# SpareTime

As a whole, we were pretty pleased with our project. It turned out to be a pretty professional beta product and we believe it could have legitimate potential. We believe that we can attract many users to our website because of how easy it is to use and because of how helpful it could be. We see a lot of people using whenisgood and Doodle and those markets are definitely reasonable targets for SpareTime. Our value added through our heatmap, connectivity with people's Google Calendars, and Facebook will propel our growth if we execute scaling properly.

How did your milestones go?

Our group did a good job of outlining our goals and setting manageable milestones for us every week. For the most part, we were able to accomplish more than we set out to accomplish each week and this helped keep our confidence high as we were building the website. We also got a great head start from some of Mark's frontend work over spring break. Having the guidance of the TA's to force us to have some presentable material every week definitely pushed us to get more done every week. As a whole, we got started on the project pretty early, making many of our design and over-arching implementation decisions early in the process. This made it much easier to understand what some of our challenges would be and helped us make a more realistic set of goals from the outset for our project.

What was your experience with design, interfaces, testing, etc.?

We tried to stay on the simple side in terms of design and interface. From the outset, we decided upon a typical workflow for a user to our website and concluded that it would be beneficial to have as few seperate pages as possible. This led us to our decision to have a homepage for the un-registered user and a calendar page from which a user could perform all their functions on our application. Because SpareTime is at its core a calendar application, we had plenty of great examples of user interfaces (ie. Google Calendar and iCal). We used these to often guide our design choices and we felt that this was a successful method. At the same time, we wanted our user interface and design to be unique and identifiable with SpareTime. Brand creation is an important thing to think about from the very beginning, so we thought it was important to make sure that our calendar can be easily identified as being our application.

We had to make a quite a few conscious decisions as we were designing our application. We especially put a lot of thought into the heatmap. We had the choice of giving users a gradient of colors based on peoples' availabilities or of only providing three or four colors on the heat map. The benefit of only having a few colors is that it makes the heat map seem a lot clearer. However, we thought about the downsides of that. If many people are invited to an event (let's say 10 people), then knowing whether 3 people are busy at a time or 7 people are busy at a time is a large difference. However, with only a few colors, this difference would not be completely clear. So, we decided to have a smooth gradient in colors from green to red based on how many people are busy. Users could discern the difference when they hover over a time slot (a popover it tells them who is unavailable).

Another interesting decision to make was how to properly deal with overlapping events. We had to do use some small, but clever algorithms to decide how to handle overlapping events in such a way that they are all still visible.

One major design/implementation decision we had to make was whether or not to open-source the creation of the calendar. We decided to implement our whole calendar from scratch because we believe that the calendar is central to our application. So, we believed that we should develop this critical part completely on our own. The talk from Brad Smith confirmed this when he said that we shouldn't open-source those features that are central to our application. On the other hand, having friends and connections are not central to our application so we are comfortable importing them from Facebook (which is an established social network already). In the case of Facebook, we wanted to leverage the power of the social network that already exists there.

For the most part, our testing was limited to the four of us creating the app. During the last week and a half before dean’s date, we were really able to start getting more users to test our app. This helped reveal some bugs (especially with how we were able to handle importing Google Calendar events). We would have liked to get more users earlier. One thing we would have to do a lot more of going forward is stress testing based on increased number of users. Specifically, it would be important to test how our heat map features scales as one invites more people to an event. The average use case would most likely include just a few users; however, we should be able to handle many people invited to an event.

What are some surprises (pleasant or not), that we found? Feel free to tell me if what I'm saying is true.

* We also found jQuery to be a very useful tool, along with Ajax, to have functionality that did not require reloading the page.
* We were pleasantly surprised to find that Django did a great job of setting up our project in terms of the MVT architecture. This made organizing our code slightly easier.
* We were surprised to find out how complicated the Google Calendar API is. Particularly, Google does a lot of things when storing information about its events that can be a little difficult to keep track of. For example, repeat events are saved as RRULEs and exceptions to repeat events are stored as another variable. This required some significant work on the backend in order for us to successfully import all of the events that a user had on Google Calendar.
  + We also found that there were odd exceptions to the format of a Google Calendar event request. For example, events on ICE are formatted differently so we had to deal with them seperately. We, especially Sammy, had to spend a lot of time tailoring a lot of our Google Calendar connectivity within our application.
  + During the importing of events, we had to mold everything in our database so that we grabbed every single one of a user's events and not just a subset of them.
* Heroku turned out to be slower than we expected or would have wanted. While it was very easy to push our website (there were very few settings we had to change in order to make it work) to Heroku, we found that its performance wasn’t great using the free developer accounts. There was slight, but noticeable lag when we populated events for a user with a large number of events. This is something we want to avoid because the users who keep track of lots of events are the audience we want to target most for our application.

What choices did you make that worked out well or badly?

Working with Django was a great choice for us. The python scripting language is very natural to use and has a lot of built-in functionality, especially for parsing text and inputs that was useful for us on the backend. Django also had some great plug-ins, which were useful in simplifying our application. Django was especially easy to learn and get started with. We never had to deal with doing SQL queries or any specific database related code other than creating the models through Django.

Using jQuery was also a great idea. Rolled into this was our use of AJAX. Although we were all new to these tools, learning how to use them was valuable because it made life much easier than using straight Javascript. AJAX made the flow of our application much better because it allowed us to change the display on the page without having to refresh the page every time. Essentially, it allowed us to do everything on one main calendar page.

We made the decision to have dedicated event pages about halfway through the project. It was a very good idea because it made the use of the application to share events with others very intuitive.

What would you like to do if there were more time?

There are many features we'd like to implement going forward, in addition to some of the bugs we'd like to fix. One big feature we envision is allowing users to create and search for public events. This would work similarly to Facebook's public events where someone can organize an event they'd like to advertise for the public. Other users then can search for public events in their area that might interest them. This would allow organizations, such as those in Princeton, to advertise events to the public and have users, such as students, find out aobut them and choose to attend. We will continue to work on this during the summer and hope to launch and have users soon. We also hope to clean up our code and would have liked to do that if we had more time before presenting. We will most likely do that in the next few weeks during the summer so that it will be easier to organize our code and new features moving forward.

We also noticed that our website slows down slightly at different points. We'd like to speed that up. There are some ways we can approach this. First, we can just increase the number of dynos doing work on our heroku server. This would require a monthly payment. Alternatively, we could move to a better hosting service (ie. AWS or EC2). We would have to do some research to see if making such changes would actually result in a speed increase. Another thing we could do is try to look for performance improvements in our code. We could do this by somehow making fewer calls to the method that populates events on the frontend.

One of our big concerns is in terms of security. We have thought about the level of security we'd like for many of our features. For example, we have dedicated pages for each of our events. So, there is a direct link to each event that is added on anyone's SpareTime account. As a result, it is possible to currently view any event simply by typing in the url, which is easy to parse (each event has a different id). For a release version of SpareTime, we'd like to tighten access to these event pages. For example, a personal (a.k.a. private) event should stay personal and an event page should only be visible to the people invited to an event. On the other hand, we could have public events that are visible by anyone and which anyone could add them to.

We'd like to have group functionality. That is, rather than having to add events to friends one-by-one, you could create dedicated groups of people whom you could include in an event all at once by simply typing the name of the group. This would vastly reduce the time required to invite the same groups of people to multiple events.

We also would like to implement picture-sharing functionality. We want to be able to add pictures to events, both before and after they happen. This increases the social value of the calendar because people can share pictures from the event directly on our application.   
We would have liked to implement an improved notification system. That is, people could opt-in or have to opt-out of being notified when they are invited to an event by email or on Facebook. This allows people who don't necessarily regularly use our app to know that they have been invited to an event or if there are any changes to the event.

Finally, we'd like to continue improving the UI/UX experience. We believe that it is of the utmost importance that our application is intuitive and easy-to-use. We want to improve the flow for the user as they go through the various features of our application and we want to make it obvious what is happening at each step so that they can maximize the use they derive from it.

How would you do things differently next time?

There is not a whole lot we would do differently next time. One thing we would have benefitted from is having better communication about the division of tasks and in the functioning of the system as a whole. It can be difficult to keep track of all of the moving parts of a (potentially new and unfamiliar) large system. This includes having a better division of labor.

We would also have tried to test the system with more users earlier. We tried to get people to use it, but mostly Sammy did the best job of recruiting people to use the application and try to find bugs. Sitting down with users and watching them play with the application was very helpful late in the process and we would have benefitted from doing more of this earlier.

What did we learn?

Often of the things we learned from this project was to be ready for unexpected changes. For example, Facebook changed the way it handled accepting requests from localhost and we saw this two days before our demo. This concerned us a little bit because we had considered using localhost to demo the application, but fortunately we were able to deal with the change. Something that was completely out of our control was when the Google Calendar API was down for about an hour one night that we were hard at work. We were very confused for a while because we thought something was wrong on our end. Even though it wasn’t our fault, it caused a lot of stress and could have been problematic if we had many users in the real world.

Many of us learned most, if not all, of the stack involved with web development, both on the frontend and the backend. These are very valuable skills for a future in software development. On a smaller scale project, we were able to see and write code on both ends, which gave us a better understanding of how the frontend and backend interact to form a website.

It was a valuable experience learning how to handle and deal with Heroku. Making a website work on localhost is one thing, but pushing it into the real world comes with its own learning experiences and challenges. Understanding a bit of the technology behind web hosting will be something that helps us moving forward, especially in knowing what it takes to take a website to scale.

Learning how to use Git in a team context was also very helpful. We will most likely use Git or some other version control for any project in the future, so it’s good to have a solid understanding of it from this project. Git really helped us keep track of our changes and allowed us to experiment with code and see each other’s experimentation without changing the master code base. It also helped us roll back any mistakes we made with our code and made sure we couldn’t accidentally lose all of our work. We did run into some troubles, and never fully figured out how to resolve merge conflicts, but all-in-all we used Git extensively and benefitted from using it.

What should next year's class learn from your experience?

Next year’s class should be aware of how quickly the end of semester comes. After spring break, there is only a little over a month and a half left to complete the website before deans date. Thus, it is very helpful to get a lot done over spring break. It was very helpful that we, especially Mark, was able to get a lot done on the frontend over spring break and we were all able to learn Django during the break so we could go straight into development in the weeks afterwards. It is also very helpful to just go with what somebody in the group already knows how to do. Pal knew Django and Heroku from past experience, so it was helpful that he could easily push all changes to Heroku and figure out how to debug problems. It can be frustrating for everyone to be trying to learn and figure things out like that, so any prior knowledge in that area can save lots of time.