"Get-A-Room" Project Proposal Spec

Peter Dolan, Carlos Girod, Alec Powell, and James Webb
{pdolan,cgirod3,atpowell,jmwebb}@stanford.edu

January 12, 2016

Abstract

Room reservation systems exist for many workplaces on Stanford's campus, but they are extremely difficult to find and navigate for anyone but the most dedicated. Our group envisions a service that creates a simple, streamlined approach for booking rooms using either a modern messaging platform or a more traditional web application. Our goal is to make booking a room as easy as possible for the user.

1 Description of the Project

The difficulty of finding a group room to work in is a ubiquitous Stanford problem, so common that our group encountered it while writing this very paper. Simply put, there is no simple and effective way to get a group study room without trekking all the way out to it. In the all-too-frequent situation that all the rooms in a single building are booked, the walk is wasted which results in frustrated students with no place to go.

Some Stanford locations are more forward thinking about this problem: Old Union has a simplistic but effective reservation system online in place to combat this problem However, altogether too many buildings do not. Luckily, Stanford hosts a central source of data via a web application called 25live. With room locations and a detailed calendar for each of them, 25live might seem like a the perfect solution for students. Unfortunately it suffers from several serious issues which are outlined later in this paper. Our group wants to make this data easily accessible and equally malleable by pulling it and creating a modern database and interface. We visualize either a traditional website or a trendier messaging style platform.

From the user's perspective, our platform would be simple. They (the user) navigate to either our web platform or messaging service, and request a room. They do this either via a message (something like I want a room near Huang in an hour) or via a series of menus. From there, our backend determines the viability of the requests and returns either a message indicating no rooms, or a positive with the option to book. Should they book, our data is updated with this new information.

The fundamental goal of our system is to make a fast, simple room booker. In an ideal world, the UI is as simple and streamlined with as little hassle as p. This should make room-booking significantly easier.

2 Need for the Product

In general, there is confusion about room reservation at Stanford - there is no centralized online place to go to find out if a room is reserved, schedules are not always posted outside of rooms, and rooms may be occupied for the wrong reasons. We have noticed (and personally experienced) this problem area over our time here and see this as a problem that can be solved by a cohesive, all-encompassing and easy-to-use application.

After a thorough investigation of other products currently used by Stanford affiliates to view room reservation schedules and book rooms, we came to the conclusion that these products can be incomplete (only for specific buildings such as Old Union), confusing to deal with, not fast (too many pages to load), and not user-friendly. We will discuss these products in detail under Section 5.

Some use cases that demonstrate the need for this product include: A group of students or class that wishes to reserve a room ahead of time for a meeting Individuals wanting to find a room to use presently for studying, a quick meeting, or phone call A professor/TA wishes to schedule office hours for the upcoming week, or find a room to hold a review session for their class before an exam

Our product will solve this problem by providing a hub for any person with an SUNetID to find an open room to either use immediately or reserve for a future time. The product will display all on-campus buildings and rooms within them, indicating whether each is currently in use and the future reservation schedule for each room.

We feel that a streamlined system for room reservations will make the student body more productive, the signup process more efficient, everyone better informed about what rooms are in use or not, and the optimization of all rooms available for use on the Stanford campus.

3 Potential Audience

The target demographic for our product is Stanford affiliates. We feel that any student, professor, or employee at Stanford has encountered the problem of room reservation, whether for personal needs, group projects, or class/office hours. Everyone at Stanford is entitled to the best workspace possible to make their learning easier; we want to support this culture of innovation by making room reservations less confusing and less of a hassle to everyone.

In order to accurately determine the needs of our audience, we plan to craft a survey of the student population to find the specific problems with room reservation and identify more user stories to add to our plan. After creating the minimum viable product, we will again turn to fellow students for testing and bug-catching to make the application as helpful as possible. Our goal is to create a platform that is a one-stop-shop for room reservations, and one that users will find it very easy to interact with.

4 Discussion of Competing Products

4.1 25Live from CollegeNET

Stanford University currently uses a third-party web application called 25Live developed by CollegeNET, a software developer that makes products for higher education institutions. 25Live allows university (in this case Stanford) affiliates to search for and reserve rooms on Stanfords campus. An average workflow using 25Live would involve using a search bar to check the reservation schedule for a particular room, building, or cluster of buildings. Upon seeing the reservation schedule for a particular room or room group, the user could then reserve a room if it is not currently reserved.

There are several major issues, mostly with the 25Lives user interface, that we believe our product would substantially improve on:

- **Problem:** The system is painfully slow when populating the reservation schedule for the room or room group. For every room or room group query, the system will populate the reservation schedule calendar with the reservations for all seven days of the current week. For more popular rooms, such as lecture halls, loading the entire week of reservations seems to seriously slow down the system, sometimes taking several minutes for the calendar to populate. This problem is compounded when a user queries a group of rooms instead of a single room.

Solution: Our application will employ more modern database architectures to improve speed with fewer page loads. We also plan to prune query results to include only rooms and room groups that would be especially relevant to the user. This will hopefully save on the amount of data that must be fetched from the database and improve performance.

- **Problem:** The UI is clunky with an esoteric navigation system. Some links on the applications home page seem to be shortcuts to viewing the reservation schedules for various room groups; however, the language of the links make it unclear to where exactly they direct the user.

  Solution: Although we have yet to finalize a direction for our UI (web application or messenger bot), modern application design backed by JavaScript would allow for much needed interface improvement. We plan to simplify navigation by reducing page loads and the number of inputs the user must provide.

- **Problem:** Much of the decision making is left up to the user knowing exactly what room he/she wants to reserve. There are no maps to indicate where a particular room is on campus, nor any indications of the size and style of the room (e.g. group workspace, silent study room, etc.). This can lead to confusion and dissatisfaction among users, and also limits discoverability.

  Solution: These issues can be resolved with a few small additions to our applications UI, including a real-time map of room availability and query fields that allow the user to narrow a search room type.

25Live does have several benefits we plan to adopt into our product:

- Most importantly, 25Live presents a near comprehensive single source of information. As the only reservation system for most of the buildings on Stanfords campus, many users must schedule their meetings through its interface, meaning its data is often the most up-to-date.

- Reservations made through 25Live are given legitimacy by Stanford University, meaning users making reservations through the application have guaranteed use of the reserved room(s) at the reserved time(s).

4.2 Old Union Reservation System

Although most buildings on campus support room reservations via 25Live, the buildings in the Old Union cluster (The Nitery, Old Union, and The Clubhouse) use an in-house reservation system. This system accomplishes fundamentally the same task as 25Live with some minor differences. Via a web application, users can select a room group which directs them to a reservation calendar. From this calendar view of the current week, users can see all current reservations for the selected room group. If the user wishes to make a reservation, he/she can then select a slot on the calendar that is unreserved, and then is prompted to complete a reservation form.

Old Unions reservation system also has several issues (mostly with its UI) that we plan to address with our product:

- **Problem:** The system requires the user to click through and load multiple poorly constructed pages in order to complete a reservation. This needlessly prolongs and convolutes the process of checking current reservations and creating new ones.

Solution: With both of our potential approaches to UI (web application or messenger bot) we plan to resolve the issue of"too many screens" In our web application solution, we plan to implement an in-page JavaScript application that will consolidate user interactions to a single, clean page in the web browser. This will improve speed and reduce the number of clicks required of the user. In our messenger bot solution, the bot will ask as few questions as necessary to successfully ascertain a users request and book a reservation. Current trends show that this form of interaction is much less taxing on the user than navigating multiple web pages.

Old Unions reservation system does have some positive attributes that we plan to incorporate into our product:

- The Old Union reservation system addresses some of the issues of 25Live, namely the ability to check the size and style of a particular room. This allows users to make an informed choice on which room will best suit their needs. It also reduces unused space that can occur when a single user unknowingly reserves a room intended for many people.
- Although the entirety of the reservation process can take several minutes due to excessive pages, Old Unions system does load the reservation schedule calendar for a given room group in a matter of seconds. This is a marked improvement on 25Lives sluggish runtime.

5 Major Technologies Used

Our system is going to have several layers of functionality. The first, which will entail interaction with the user, will be one of two forms. The website frontend will be written using traditional HTML, CSS and JavaScript. The messaging side of the application is going to be handled via a google voice account. Google voice has a Python API which can receive and send texts. Our backend would be written in Python (with some sort of logic on how/when to reply to messages). All of the data we would store would be stored in a sql (or sqlite) database. Were planning on translating all the input we receive into SQL queries which will pull the data. This will then be parsed, and sent back to our front end.

The data for our SQL db will will be ingested in two ways. The first is from the simple messages from users. If they reserve the room, we will add an entry in the db to reflect this. The second is via an hourly or daily html data dump from 25live. This will be achieved via a Python library called urllib2, which pulls html from specific urls. If our group is able to get access to a purer data source (ideally another sql database) we will pull directly from that, saving us the costly step of navigating through a complex and prohibitively slow web application.

6 Resource Requirements

This project shouldnt require any unusual resources. We will be creating a full stack web application, so the libraries and technologies that weve described above should manage to fulfill all of our functionality. The only resource we may have some trouble utilizing is the database driving the 25Live website. We believe we can get in contact with someone who can tell us from where the 25Live data is derived, and once that is done, we can attempt to perform our hourly or daily data dump into our database space. In addition, even if we cannot through various means find the original data source for the 25Live website, we can use the means described above of grabbing the raw html and using nlp to parse the information and grab what we need.

7 Potential Approaches

7.1 Data Approaches

7.1.1 Data Collection from Get-A-Room Users

To get our product up and running, we will need to allow for our users to easily provide information to our reservation database. This means enabling the users through some UI (see UI Approaches) to be able to declare they are currently using a space, they are no longer using a space, and they are reserving a space for a future time. This data source will only reflect the actions of Get-A-Room users.

7.1.2 Incorporation of 25Live and Other Data Source

Because in our product will have low adoption in its early stages, we plan to supplement the reservation information submitted by users with information collected from existing data sources (see Figure 1 in Appendix). The two main sources of current Stanford reservation data are 25Live and the Old Union reservation system. By importing this data, we can give Get-A-Room users more accurate reservation information. Hopefully with time, the need to incorporate outside data will lessen as more users consolidate on Get-A-Room.

7.1.3 WiFi Connections

To allow for precise "real-time" estimates of room occupation, we are also considering to collect data from WiFi routers in various Stanford buildings. If Stanfords IT department will grant us access to router information, we could hopefully count the devices connected to a given router to get a sense of how many people are occupying a given room. This could be used to supplement our reservation database.

7.2 UI Approaches

7.2.1 Web/Mobile Application

One approach to UI would involve creating a dynamic web app through which users could claim a room, make a reservation, and check the status (free/occupied) of rooms on campus. This approach has several benefits:

- JavaScript web apps and iOS/Android apps are widely used among college students. The interface would be familiar to the user and the technologies involved are well-documented.
- Visual solutions to user stories would be relatively simple to implement. For instance, online calendars and interactive maps (both potential features of our product) have many existing examples in current web and mobile applications. Conveying this information via text message would be more challenging.

7.2.2 Messenger Bot

Messenger applications are becoming increasingly popular as a replacement for traditional mobile/web application interfaces. Messenger moguls WeChat and Facebook Messenger are already starting to capitalize on this trend. Using a messenger bot interface has several benefits:

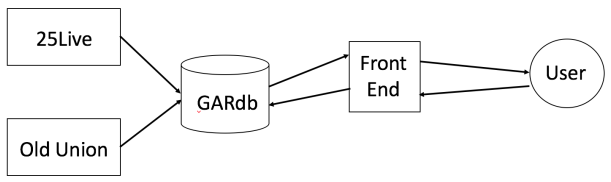

Figure 1: Application diagram. GARdb is the proprietary database that will power our application.

- No graphical UI would need to be implemented, as all processes between users and our system would be conducted through text messages. This would, however, require substantial NLP work to assure user requests were properly handled.
- Natural language is a universally understood way of making requests. Assuming we could implement an accurate NLP system, user adoption of our product would require very little learning curve.

8 Assessment of Risks

In considering any possible risks for this app, we thought about the ways in which the app can be purposefully or accidentally misused such that the function of the app is undermined. To this effect, we thought through some of the use cases for this app, one being that users may use the app to find an open room, yet when they begin to use the room they may forget to check-in, showing other users that the room is occupied. This can be countered potentially by sending a push-notification to the user (e.g. 10 or so minutes after the user said they needed a room) asking, Have you found a room? If so, check in! Another use case that may occur due to malicious use of the app would be users checking in to multiple rooms or rooms that the user will not be studying in. A way to combat this is to, potentially on an hourly or bi-hourly basis, monitor unusual reservation behavior by a single user. Other risks may be users going to a room that is not available, uninterested in the fact that another user is occupying the room.

9 Next Steps

Our next steps involve organizing and preparing the various technologies to make a minimum viable product (MVP) for the web app. One major step will include investigating the 25Live data source and whether or not we can gain access to the original source. Otherwise, we will use the HTML scraping method as described above. Additionally, we will need to partition the different parts of this full stack web app among the different members of the team. The database, whether it comes from the 25Live source or from our own source, will need an efficient relational design such that we can insert and search information as quickly as possible. Before starting on the backend, we need to consider the different use cases and functionality that we want to allow the user access to. Once we have a model of user interaction, we can begin to understand the different classes and methods we must write in order to encompass all functions managing communication between the user and the database. Finally, we will have to do a lot of work towards designing the UI. Then, we can begin writing the HTML and CSS, potentially using bootstrap methods.

In order to determine a solid direction towards a product, we need to determine what the full functionality of the minimum viable product will be. We will aim towards creating a versatile backend framework such that adding functionality after creating the MVP will not be too daunting. When the MVP is created, we will begin the user testing process to get feedback on the application prototype. We will then be able to iterate on our design and implementation, to make sure our product is effective and helpful to potential users.