it.

**Group Dynamics**

We have decided on the following roles:

Project Manager: Karthik

Front End: Linsen, Fung, Tiffany

Back End: Micaela, Panji, Keith

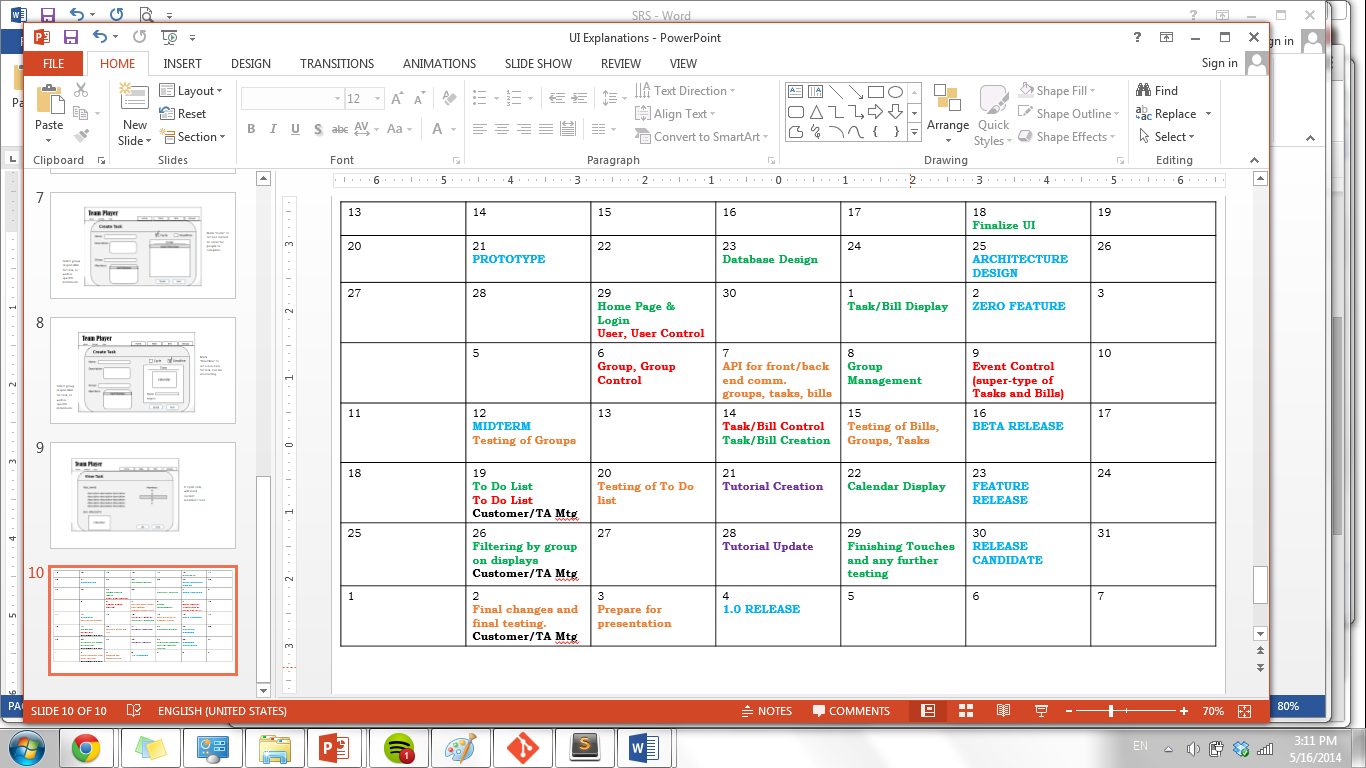
Our project manager was selected because Karthik was interested in understanding the integration of front end and back end, he also has some experience with other frameworks and APIs. Also, with his knowledge in a little bit of both sides, we felt that he would be able to help with future decisions. Other roles were chosen based on the previous experiences and knowledge each of our members have. Those in the back end have taken database classes and are familiar with more development languages. Those in the front end have taken web programming classes and have some experience with HTML, CSS, JS and web related skills.

Roles are flexible since members will be expected to support each other. Those who have knowledge in areas can help with coding. And those who end up having less to do for a week can help with researching, teaching, and learning new tools to help with project development.

Based on how our group has currently been cooperating, each of us are very flexible to new ideas and learning. At the beginning of this project, we had critical discussions over what project our group would work on. Each of us were not afraid to make claims for why we felt a project would be better than another. Our problem was deciding on what we would all move forward with, which we have learned from by voting. If a disagreement arises, we all have a chance to discuss our thoughts about the problem. Members are asked for their ideas and are free to input where they see a problem with a current situation, and can make suggestions. We will decide by consensus - we are each flexible and just want to do what’s best for our team. If we absolutely cannot agree on something, we will have a vote and majority wins.

**Schedule / Timeline**

Below is a rough schedule we have put together based on class deadlines and our core features: group management, login, bill/task creation, calendar display, to do list display, and bill display. In blue, are deadlines for our CSE 403 class, and in green are the TeamPlayer deadlines we hope to meet. Additionally, in purple are deadlines for tutorials – first to create, then to see whether our customers know how to use it, and in black are customer and TA meetings to make sure our app is usable.



We first estimated the amount of time it might take to finish these features, some of which could be grouped together. We realize that we need to organize the front and back end simultaneously, since it is essential that they will need to communicate with each other. We want to know immediately that both ends tie together nicely.

* Task & Bill Display/Login = 1 week

This merely stores user ID, email, and group names in a database. The system needs to handle group invitations, which can be as simple as sending out emails. This should take 2-3 days but the setup of a working database may require 3-4 days more.

* Group Management = 4 days

This ensures accurate viewing of what groups a user is a part of. There is a lot of back end authentication, as well as display and updating after events.

* Bill/Task Creation = 1 week

Creating bills and task are very similar, we need to create a simple UI for inserting data values into the database. Simple forms and validations are easy to implement.

* Calendar/To Do List Display = 1-2 weeks

Displaying calendar takes longer because of the complex ways we can put them on the web app with possible features like monthly, weekly, day views and color coding events and tasks. More importantly, we need to figure out how the data will be stored on database to allow scalability.

**Risk Summary**

We feel that major risks we will face while completing this project include integration between code, like connecting the front end UI and backend database, managing time to finish all requirements before deadlines, and overestimating our ability to implement certain features. The front end and the back end are being developed separately so we will have to make sure to communicate carefully so the two parts will be easy to combine. The front end javascript has to be able to take into account the backend code from the beginning so they can be joined.

We are most worried about time management because it is hard to accurately estimate the size of a project which can lead to not having enough time to complete all the features. This is a serious risk because we have a given amount and if we do not finish by then we can’t just extend our deadline. Features relates somewhat to timing in that we are not sure how long each feature will take, and so we need to make good decisions about which features are core and which ones are stretch features. Or else we could make the project too complicated and unfeasible and not finish on time.

To try and reduce risks, we could attempt to build a small sample of the project, like the home screen to see how long it takes. Then with that information we can have an estimate to how long we will need for the whole project. If we cannot overcome a problem, like we are running out of time, we will organize more times for the group to meet so we can meet the deadline. We can also move people around to work on different sections, so if one part is falling behind other people can work on that part instead. If nothing works and we have no time left in the end, we would probably have to end up cutting features.

The points in our process where external user feedback would be most helpful would be after we have our UI and features figured out, and after we have a useable website. At this point feedback would be helpful for finding bugs, flaws, or anything missing that people may find useful.