I. Introduction

The purpose of this project was to create an application for a local non-profit tutoring organization called New Haven Reads. The application will improve the way that the staff interacts with the schedules of tutor-student appointments. It will allow a more efficient work-flow at New Haven Reads, freeing staff members to work on tasks other than those related to the schedules. It addresses scheduling on two fronts: first, it provides an intuitive interface for using the schedule to check in students and tutors, and make last-minute changes to the day's matchings; second, it allows for efficient editing and maintenance of the schedule.

II. Background: New Haven Reads

The New Haven Reads Free Community Book Bank was founded by Christine Alexander in 2001 as a place to collect books and distribute them, free of charge, to the community. New Haven Reads offers free books to anyone who wants them, and currently gives out about 100,000 each year to individuals, teachers, and other non-profit organizations.

A few years after its start, New Haven Reads began offering free one-on-one tutoring to local children. The tutoring program has since grown from fewer than 10 children tutored each week to more than 500. This tutoring is made possible through the work of over 350 volunteer tutors. Tutoring takes place at four distinct locations.

III. Scheduling at New Haven Reads Now

Schedules of tutors-student appointments are maintained by the scheduling coordinator at New Haven Reads. These schedules are stored as Google Doc spreadsheets shared with between the staff and printed weekly. Thus, any changes made mid-week are usually written in by hand. These hand-written changes often get confusing. During the week, the schedules are used by coordinators, tutors, and students to identify which tutors work with which students. It is also used to check-in tutors and students when they arrive, to indicate when students and tutors will not be coming (or did not come), and as a convenient place to leave notes about permanent changes to the schedule. Furthermore, to cope with students and tutors who do not come, lots of handwritten changes are often made, as tutors are moved from a child who did not come to one whose tutor is absent.

At the end of the week, the printed schedules (with their written-in notes) are used by the scheduling coordinators to update the schedules. Students and tutors who have consistently not come for their appointments are called and possibly removed from the schedule, and a staff member or volunteer counts various statistics. The paper schedules are then filed into a binder.

IV. Goals of the Application

The primary goal of this application is to move this spreadsheet-and-paper work-flow into a web browser. The data will be moved to a database. Staff members will be able to access and edit the schedules from anywhere with an Internet connection. Coordinators will be able to make short-term
changes to the day's or week's schedule, as well as check-in students and tutors as they arrive. And all data relevant to the schedules will be automatically stored and easily accessible in a useful format.

V. Daily Check-In Interface

The primary component of the project is the daily check-in interface. This interface is designed to replace the paper schedule used by coordinators for checking in tutors and students, matching them to each other, and recording data. It has three components: a datepicker, tabs, and the HTML tables displaying the relevant data.

The HTML tables display the tutor-student appointments for the given day, separated into tables by time of appointment. The tables display, for a given pair, the student's name and the scheduled tutor's name. There are 'Y' and 'N' “buttons” associated with each tutor and each student. Users can click these letters to indicate that the corresponding person has arrived, or that he or she will not be coming (or did not come). When a 'Y' is clicked, the person's cell is colored green, and the change is recorded in the database. Similarly, when an 'N' is clicked, the person's cell is colored red, and the the change is recorded. After a 'Y' or 'N' is clicked, it is replaced with a 'Z'. If a 'Z' is clicked, the corresponding person is returned to a neutral “not here, but also not not here” state. The cell is uncolored and the change is recorded in the database.

There is also a column for the “actual tutor”. Since students sometimes work with tutors other than their scheduled tutors, and since the tool for making this change and recording it has historically been the schedule, this interface provides functionality for moving tutors to other students. Users can click on a tutor's name, and then click on an actual tutor cell. The actual tutor cell (whether it was previously blank or previously contained another tutor's name) will now contain the name of the chosen tutor. In the database, this tutor will be recorded as the actual tutor for the session. All of the interactivity of these tables is made possible through jQuery-heavy Javascript in schedule.js.

The tabs, provided by jQuery UI, separate the data into days. The tabs are labeled with the days of the week, and clicking on one of them brings up that day's data for the current week, complete with any presence, absence, or matching information that has already been associated with it. Specifically, and AJAX call is made by jQuery to the web server, which returns the HTML that is displayed. Elements of the HTML contain classes and data attributes that allow schedule.js and the relevant CSS stylesheet to display the information correctly.

The datepicker, also provided by jQuery UI, allows users to navigate to dates other than those within the current week. The datepicker actually functions as a week-picker, since it determines to which dates (that is, which week) the days on the tabs refer. There is work to be done to make the datepicker more directly select a week, but it is perfectly functional as it stands. A week is represented by the Monday of that week, and if a day other than a Monday is selected, code in schedule.js causes the most recent Monday to be selected instead.

VI. Editing the Schedule

The other component of this project is the schedule editing interface. This interface is powered by the jQuery plugin SlickGrid. It consists of tabs (again, one for each day of the week) and SlickGrid spreadsheets. When a tab is selected, an AJAX call is made by jQuery to the web server. The server returns a list of appointments as JSON, and this data is used to construct the SlickGrid spreadsheet. The spreadsheet can be navigated by mouse of keyboard, and consists of columns for the time, student
name, and tutor name. Double clicking a cell or pressing <enter> while the cell is selected allows it to be edited, though editing is disabled on the time column. When a tutor cell is edited and the name is changed to another tutor's name, that change is recorded in the database. Similarly for a student cell.

This minimum functionality is necessary for this interface to be usable, but again, there is work to be done. Most importantly, there is no means by which a user can add new students and tutors to the system. This feature is at the top of the list of features to be implemented before the system goes into use. Second, there is no error handling for names that are entered which are not valid student/tutor names. These two problems are interrelated and will likely be solved together in the next version of the system. Another feature that will likely appear in future versions is suggested completions. The idea is that when a user begins typing a name into the cell, an AJAX call will go off to the database, asking for possibilities that match what has been typed so far, and displaying those possibilities to the user.

VII. Technical Components

The project uses the Django web framework. I chose Django because it allowed for a quick, easy startup, letting me bypass much of the minutia necessary for getting a website working, which is not directly related to the project. Another benefit is that since it is so popular, it is fairly likely that New Haven Reads will be able to find someone with Django experience, or at least someone interested in learning Django, to work on the project after I have left, if they need to make changes to it. Django ties data in the database to data model classes in python, allowing for easy data manipulation. Django also provides an authentication system, which is important for protecting sensitive student information from the public.

The front-end work is done by Javascript that uses jQuery whenever possible. jQuery is a reliable, predictable, useful Javascript library. It is primarily used for interacting with DOM objects (jQuery's most notable feature is its used of selectors like those found in CSS stylesheets), acting on events, and making AJAX calls to the web server.

Web hosting is provided by dotCloud. dotCloud allows users to choose from tens of different services that are offered. Among these is the python hosting needed to run Django. dotCloud offers up to two such services for free.

The database being used is a PostgreSQL database. This database is also hosted by dotCloud, and PostgreSQL is the second service being used. I began the project intending to use an SQLite database, which was particularly simple because SQLite is built into python, and interacts quite well with Django. Further, SQLite has been shown to perform as well as more complex databases like PostgreSQL and MySQL in most circumstances. However, since the SQLite database was a local file that was pushed to the dotCloud server along with the code, everytime changes were made to the code, the old database file (that didn't reflect the changes made while the site was live) overwrote the server version. The recommended solution from dotCloud was to use PostgreSQL instead, and let dotCloud host it. Now, since I do not push a database along with code changes, the data is not destroyed with each push.

VIII. Other Design Details: Appointments versus Sessions

One design problem was determining how to represent appointments, which are scheduled meetings that occur every week, but which also have data tied to those weekly instantiations. We want a way to talk about the recurring appointment that happens on Saturdays, but also talk about whether the student
showed up for its instantiation on Saturday, December 17th. The solution used in this system is to draw a distinction between appointments and sessions. Appointments are the scheduled pairings tied to a particular day of the week at a particular time, but not to any particular date or week. There is an appointments table in the database that tracks the student, tutor, day, and time associated with each appointment. Sessions are instantiations of appointments. Associated with a session is the appointment of which it is an instantiation, data describing whether the student and tutor were present, the tutor who actually did the tutoring at the session, if tutoring took place (this will usually be the tutor associated with the appointment, but not always), and the date the session took place. Future versions may associate sessions with weeks instead of dates, since each session takes place once per week.

With this design in place came the question of how to handle the creation of sessions. Creating appointments is easy in theory – an appointment is created when a user chooses to create it. But a user should not be responsible for creating an instance of that appointment – a session – for every single week.

The functionality for adding and removing appointments, as mentioned earlier, does not yet exist. But it will be designed with the following invariant in mind: if today's date is X, and the session in the database with the greatest (most future) date has date Y, Y >= X, then for all appointments (if there is a notion of active versus inactive, then for all active appointments), there is a session associated with that appointment for every valid date (date that corresponds to the appointment's day) Z, X <= Z <= Y. In other words, if there's a session associated with some date in the future, there are sessions associated with all appointments and all valid dates between today and the date of that future session.

With that assumption in place, we do the following. When launching the schedule check-in view, we first check whether a session exists for the date 26 weeks from today. If it does, we assume, given the above, that all sessions for the next 26 weeks exist, and we continue as normal. If it does not, we produce an “updating database” screen, and send an AJAX call to create the sessions corresponding to the next 52 weeks. If this operation times succeeds or times out, the page is refreshed, which causes the check to happen again, and the process to continue if necessary. With this procedure in place, the New Haven Reads staff will have to wait a few minutes twice a year in the worst case.

To maintain this invariant when the ability to add new appointments is implemented, we must programmatically create sessions for the new appointments for all valid dates through the week of the most-future sessions that currently exist. Similarly, if we introduce the notion of an inactive appointment, we must delete all sessions between the date it is rendered inactive and the date of the most-future sessions. If it is reactivated, we must create sessions for all valid dates from the date of reactivation until the date of the most-future sessions.

IX. Future Work

There is a lot of work that can be done to improve this system. Below I list the ideas that I have divided into three categories: the work that must be done to make this system realistically usable; features/improvements that can be added relatively easily to the system; and features/improvements that will likely require a significant amount of additional work to implement.

The Necessary
- Make it possible to add tutors from the UI
- Make it possible to add students from the UI
- Make it possible to change student/tutor info from the UI
- Make it possible to add appointments
- Make it possible to make appointments inactive
- Handle non-existent tutor/student names in the schedule editing interface

The Reasonable
- Add reporting functionality (i.e., the ability to make (somewhat) arbitrary queries form the UI)
- Make the actual tutor column not change size
- Align the tables in the check-in interface
- Replace the 'N' column with 'call-out', 'no call / no show', and 'left early' columns
- Replace 'Y', 'N', etc with icons
- Add a notes section to the check-in interface
- Add a notes section to the editing interface

The Far From Here
- Integrate email/text message alerts to be sent out when tutors/students call out
- Integrate with volunteer information database