StaffStuff: A Custom Web-Based Software Solution for Organizing the Unique Staffing Needs of an Online Education Production Department

Christine E. Barnhart

A Thesis in the Field of Information Technology

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

May 2016

Abstract

StaffStuff is a custom software solution for management of video production staff and locations, including a web based user interface and a database of resources and the production projects to which they must be allocated. The package is tailored to the unique challenges of scheduling the filming of campus classes and other educational projects, within the context and restrictions of the clients' particular institution and department, and with consideration of their unique business logic. A successful beta was released on January 2, and is in current use for Spring semester and for Summer School planning.

The system ingests data about the courses and other projects for which the department produces media, either in bulk by file upload or by individual data entry. Each project may have a number of bookings for each instance of a video shoot or other required work session, including precise details of what resources will be required. For efficiency, it allows a regular schedule of weekly shoots to be defined if applicable, and for exceptions to that schedule to be entered. Resources may then be allocated to these bookings, such as particular locations and personnel. The system is able to run complex analyses to assist the user by identifying resources which are available for use. It reduces human error further by alerting administrative users to bookings which do not have appropriate resources allocated without conflicts, automates most routine notifications sent to staff, and provides a self-service interface for schedule information and shift requests.

Dedication

Dedicated to my late father, Dr. Richard D. Barnhart, a committed teacher of Computer Science and Mathematics who always reminded me that the only problem with computers is that they do exactly what they're told.

Acknowledgments

This project could not have been developed and completed without the support of my family, friends, colleagues, and advisors. Many thanks to Larry Bouthillier, my thesis director, for taking on an inexperienced student with a project in a little-known framework, and providing crucial pointers along the way.

Thanks are due as well to my professional colleagues for allowing me to beta test the StaffStuff system in our media production department and providing feedback and bug reports: Dr. Rebecca Nesson, Director of Online Course Content and Production; Rod Lindheim, Production Manager; Christian Wisecarver, Assistant Manager for Online Education Production; and all of our full time and part time production staff and student support technicians.

I stand on the shoulders of the programmers that have come before me, and so I sincerely thank the developers of the Legacy production system that the department used as it evolved over fifteen years, and who are developing new systems for us even now. Many of my ideas are patterned after architecture used by Dr. Bill Robinson and his engineers to represent our particular business logic. Special thanks as well to Rute Santos for explaining the underlying database and file structure little by little over the years, and for her assistance in evaluating existing third party software packages for staffing logistics.

To my friends Dr. Nickolas Falkner and Michael Zehr, my thanks for all your advice and support over the years and on this project, for always knowing what theory or

term applied to whatever problem I needed to solve when I didn't know where to begin, and especially to Nick for reading and giving notes on this thesis paper. Very specific thanks are due to Jason Parsons for suggestions on debugging a crontab entry.

Finally, to my husband Jonathan Rothberg and our family, thank you for making do without me every weekend and most evenings for the past six months, for bringing me hot meals, and for deploying or removing kittens as needed.

Table of Contents

List of Figures

Chapter 1 Introduction

One of the biggest challenges in managing a media production department is the efficient allocation of limited physical and human resources which must be shared among a number of projects. In small departments, the booking of these resources can usually be accomplished using ad hoc methods such as written calendars, sign-up sheets, and direct interpersonal communications. As the quantity of projects and resources increase, however, the volume and complexity of the requests become unmanageable using only such manual tracking; the process becomes time consuming and the results are prone to frequent human error.

Computerized databases are a long-standing solution for tracking inventories, and systems built around them are common choices for allocating resources according to complex parameters. Several products targeted to media production enterprises are currently available, with varying levels of sophistication and configurability; however, none of the systems reviewed could meet the exact needs of a client with an unusual departmental organization and business logic, without incurring substantial costs for customization of the underlying code.

StaffStuff is a system written specifically for one such client and as such is completely customized for their desired operational procedures. It models the logistical operations of the client's production department in a relational database, using custom algorithms and relationships when needed to accommodate unique business logic.

Advanced functions include a complex analysis of resource availability which matches job requirements to particular skill sets, while accounting for the employer's work schedule policies, the employee's availability, and conflicts with other resource requests. It automates many of the time consuming administrative tasks such as routine staff communications, and empowers all users to locate information and run reports designed for common use cases, without requiring assistance from a departmental administrator.

Chapter 2 Background

The client for this custom software system is a busy academic media production department which is expanding in both volume and the types of production encompassed. The majority of bookings are for video recording and online live video streaming of classroom lectures and meetings, but may also include studio recordings, field trips, editing sessions, web conferences, and more.

The logistics of the department had recently been handled using only common office tools such as spreadsheets, shared calendars, and email. Incoming requests for resources were recorded using a physical or online notepad, and the likelihood of resource availability for a given request was determined with a hand-drawn grid on a whiteboard in the office. Communication with potential staff for a booking was done directly and one at a time, by email or text message. Once the booking was arranged, the details would be entered in the spreadsheet for recurring bookings, or in a shared calendar for one-time requests. One-time cancellations of recurring bookings were also tracked in the shared calendar.

Figure 1 Whiteboard visualizing staff availability and weekly default schedule

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	day	subj	numb	crn	title	instr	Producer	begin	end	build	rm	shoot type	video-grapher	Start date	End date	Notes
93	W	JOUR	125 films	14758	FILM SCREENINGS: Journalism Ethics Through Film	Erlick	MM	1940	2140	SCICEN	D	tripod non-live	Scott	9/2/2015	11/18/2015	Dates: 9/2, 9/16, 10/7, 10/21, 11/4, 11/18
94	R	ANTH	1168	14743	Maya Glyphs	**Tokovinine**	PK	1730	1930	1STORY	302	CiaB live	Jake	9/3/2015	12/17	New instructor but was Zender's TA
95	R	Caution: booking this slot requires staffing adjustment						1940	2040	1STORY	302					
96	R	Caution: booking this slot requires staffing adjustment						2045	2145	1STORY	302					
97	R	ENVR	166	14545	Water Resources Management	Horsley	DJ	1730	1930	1STORY	303	Tripod live	Rich	9/3/2015	12/10/2015	
98	R	BIOS	157	14263	Viruses	**Bonham**	PK	1940	2140	1STORY	303			9/3/2015	12/3	New instructor
99	R	MGMT	5000	13351	Strategic Management	Chevarley	CF	1730	1930	1STORY	304	Ctl Rm live	Seth	9/3/2015	12/10/2015	No class on 10/1, 10/8, 10/22. Weekend intensive will not be recorded.
100	R	GOVT	1058	14748	Truth, Knowledge/Democracies	Robichaud	DJ	1940	2140	1STORY	304			9/3/2015	12/17/2015	
101	R	ENVR	118	13556	Environmental Mgt/Intl Tourism	Epler Wood	JO	1730	1930	1STORY	306	Ctl Rm live	Steve R	9/3/2015	12/10/2015	
102	R	CSCI	97	14306	Software Design: Principles	Gieseke	KK	1940	2140	1STORY	306			9/3/2015	12/17/2015	
103	R	MATH	15 sec	10436	GRAD SEMINAR (see note) -- Introduction to the Calculus A	Towne	MM	1625	1725	1STORY	307	Tripod live	JimN	9/10/2015	10/1/2015	Meets for 4 weeks, on 9/10, 9/17, 9/24, and 10/1.
104	R	MATH	10	12572	Precalculus	**Arias**	KB	1730	1930	1STORY	307			9/3/2015	12/10/2015	
105	R	STAT	139 sec	14787	SECTION -- Intro to Statistical Modeling	Rader	GA/KK	1940	2040	1STORY	307			9/10/2015	12/10/2015	
106	R							2045	2145	1STORY	307					
107	R	MGMT	5330	13439	Principles of Fundraising	F.White	CW	1730	1930	53CHUR	L01	Ctl Rm live	Bob	9/3/2015	12/17/2015	
108	R	ENVR	210	13757	Critical Analysis/Envr Systems	Leighton	JO	1940	2140	53CHUR	L01			9/3/2015	12/10/2015	
109	R	GOVT	1886	14188	Nuclear Weapon, Int'l Security	Nichols	DJ	1730	1930	BYERLY	013	Tripod live	Steve G	9/3/2015	12/10/2015	
110	R	MGMT	6040	14788	International Marketing	Nugent	KK	1940	2140	BYERLY	013			9/3/2015	12/10/2015	
111	R	SEAS		80000	CS Colloquia		RL	1600	1725	MAXDWK	G115	Ctl Rm live	Bill	9/10/2015	12/3/2015	select dates: 9/10, 9/17, 10/1, 10/8, 10/15, 10/22, 10/29, 11/5, 11/12, 12/3
112	R	STAT	100	14574	Intro to Quantitative Methods	Fosse	KE	1730	1930	MAXDWK	G115			9/3/2015	12/10/2015	
113	R	MGMT	1000	13339	Financial Accountng Principles	Haselkorn	GA/VQ	1940	2140	MAXDWK	G115			9/3/2015	12/10/2015	
114	R	GOVT	1897	14869	American Foreign Policy	Imparato	KK	1730	1930	MAXDWK	G125	Ctl Rm live	Ian	9/3/2015	12/10/2015	
115	R	BIOS	40 sec	13099	SECTION -- Introduction to Proteomics	Viel	(JO)	1940	2040	MAXDWK	G125			9/10/2015	12/10/2015	
116	R	ANTH	1050 sec	14065	SECTION -- Moctezuma's Mexico:Then, Now	Carrasco	(CF)	2045	2145	MAXDWK	G125			9/24/2015	11/19/2015	
117	R	MATH	15	10436	Introduction to the Calculus A	Towne	MM	1800	2100	HARVAR	104	Tripod live	ChrisM	9/3/2015	12/10/2015	no lecture 10/1, 11/12, 12/17 (exam days)
118	R	CHEM	1ax	14578	General Chemistry I (Lecture)	Tucci	KB	1800	2100	SCICEN	B	Tripod live	Cary	9/3/2015	12/10/2015	lectures from CHEM E-1a
119	R	ECON	1010 sec	10782	SECTION Microeconomic Theory	Neugeboren	(CF)	1715	1815	SEVER	210	Tripod non-live	Reb	9/3/2015	12/10/2015	
120	R	STAT	190 sec	14562	SECTION -- Quant Research Methodology	Blackwell	(PK)	1900	2000	CGIS Knafel	K050	Tripod non-live		9/3/2015	12/3/2015	
121	R	MGMT	1000	14267	Financial Accountng Principles	Pavlina	GA/KE	1730	1930	SEVER	113	Tripod non-live	Joe	9/3/2015	12/10/2015	
122	F	CHEM	1ax sec	14578	SECTION (weekly review)		(KB)	1630	1800	SCICEN	D	Tripod non-live	Cary	9/4/2015	12/4/2015	Some dates will meet in an alternate room.
123	F	BIOS	10	14563	Intro to Biochemistry	Haynes	JJ	1730	1930	SCICEN	B	Tripod live	Reb	9/4/2015	12/11/2015	Recording the lectures now for a Spring online offering.
124	F	STAT	104 sec	14527	SECTION - Intro/Quant Methods/Economics	Parzen	KE	1600	1700	1STORY	306	Ctl Rm live	Bill	9/11/2015	12/4/2015	
125	F	CSCI	90	14300	Cloud Computing	Djordjevic	KE	1730	1930	1STORY	306			9/4/2015	12/18/2015	
126	FAS CLASSES															
127	MW	HSCI	128	14820	Sci Fiction, Religion, Society	Ragab	MM	1000	1100	SEVER	102	Tripod non-live	Jason	9/2/2015	12/2/2015	History of Science 192 - catalog lists MWF but instructor will only meet on MW. -- course requires a second lav. Shoot screen for slides.
128	MW	PSYC	1356	14542	Evolution and Cognition	Krasnow	GA/VQ	1000	1100	WJAMES	105	Tripod non-live	Steve G	9/2/2015	12/2/2015	Psychology 1305

Figure 2 Shared spreadsheet (partial) detailing the Fall 2015-16 weekly default schedule

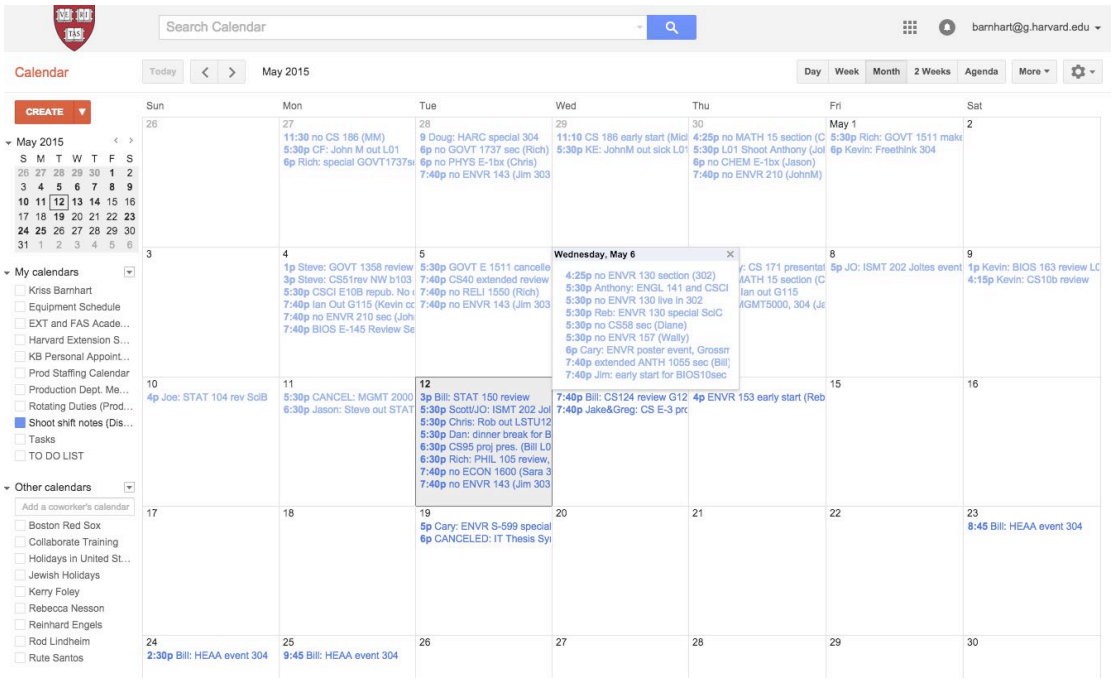

Figure 3 Shared online calendar tracking deviations from the weekly default schedule

When the department was small and the booking needs were relatively simple, this manual tracking process was adequate, but it did not scale with departmental growth. When the weekly default schedule was contained in a single document which could be understood by a human reader in a short time, it was reasonable for individuals to locate their assignments in this document without additional filtering, and to remember or make personal notes for the relatively few deviations from their routine. However, as the volume and complexity of projects increased, the length of the documents and the distribution of vital information across additional documents steadily increased as well. Human error in gathering and interpreting the data to determine the details of a specific booking resulted in missed assignments and lowered staff morale.

As the size of the staff increased, the established communications and negotiation practices also did not scale. Locating available personnel via multiple personal conversations had the advantage of strengthening human relationships among colleagues, but was inefficient. The man-hours needed to handle each instance of deviation from the default schedule individually via traditional interpersonal communications exceeded the available resources.

Chapter 3 Prior Work

Review of Existing Software Products

During the period from Summer 2014 to Summer 2015, the production department researched and viewed demonstrations of various scheduling software, including ScheduALL, farmersWIFE, Xytech MediaPulse, WebCheckOut, and PlanDay.

The marketing materials for ScheduALL's Operational Management module described it as addressing many of the problems that we had identified in our current scheduling process (SchedulALL, 2015). However, the product is targeted at extremely large organizations with many divisions, such as national broadcast networks. While it might be sufficient for most of our needs, the interface appeared complex and the product was more expensive than the department could afford.

WebCheckOut is designed primarily for equipment reservation and circulation, but has an add-on Personnel Scheduling Module (WebCheckOut, Inc., 2015). However, the module is intended for straightforward shift work such as staffing a circulation desk, and does not handle the more project-like structure of scheduling shoots at multiple locations for a given course with many metadata fields. It also does not allow the level of detail needed to track different skill sets and match them to these bookings whose needs may change.

PlanDay's market is the hospitality industry, not media production, but many of its functions strongly resembled the system that we needed. The system allowed a weekly template schedule as a starting point, and then variations on that default for a given date. The system clearly identified positions that were unstaffed on a given shift, and tracked which employees had the specific qualifications to fill each position. Employees could log into the system and view their personal shift schedule. (PlanDay, 2015) However, the design did not include custom data fields, and mapping the data fields for the restaurant shift schedule to the more elaborate metadata for production projects, bookings, and resources would have required combining many of our desired fields into only a few text fields. There was also no provision for injecting or extracting large amounts of data without manual data entry. We contacted PlanDay to inquire if the company would be interested in customizing the product to fit our needs, but they declined, citing their need to focus on their target market.

Xytech MediaPulse was a much more promising product for our scale and workflow structure. The GUI involves a timeline for the user to drag and drop items for easy booking, to group items, and to define different types of shoot. (Xytech Systems, 2015) We did not request a trial installation of MediaPulse mainly because the company was very slow to respond to requests for documentation about the API, and the eventual documentation sent was incomplete and over five years old. Repeated requests for information and even the trial contract were ignored for months. The management team did not feel that this was a good choice going forward.

FarmersWIFE is a very similar product to the Xytech software and the company is small and responsive. They are willing to write custom code for WIFE to interface with

the registrar's system and our recording system, for an additional fee, or provide information so our own engineers can write it. It is highly customizable, and has essentially unlimited custom fields. (FarmersWIFE, 2015) We ordered a cloud based trial installation and a full week of training workshops to set up our custom configuration.

Unfortunately, the user interface for WIFE was very technical and cumbersome. The user must understand computer science concepts such as objects, classes, and trees in order to merely navigate the menu options. While it is ostensibly a GUI with drag and drop actions, the actions do not always have the intuitive result, and there is no "undo" capability at all. For example, there are confirmation popups for some actions, such as canceling a booking, but the wording in the message is unclear as to what is being canceled (e.g., the assigned staff or the whole booking?). If the user accidentally schedules a conflict, the system offers to resolve it, but does so by canceling the old booking instead of the new one. It is not efficient to use; for example, it is not possible to select several bookings, such as "the rest of the semester" and change one aspect of them such as the room assignment, unless you create the booking as one single ongoing event. However, if you use the latter method, it is difficult to make adjustments to one particular day of the term, such as for absence coverage, and it is easy to accidentally delete an entire semester worth of historical information about a course, instead of just one day, with no "undo." It would not be a simple task to train additional staff in the department how to use the software without risk of accidentally deleting important information that can only be recovered by requesting a copy of a recent database backup.

There is a severe limitation in WIFE's ability to generate views and reports: it will only generate a report on an actual object in the database. While system users, rooms, and

pieces of equipment are objects, projects and bookings are not and people listed in the address book, such as clients or actors, are not. Thus, while one can view all of the bookings for a particular camcorder, one cannot view all the bookings for the course "Introduction to Statistics." One can view all the bookings for videographer J. Smith, but one cannot view a list of the bookings that have no videographer assigned. One can view all the bookings in room 302, but one cannot view all bookings that are happening today.

Because of this flaw, WIFE does not solve one of the most fundamental problems in the department; it does not allow the user to define what type of staffing is required for a booking and highlight it in a way that it will not be overlooked until the requirement is filled. In fact, precisely because no object is assigned, unstaffed shoots are more likely to vanish from the user's object-based view than they are to command attention. The software trainer suggested a workaround, where the user creates a dummy object and assigns it to any booking that is not completely staffed, and then views the dummy's schedule. However, if the user forgets to replace the videographer user object in the booking with the dummy rather than deleting it, the booking vanishes from the view and the user must think of another way to locate it and add the dummy object back. Even with this workaround, the WIFE system risks adding more human error to the booking process instead of preventing it, and does not ameliorate the need to keep track of bookings in other ways such as written notes and separate calendars in case information is lost.

The Legacy Production System

Until August 2015, the production department utilized a proprietary system written in-house to control the scheduling and recording of course meetings, which we

refer to as “the Legacy System.” Its purpose was primarily media file management: scheduling digital recordings to begin and end, allowing basic editing of the media files, transcoding and generating web-friendly presentation formats, and transferring files to the servers. Although it included a data field to note which person was assigned to staff a given class meeting, this feature was so rudimentary that it fell into disuse eight years ago. However, it is worth looking at this system because the database and file directory structure was nearly ideal for modeling the department’s business logic.

The object models in the Legacy System were:

- Offering: a college course that was to be recorded for online viewing
- Presentation: a recorded class session, produced into a format viewable online
- Presentation Default: Each Offering could have a Default set for three recording types: lectures, sections, and projects. Projects were any sort of recording that was not a lecture or section, such as a review or a field trip. Changing the information for a Default updated the information of any Presentations of that type which were not yet in progress.
- Buildings, Rooms: locations on campus
- Users: Any person in the database whether they could log in or not, including students, staff, faculty, guests.

When setting up a semester, the system imported the data for the Offerings and the Lecture Defaults from the registrar’s database, including associated Users such as the course instructor and the associated Building and Room for the Lecture Default. Data for

Section Defaults, and Project Defaults if applicable, is normally not available when the setup is run, so users would enter it by hand on an ongoing basis. Shortly before the term began, the system's presentation generation would run, creating the directory structure on the local and public servers where the presentation files would be located, as well as a row for each upcoming presentation in the database where its media files' metadata would be stored. Presentation data included the date, start and end times, and locations of the class meetings, and could be individually edited if there was a change for one date, although editing the Default's information would change it back to match the default. The Legacy System used this metadata to start and end digital recordings on networked encoders associated with each classroom.

The legacy Offering model corresponds to what most scheduling software would call a Project, which to a less specialized department could be a television series, or a film, or a short news feature. The Presentation would correspond to a Booking, which would be a particular studio session or editing shift, or in this case a recording of a classroom meeting. Less common in the scheduling software reviewed, but absolutely vital to efficiency in this department is the idea of a Default, which enables us to generate a number of identical Bookings on particular days of the week over a given period of time such as a semester, and then change a given attribute of those Bookings, such as their time or location, by editing their Default.

In order to identify a specific Presentation, the Legacy System required assignment of a "typenum" code unique to that booking within the Offering, consisting of three characters: the letter which identifies the type of Presentation and then two digits, such as "L01," "L02", "S01," "P14." Each offering used the Registrar's five digit

"CRN," the Course Registration Number code, to uniquely identify it within a semester. The semester is represented by a two digit code: 01 for Fall, 02 for Spring, and 03 for Summer School, and the academic year. Thus, the naming convention to identify a specific Presentation consisted of the year, semester, CRN, and typenum, such as 201502-23517-L01 for Lecture 1 of the course with CRN 23517 in the Spring term of the 2014-2015 academic year. This naming convention was used in the Legacy servers' directory structure and file names, generating predictable URLs for online resources, and setting the rules for file access authorization.

As of January 2016, the Legacy System was completely depreciated as the primary recording, file storage, and content delivery system for the department. However, the migration to the new system, called Opencast Matterhorn (Opencast, 2016), is not complete. The version of Matterhorn currently used by the client does not yet have a good user interface for scheduling recordings or any means of setting access authorization rules for the public files. As a stopgap, code was written to propagate Offering and Presentation metadata from the Legacy System into Matterhorn, incidentally preserving many of the naming conventions. The authorization rules are still entirely generated using the Legacy UI and its file naming conventions.

Chapter 4 Technology Choices

Given the ambitious scope of the project and the limited time frame, we decided that I would work in a familiar programming language, PHP, and that I should use a framework so that I would begin with some basic functionality in place, such as user authentication and security which would have been as time consuming to build well. I had no prior experience with any of the popular PHP frameworks, so I evaluated informal online user reviews of CodeIgniter, CakePHP, and Laravel. Most users had a preference for one or another, but consensus appeared to be that while there were some minor pros and cons for each, none was objectively superior to the others for general purposes (Laravel.io, 2014). I selected Laravel mainly due its recent rise in popularity (Web Revisions, 2015).

Use of a framework automatically provided an architecture to organize my code. Most PHP frameworks, including Laravel, are object oriented and designed in a classic Model View Controller pattern. This architecture and the heavy emphasis on efficiently using the relationships between the data models, helped refine the way I approached representing the real world in the data.

I set up my local development environment on a Mac laptop with MAMP running my application in MySQL and PHP 5.6.10, which was the latest version at the time. I chose the Sublime Text editor, Cyberduck ftp client, and the native Mac OSX Terminal for command line operations and eventual ssh to my production environment. My GitHub

code repository served as off site backup and version control. I kept Chrome, Safari, and Firefox updated to the most current versions for testing throughout development.

Being unfamiliar with Laravel and never having used tools such as a dependency manager, I followed step-by-step instructions from a Web Developer Bonanza tutorial (Flynsarmy, 2015) to install the PHP dependency manager Composer in my local environment, and Laravel 5.1.10 which was the current version in late August 2015.

I purchased the domain "staff-stuff.com" and set it up with a placeholder page in my web hosting account at Dreamhost. I also set up subdomains for a "sandbox" installation to be used for testing and demonstrations, and a second subdomain to be used for the client's real installation.

Chapter 5 System Overview

User Experience

The StaffStuff system is a customized experience for each user type, which allows the user to navigate efficiently and intuitively to their most likely desired features. When users log in, the landing page is a dashboard which already displays the information most likely needed immediately, and quick links to the next most likely desired features. Choices of which information and features should be provided to each user type are based on the user stories enumerated in Appendix 1.

All employees' dashboards prominently display a schedule of bookings where the user is personally required to be in the current week, including the times and locations with links to detailed information. The dashboard also displays any notifications not yet emailed to the user, and the links to accept or decline offers of work shifts. These are the most frequently required items for the part time staff, and are also commonly needed by other users.

For the full time staff, the dashboard also includes a list of direct links to the projects that the user is actively working on, the current day's schedule of bookings, and buttons which execute the next most commonly run queries to the database.

Administrative users' dashboards have the above features, plus an area listing pending items that require an administrator's approval. At the bottom of the screen,

administrators have the form to run a more advanced query, which will be discussed in the section on Schedule Views.

Features and information not included on the dashboard are accessed via tabs along the top of the screen. For information on a person, place, project, or booking, the user selects the corresponding tab, and then may search for it using a form on the tab's landing page. The search results are displayed as a list of links to the profiles of the individual items. Each profile contains direct links to all other associated profiles. For example, a user might search for a Biology class in the Projects tab, and select the current semester's offering of Biology 101 from the results to view its profile page. The project profile displays links to the people involved, and each of those people's pages has links to their other projects. The page for a booking contains links to its location's profile, its parent project's profile, as well as the involved people. This web of linked information minimizes the number of steps a user must click through to answer the various questions in their user stories.

The Schedule tab displays sets of bookings in various formats including a list, a timeline graphic, and a calendar table. Bookings displayed are the results of a query based on a "Power Search" form filled out by the user, to select which dates and kinds of bookings they need to see in the schedule.

Administrative users have additional links throughout these pages which allow the editing of existing profiles and the creation of new ones, and the assigning of users to the jobs required for each project and booking. Advanced features include a staffing analysis for each job which identifies which users are qualified and available to work, detailed in

the “Staffing Analysis Table” section below. Most changes to bookings trigger automatic notifications to affected staff, which is discussed in the “Notifications” section.

Finally, the “Admin” menu tab which is restricted to admin users contains additional convenience features, such as bulk data upload of projects and weekly booking defaults, automatic generation of various administrative checklists, and calculation of staff hours worked.

Relational Database Structure and Models

Although it should not be obvious to the everyday user, the key to understanding the underlying StaffStuff code behind these user experiences lies in the data modeling of the client’s business logic. The Laravel framework provides a typical model-view-controller architecture. Generally, each table in the database is represented by a model object in the code, so we may refer to attributes of the model with the assumption that these will correspond to columns in the table. The relational aspect of the database is represented by defining ‘relationships’ between the models in the code (Otwell, 2015).

In the Laravel framework, the relationships are defined in each model as “hasOne,” “hasMany,” “belongsTo,” or “belongsToMany.” If object A “belongsTo” Object B, there will be a foreign key in table A which refers to table B. For example, if a Booking “belongsTo” a Project, the bookings table has a foreign key to the project_id. Conversely, if Object X “hasOne” or “hasMany” of Object Y, then table Y will have a foreign key which refers to table X and we can expect either one or many rows to contain that key. For example, if a Project “hasMany” Bookings, we can expect that project_id may occur as a foreign key in multiple records in the bookings table. Finally, the

“belongsToMany” relationship is mutual and represents a pivot table in the database. For example, each User may have a number of Qualifications, and each Qualification may be possessed by many different Users.

Once defined, these relationships allow the programmer to refer to attributes of the associated objects. For example, one can retrieve the room number of a lecture’s location, where the lecture is a Booking object named $myLecture, by calling $myLecture->location->room_number. The room_number is an attribute of the Location object which has a relationship with myLecture. One can retrieve a collection, which is much like an array, of all the Booking objects associated with a given Project called $myProject by calling $myProject->bookings.

The business logic is represented thus in the models and their defined relationships.

Places

- LocationType: a classification of location, such as “Classroom with control room” or “Studio.” Attributes include the default staffing needs, such as whether a booking there is likely to need a control room operator or a field crew.
    - hasMany Locations
- Location: a specific place, such as “Sever Hall, room 202.”
    - belongsTo one LocationType
    - hasMany Bookings
    - hasMany BookingDefaults.

People

- Stafftype: type of position held in the department, such as “full time producer” or “LHT” (a part time employee). This affects the number of hours per week the person may work, and whether they should be assigned to certain project roles or booking roles.
    - hasMany Users
- Usertype: type of StaffStuff account, such as “Admin” or “No login.”
    - hasMany Users.
- User: a person, such as “Kriss Barnhart.”
    - hasOne Prefs
    - belongsTo one Stafftype
    - belongsTo one Usertype
    - belongsTo one CellCarrier
    - belongsToMany Qualifications
    - hasMany ProjectRoleAssignments
    - hasMany BookingRoleAssignments
    - hasMany BookingDefaultRoleAssignments
    - hasMany JobRequests
- Prefs: account preferences.
    - belongsTo one User

- Qualification: a skill that Users can have, such as the ability to work with specific control room equipment, or advanced level skills in animation.
    - belongsToMany Users (the many-to-many relationship data is stored in a separate pivot table)
- CellCarrier: a cellular phone company, whose data is needed for the text messaging feature.
    - belongsToMany Users
- Notification: an email being held until the end of the day, to be sent as a digest.
    - belongsTo one User

Projects

- ProjectType: type of Project, such as “Online course” or “Promotional packages.”
    - hasMany Projects
- Project: a media production project, such as “Biology 101” or “Professional Programs Promo Packages.”
    - belongsTo one ProjectType
    - hasMany ProjectRoleAssignments
    - hasMany BookingDefaults
    - hasMany Bookings
- ProjectRole: jobs which might be required for Projects, such as the course instructor, producer, or lead editor.

    - belongsTo one Qualification
    - hasMany ProjectRoleAssignments

- ProjectRoleAssignment: record of which User is assigned to a ProjectRole in a specific Project.
    - belongsTo one Project
    - belongsTo one ProjectRole
    - belongsTo one Qualification
    - belongsTo one User.

Bookings

- BookingType: a type of Booking, such as “Studio session” or “Classroom recording.”
    - hasMany Bookings
    - hasMany BookingDefaults
- Booking: a production shift in a given Project, such as the filming of a classroom lecture, a studio recording, or an editing session.
    - belongsTo one BookingType
    - belongsTo one Project
    - belongsTo one Location
    - belongsTo one BookingDefault
    - hasMany BookingRoleAssignments

- BookingRole: jobs which might required for Bookings, such as a camera operator or audio technician.
    - belongsTo one Qualification
    - hasMany BookingDefaultRoleAssignments
    - hasMany BookingRoleAssignments
- BookingRoleAssignment: record of which User is assigned to a BookingRole in a specific Booking.
    - belongsTo one Booking
    - belongsTo one BookingRole
    - belongsTo one Qualification
    - belongsTo one User
    - hasMany JobRequests
- JobRequest: a record that a given User has either been offered or is interested in a BookingRoleAssignment.
    - belongsTo one BookingRoleAssignment
    - belongsTo one User[1]
- BookingDefault: the data for a set of recurring regular Bookings, such as a weekly meeting or lecture series.
    - belongsTo one BookingType

    - belongsTo one Project
    - belongsTo one Location
    - hasMany Bookings
    - hasMany BookingDefaultRoleAssignments
- BookingDefaultRoleAssignment: record in the BookingDefault of which User is usually assigned to a BookingRole.
    - belongsTo one BookingDefault
    - belongsTo one BookingRole
    - belongsTo one Qualification
    - belongsTo one User
- Holiday: a date which is skipped when automatically generating Bookings from BookingDefaults
    - no defined relationships

Additional tables

- migrations: for use by Laravel in tracking the creation of and changes to the database structure
- password_resets: table created by the Laravel auth package but not used in this application

[1] There is also a foreign key in this table to a User who is the administrator either offering the job or approving the request, but this relationship is not explicitly defined in the model because the User relationship is already taken by the person who would be assigned to the role.

- qualifications_users: pivot table for many-to-many relationship

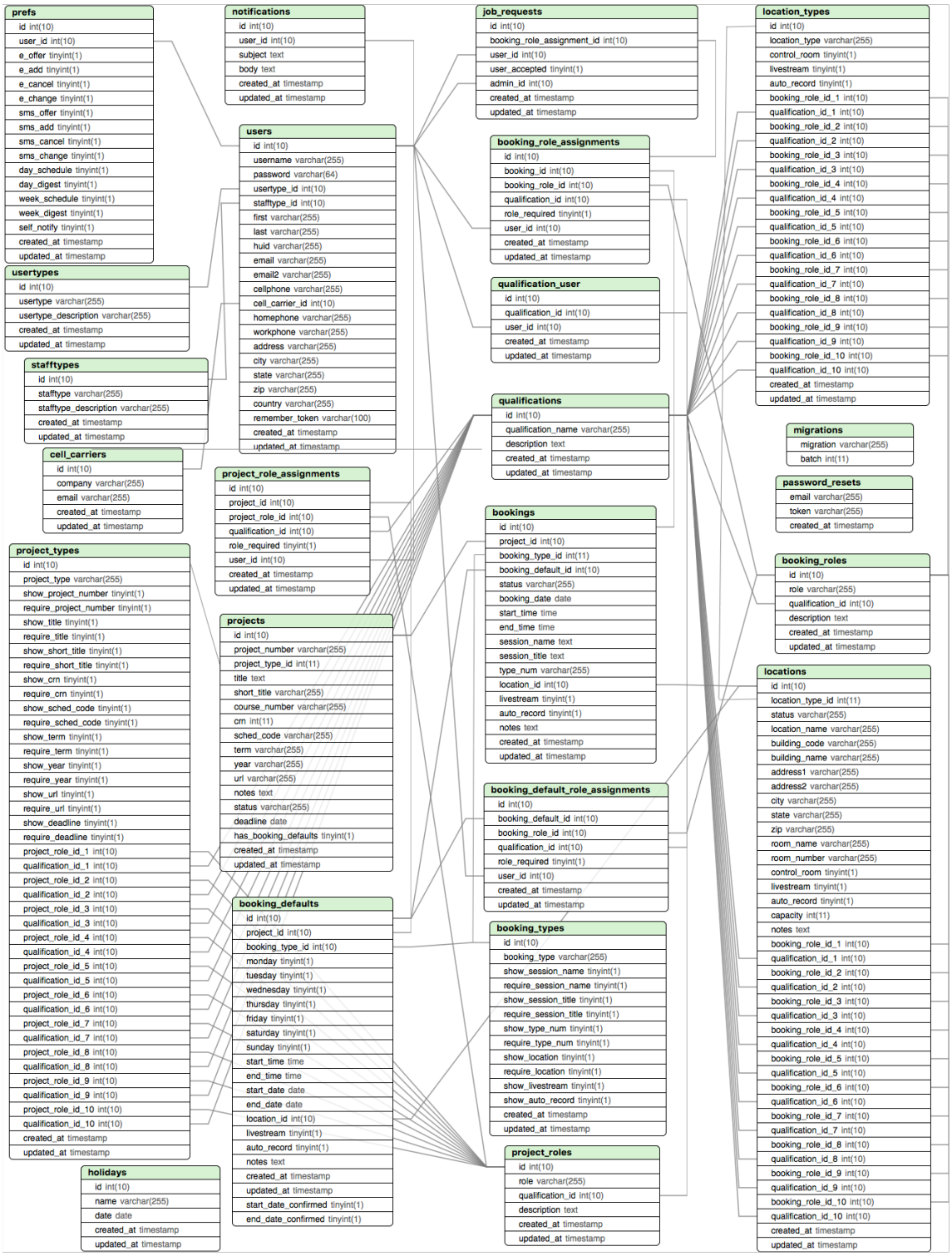

Figure 4 Database Structure Diagram

Construction of this relational database is the first task in the implementation process below, and in turn, its models and their corresponding controllers organize the code which implements the system's features.

Chapter 6 Implementation

Laravel Fundamentals and Database Setup

Due to my unfamiliarity with the model-view-controller, or MVC, architecture used by the Laravel framework, I continued in the Web Developer Bonanza tutorial (Flynsarmy, 2015) to work through the fundamentals of creating routes, binding models to the forms used to create and update them, and defining the relationships between models. It was also a test run for setting up users and user authorization, which is lauded as being included "out of the box" in Laravel (Otwell, 2015).

Once I felt comfortable with setting up a sample web based application, I began work on my own application, StaffStuff. I set up the users table and auth similarly to the tutorial, although I decided not to allow users to sign up for accounts themselves with a "Register" feature. Since the StaffStuff system is for internal departmental use only, it makes more sense in the business structure for an administrator to create accounts. Similarly, I decided to skip the feature to email password resets to users and simply have the administrator reset those as well, although if there is demand for this feature it may be implemented later.

I wrote the additional database migration files that would create the rest of the tables in MySQL, and the data seeding files which would set up a few records in each table for testing purposes. The intricacy of these relationships and my own inexperience

using migrations caused this process and the debugging to take several days, as I worked out such details as how to express the parameters for the foreign keys and in what order the tables must be created so as not to cause foreign key errors. This initial version did not include Holidays or JobRequests, or the pivot table for Users and Qualifications, which I learned about later in this process.

Laravel includes a command line interface called Artisan, and one of its uses is to generate templates for common classes such as models and controllers which extend the framework's Model and Controller classes. The model template includes little other than defining the namespace and the name of the new class. The controller template includes these plus placeholders for the basic database create-read-update-delete (CRUD) functions, as well as an index function which runs when the user is routed to a controller with no function specified. Using Artisan, I created the models for each table and defined the relationships as noted above, and created the corresponding controllers for each model. I later added an AdminController and a ScheduleController to control the views and run the analyses required for those UI menu tabs, described below.

User Interface Design, and Restricting Access to Content

Next, I sketched and then coded the basic UI layout. The Laravel framework uses a templating engine called Blade, which allows easy nesting of templates, display of data passed in by the Controller, and basic logic such as conditionals and loops within the View (Otwell, 2015).

This first pass included the master layout page which would contain the common HTML header information, visible page header and main menu tabs, as well as the

container div for whichever view the user currently requires. I reworked a CSS file which I had developed for previous projects, modernizing the tab shape and color scheme, and designed a simple StaffStuff logo for the project.

Since one of the problems we identified with existing products was the technical vocabulary needed to understand them, I grouped the system's navigation into tabs that I labeled as plainly as possible: People, Places, Projects, Bookings, Schedule, and Dashboard. System administrators have an additional tab labeled Admin. The logout function and a quick link to the user's own profile called "My Account" are always visible in the upper right of the browser window. I created placeholders for the content on each of these tabs until I could build the underlying functionality.

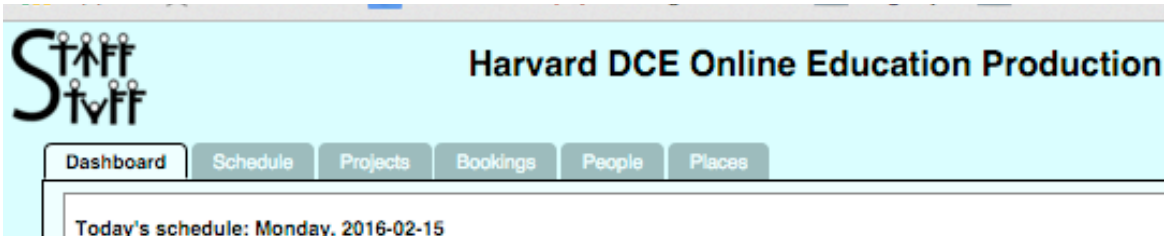

Figure 5 Main menu tabs

At this point, I began to build in the control of which content can be accessed by different types of user. The content of a page can be determined at the Route/Middleware level by completely disallowing access to a URI (Universal Resource Indicator) path, at the Controller level by having a function return a different View or completely rerouting the user, or within a View using the conditionals provided by Blade. I needed to revisit several times how I set this up, as my understanding of Middleware improved.

In the Laravel framework, the routes.php file lists the URIs that a browser might request, and to which controllers and functions they then direct the user. The function to

check whether a user should be permitted access to a particular route is defined in a "Middleware" class, for example, a function to check whether or not a user is of the "admin" type. Routes can be organized into groups and the whole group associated with a given Middleware class. I set up Middleware classes to check a user's type and allow access accordingly to admins, producers, LHTs (part time staff), and those with read-only access, and placed the URI routes into the appropriate groups. Most routes which are to directly create and edit database records are only available to admin level users. Routes needed to request or accept a job offer are not available to read-only users, and so forth.

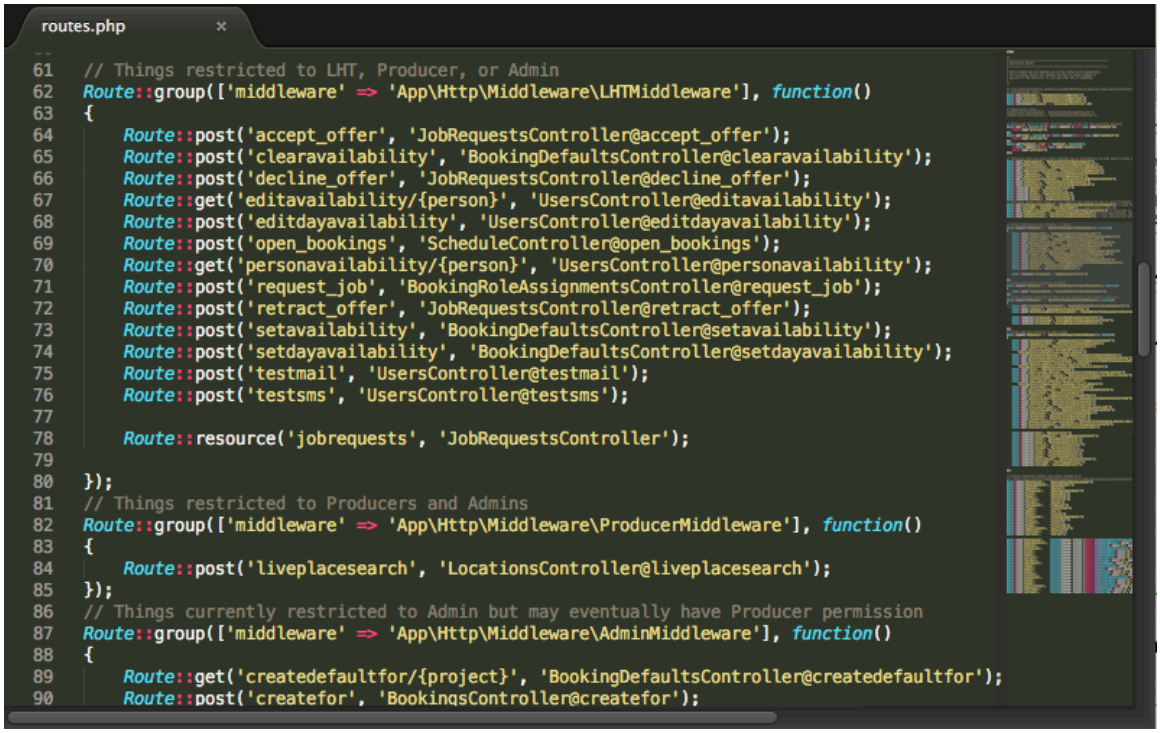

Figure 6 Sample of routes restricted by Middleware

For finer control, I mostly opted to run checks within the Blade view rather than in the Controller functions. For example, when a user views another user's profile, the UserController passes the entire User model to the view. The view then contains Blade

conditionals that will display the address and telephone number only if the requesting user is an admin level user, or is the user whose profile is being displayed. Thus, where $user is the logged in user and $person is the model whose information is displayed:

```
@if (($user->admin()) || ($user->id == $person->id))
      <p>
      Street Address: <br />
      {{ $person->address }} <br />
      {{ $person->city }}, {{ $person->state }}
      {{ $person->zip }}, {{ $person->country }}
      </p>
@endif
```

This customized content is not, however, only to restrict access to certain information. It can also help users get to the information they need quickly. One of the problems which we had identified in the existing products was that it could be cumbersome or even impossible to view data in such a way that would answer a common question, such as “who on the staff is available and qualified to do this particular job at this particular time?” Inspired by the “user stories” concept used in Agile style software development (Agile Alliance, 2013), I decided to focus on these questions when planning how a user would experience the StaffStuff site. I listed the most common questions each type of user might be asking (see Appendix 1), and considered how they would intuitively try to answer them based on the options in front of them, with the goal of answering any given question clearly in very few clicks. The content on the Dashboard tab, which also acts as the system’s home page or landing page upon login, varies according to the user’s type and is designed to expedite answering the most common questions for that user.

For example, the most common question from a part time videographer hired to record classroom lectures might be “What is my schedule today?” or “Where do I need to be next?” To answer this, I placed a box on the dashboard for LHT level users which contains their schedule of bookings for the current and following day, in a size and shape clearly readable on a smartphone screen, with links to further details. The most common question from a producer might be “Who filmed this footage I’m viewing?” or “Was my special event booking approved?” For producers, their dashboard includes links to all their active projects. Upon clicking through to the project’s profile, they view a listing of all the associated bookings and their status. Clicking a booking takes them through to its details, including all the people involved and their roles.

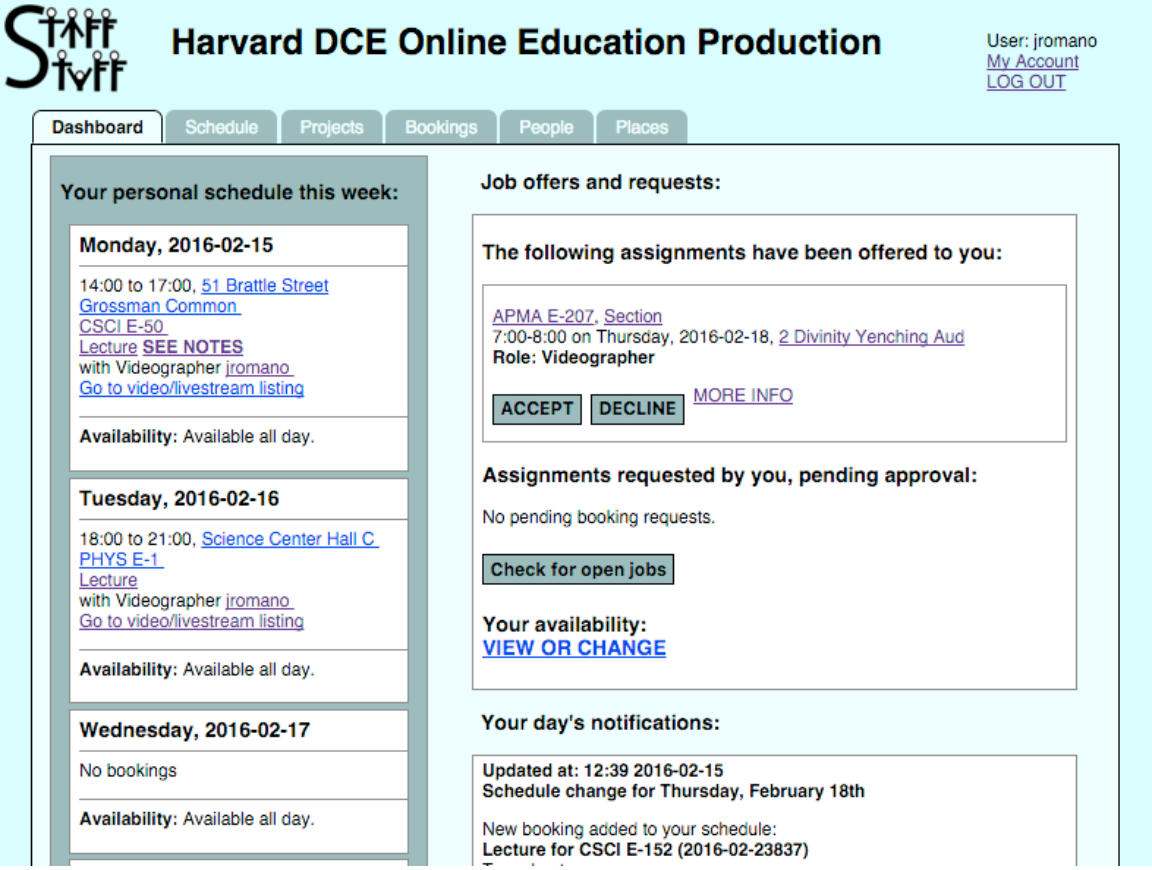

Figure 7 Dashboard for LHT user, including clear personal schedule

Thus for an LHT type user, the content in the template (Figure 7) is defined as follows, by including the re-usable partials which address their most immediate needs:

```
<? /* LHT view */ ?>
@if ($user->lht())
  <div id='lht_schedule'>
    @include('/partials/my_week_schedule')
  </div>
  <div id='lht_extras'>
    <h3>Job offers and requests:</h3>
    <div class='pending_items'>
      @include('/partials/pending_items')
      @include('/partials/check_jobs_button')
    </div>
    @include('/partials/view_queued_notifications')
  </div>
@endif
```

The dashboard for an Admin, Producer, or Read-only user would simply include different partials as appropriate to answer their most common needs (Figure 8).

I revisited the idea of the one-click answer to the most asked questions later in the project after I had built the search functionality, by adding “quick search” buttons to the dashboard which use hidden forms to request such things as a timeline graphic or list of today’s bookings, a calendar of only the bookings associated with a producer’s own projects, or for admins, a list of only the potential problems in an upcoming week such as unstaffed positions or location conflicts.

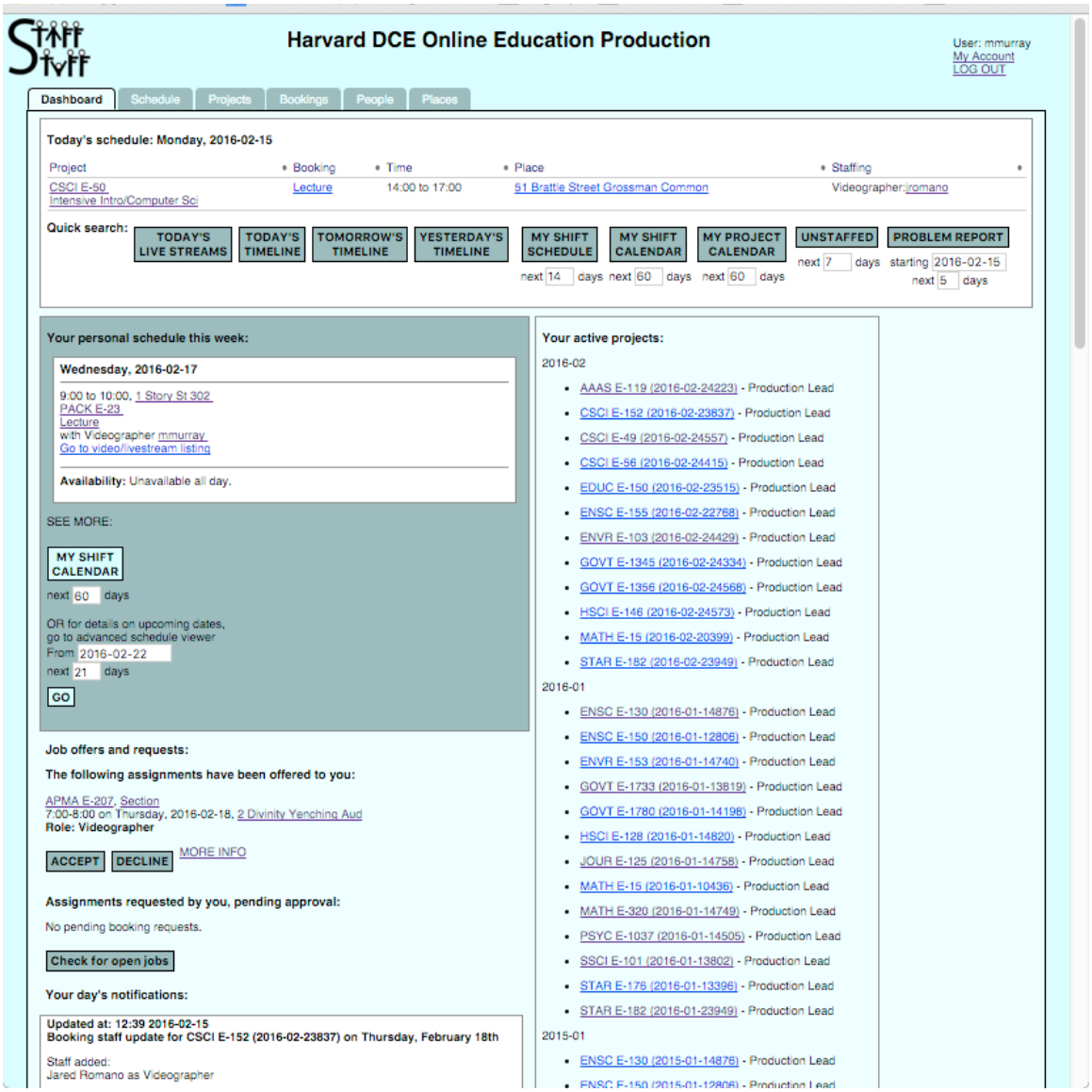

Figure 8 Dashboard for Producer user, including active projects and Quick Search

I made it as easy as possible for the user to drill down for further details by making almost every reference to another object into a link to its profile. Every person that is mentioned, every booking, every project, and every location is a link to that profile. Thus, a user reading their schedule who is not familiar with a location named, can click through to its profile to find the address, a map link, and any comments. If a producer needs to contact the videographer working a particular booking, they can click

on the person's name to find their contact information and their current schedule. This minimizes the amount of searching and streamlines the user's navigation to exactly the information they need.

The CRUD Basics

Once the UI layout was planned and placeholders for the content had been coded, I began work on the basic functions of each controller to create, read, update, and delete records in the database via the models' functions. In Laravel, the functions included in every Controller template are

- create() - returns the view that contains the form used to create a new record, including any data needed for the view such as the items in dropdown menus.
- store() - processes the form from the create view, saves the record, and returns a view to the user, usually by rerouting them to show() the newly created model.
- show() - returns the profile view for a given model.
- edit() - returns the view containing the form to edit the attributes of the model.
- update() - processes the form from the edit view, saving the changes to the database, and returns a view to the user, usually by rerouting them to show() the updated model.
- destroy() - depending on configuration, either deletes the database record or "soft deletes" it by setting a flag without actually deleting the record.

Later in the project I added code to these basic functions to implement advanced features such as notifying staff of changes to their schedules, but in this first phase I

concentrated simply on the functionality of creating records, displaying them as the models' profiles, editing them, and deleting them if desired.

The UsersController was partially written during the tutorials in my first few weeks learning Laravel, which included registering (creating) users and showing and editing their profiles (Flynsarmy, 2015). Some models such as JobRequests and Notifications will not have any direct CRUD manipulation at all, and so these functions in their Controllers remain undefined. Prefs, the user's notification preferences, are controlled entirely in the UsersController by elaborating on its CRUD functions, and the Pref model does not even have a Controller of its own. Most CRUD functions however needed to be written. I began with the most straightforward: Locations, Bookings, and Projects, and their associated types.

LocationTypes/Locations, BookingTypes/Bookings, and ProjectTypes/Projects

Part of the database seeding was the records for a few LocationTypes, BookingTypes, and ProjectTypes. These models exist not only so that one can search easily for which items are in a particular category, but also as a shortcut for the likely attributes of an item in that category while it is being created. The ability to create and edit LocationTypes, BookingTypes, and ProjectTypes is restricted to administrators on the system, but is otherwise very straightforward.

The user creating a Location, Project, or Booking must first choose what type it will be, but is given the option to deviate from what the default values would be for an object of that type. For example, when creating a new Location, the user must first select what LocationType this Location will belong to. This allows the system to pre-populate

some fields in the create form, such as whether that location has the technical capability to broadcast live, and allows the user to opt out entirely from other steps such as defining what the usual staffing requirements are for the location. For example, unless told otherwise, a classroom with a control room attached will have staffing needs defined as one Control Room Operator, whereas a typical studio might need two Camera Operators, one Audio Engineer, and a Director. However, this particular studio may belong to the LocationType studio but this one needs three Camera Operators and no dedicated Audio Engineer, so the user creating it would opt to set those explicitly rather than retain the default values from the LocationType. Also defined by the type is whether or not certain fields are required, for example, a project of type Online Course might require that the user enter a registration code and course number, whereas a project of type Video Packages would not require these fields. LocationTypes, BookingTypes, and ProjectTypes can be created and edited from the Admin tab, but cannot be deleted from within the UI due to the problems that arise in the database when records associated with foreign keys are deleted. Even editing these models carries heavy warnings in the user's view, since doing so will affect large numbers of associated items that draw their attributes from the type.

Unique Identifiers for Projects and Bookings

In keeping with the business logic of this client, part of the record for a project or booking are the unique identifiers which are compatible with existing conventions and systems, such as the Opencast Matterhorn automated video recording system, which in turn was designed to be compatible with the Legacy System. The ProjectsController and BookingsController contain helper functions to construct these identifier numbers, which

are called by the store() function while processing the form input before saving it. The ProjectsController creates its project number from the year, term and CRN of a course, the way the Legacy system did. The BookingsController has a function which can determine the next available "typenum" for its Project and BookingType, and assign it to the Booking.

Making BookingRoleAssignments and ProjectRoleAssignments

BookingRoleAssignments and ProjectRoleAssignments are not created or edited directly, so the create() and edit() functions which would provide the form views are not defined in their Controllers, although the store() and update() functions which process form input are. The forms to create and edit are included on the pages which create or edit the Booking or Project they belong to. When a Booking or Project is created, the user is next asked to select the needed staffing for that Booking or Project and it is this information which is saved as BookingRoleAssignments or ProjectRoleAssignments. For example, if a Booking will require two Camera Operators and a Director, three BookingRoleAssignments will also be stored. Each has a "belongsTo" relationship saved as a foreign key for the Booking, the BookingRole that describes the job, the Qualification that is needed to do it, and once assigned, the User that will do the job. While unstaffed, the User is null. The BookingRole and Qualification are selected from dropdown menus in the form, to be sure they correspond to existing models. Because a BookingRole model has a Qualification defined in it, the user has the option of either allowing the BookingRoleAssignment to default to the same Qualification as its BookingRole does, or of setting a different one for this particular BookingRoleAssignment.

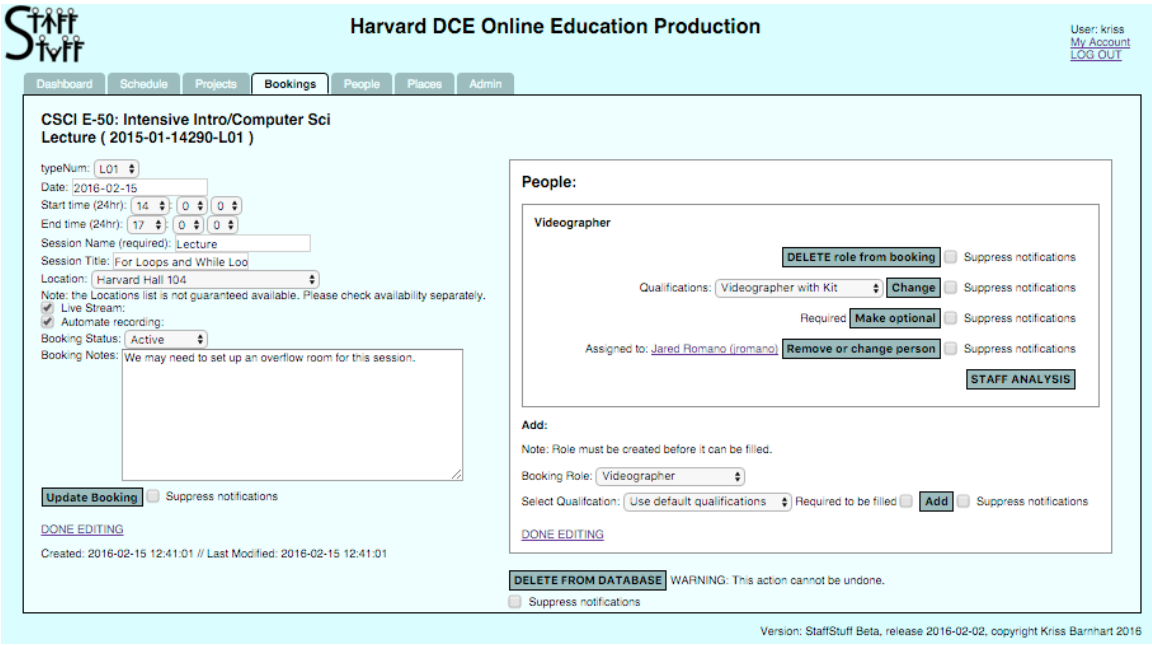

Figure 9 Edit form for Bookings and their BookingRoleAsignments

Furthermore, since a Location has BookingRoles and Qualifications recommended as part of its attributes, if a Booking has a Location set when it is created, then the user may select a checkbox to have the BookingRoleAssignments created automatically based on the Location's recommendations, and skip the step of defining what staffing will be needed.

To add, edit, or delete ProjectRoleAssignments and BookingRoleAssignments the user must navigate back to the edit screen for their Project or Booking.

BookingDefaults and BookingDefaultRoleAssignments

A key feature lacking from most of the existing products that we reviewed was a simple way to create a default schedule for a given week in the semester, yet be able to deviate from it. I will more deeply examine the behavior of a BookingDefault and the Bookings which belong to it below in the section on setting up a semester. For the basic

CRUD functions however, the user accesses the screens through the Project to which the BookingDefault belongs and its information is displayed in the Project's profile. Thus the index() and show() functions are not defined except to redirect the user back to the Project. The attributes of a BookingDefault resemble those of a Booking, except that instead of a single date, it has a date range and selected weekdays to which it will apply, such as "Tuesdays and Thursdays, from September 1 to December 15, 2015."

The BookingDefaultRoleAssignments are the counterpart of BookingRoleAssignments except that they belong to a BookingDefault instead of a Booking and would be a regular weekly shift for the User assigned to them instead of a one-time assignment. This will also be explored further in the relationship between BookingDefaults and their Bookings.

Other Models: CellCarriers, Qualifications, Stafftypes, Usertypes

In Like the CRUD functions for LocationTypes, BookingTypes, and ProjectTypes, the CRUD functions for CellCarriers, Qualifications, Stafftypes, Usertypes are also relegated to administrator access and are not expected to require frequent updates. The screens for viewing and editing these models are compact, showing an editable list of all the models in the database with a button to save changes to each one, and a line to add a new model if needed.

Thus there is no need to define the create() or show() methods to view and edit individual models, or even the index() method since the edit() view is also the list of all the existing models.

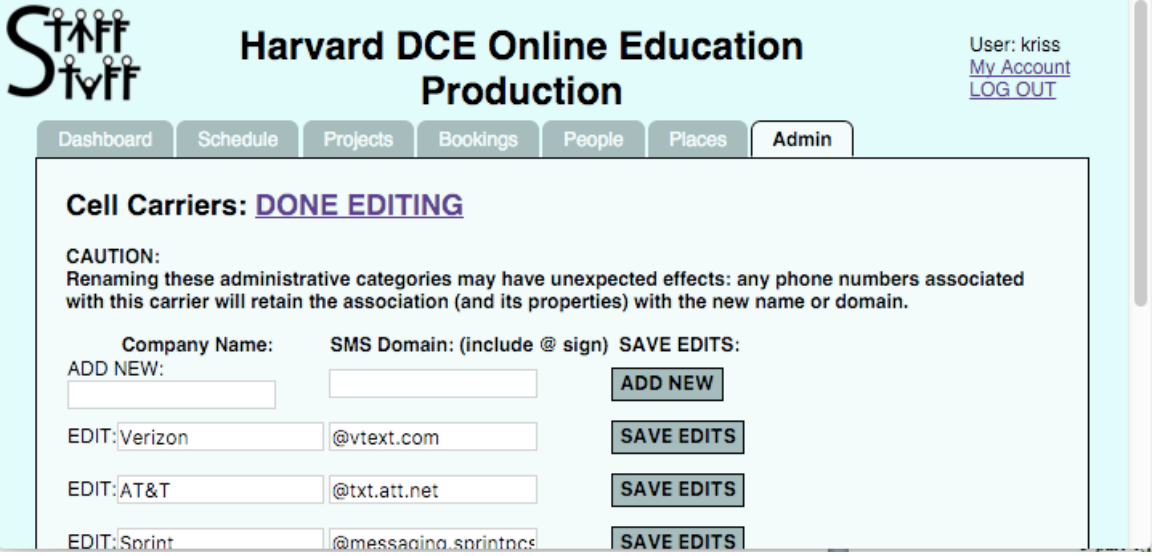

Figure 10 Admin view to list or edit cell phone carrier companies

Schedule Views and Quick Search Buttons

Now that I could fill a database with Bookings and all the other models that relate to them directly and indirectly, I was ready to search, filter, sort, and display this data to answer the questions in our user stories. The Schedule tab in the UI includes a "Power Search" form which allows the user full access to the main filteredSchedule() function in the ScheduleController should they want it. Based on this form input, the filteredSchedule() function builds a query, runs it, and passes the resulting data to a view which is also selected by the user in the form.

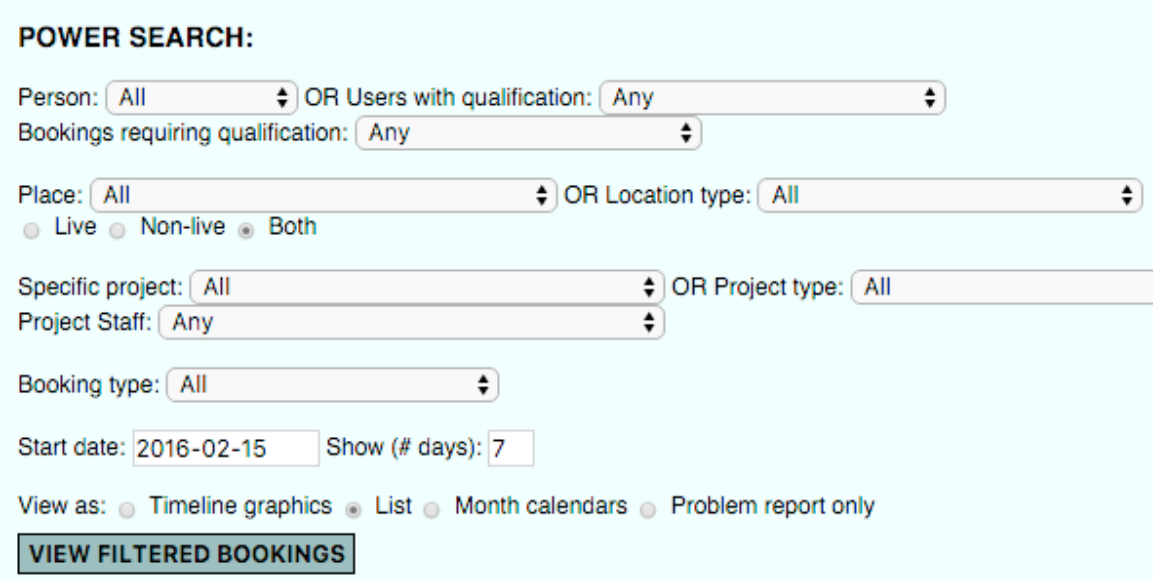

Figure 11 Power Search form

Each dropdown field on this form adds a clause to the query, as does the date range selected. Some of the fields offered to the user in this form, however, are not attributes of a Booking itself, but of models related to them, such as the User assigned to one of its Project's ProjectRoleAssignments. Laravel's Eloquent query builder allows these to be included as nested queries in that main query, rather than as explicit table joins or a subsequent filtering of retrieved data. Expressed in Eloquent form the clause for this example would be

```
$query = $query->whereHas('project',
         function ($query2) use($request) {
            $query2->whereHas('projectRoleAssignments',
            function ($query3) use($request){
               $query3->where('user_id',
                              $request['project_staff']);}
            );
         });
```

or read plainly in English, “Bookings where the Project has a ProjectRoleAssignment where the User assigned matches the one specified in the Request.”

The final row of radio buttons on the Power Search form select how the data will be displayed. The filteredSchedule() function passes this information to the Blade view along with the data, and the Blade code then includes the partial which will lay out the data in that way. Options include a list of the bookings sorted by date and time, a timeline graphic sorted by the location or people involved, or a calendar layout.

The list view is displayed in a table that is sortable by any column using the TableSorter JQuery plugin (Bach, 2007). The Blade view is coded so the referring function can send a flag to suppress certain columns from being displayed. This increases the ways in which this view partial can be used in pages. For example, when included in a location’s profile to display the upcoming bookings in that location, the “Place” column would be suppressed. When included in a Project’s information page to list all of its associated bookings, the “Project” column would be suppressed.

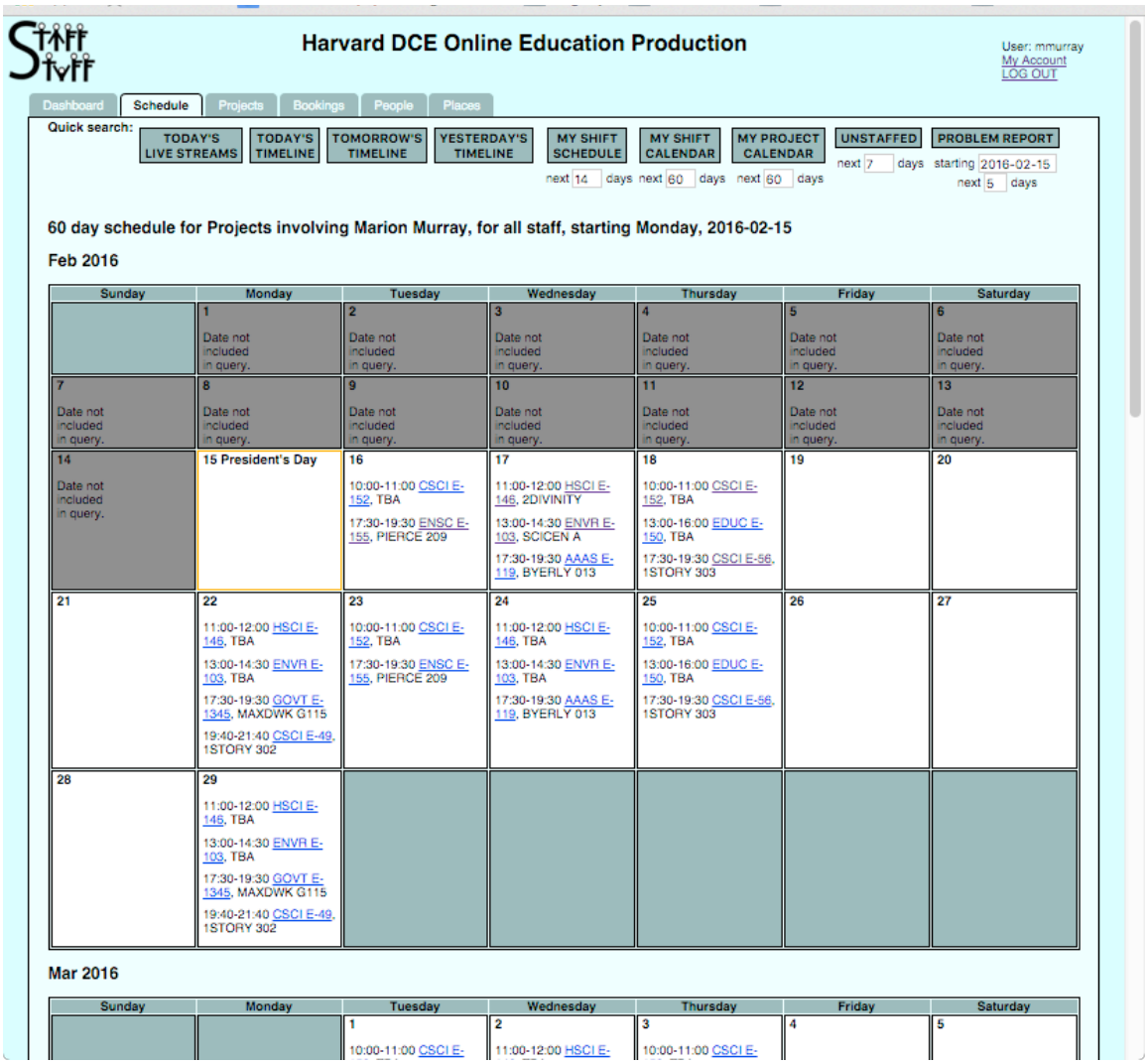

Figure 12 Calendar view: a Producer's "My Project Calendar" Quick Search

The calendar view is displayed simply in an HTML table which is styled to resemble a calendar.

The timeline view utilizes the Google Charts javascript library (Google, 2015). As a data visualization, several aspects of this timeline representation are problematic, such as the inability to choose a consistent Y axis scale for all the charts on one page (Figure 14). However, we decided that writing better timeline visualization code was not within

the scope of this project. Since this view is useful for some user stories, we decided to keep it as an option, and compensate for its deficiencies in other ways when possible.

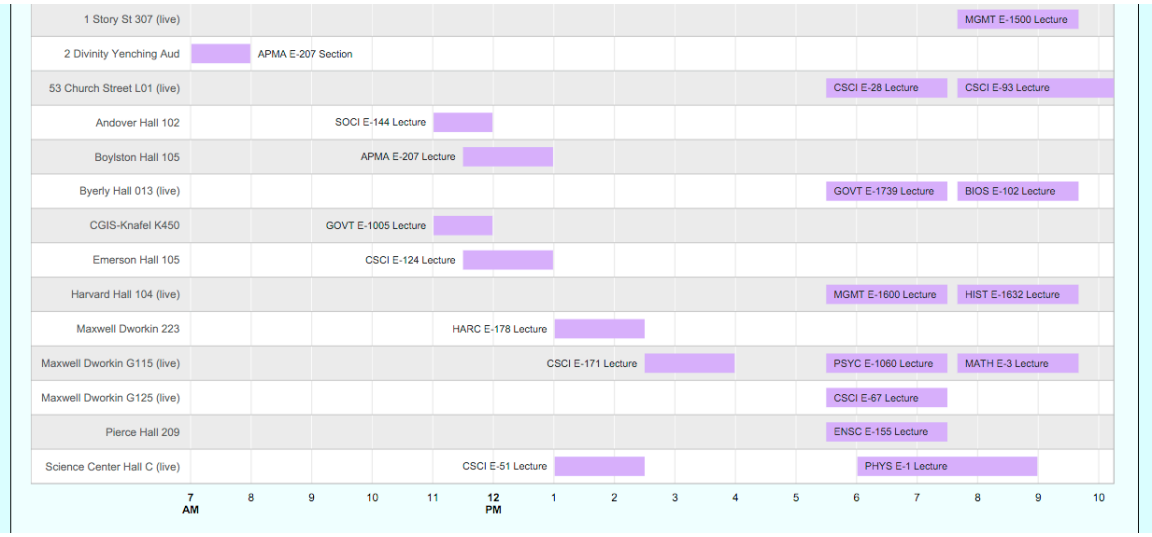

Figure 13 Timeline view: search results for Bookings on one date

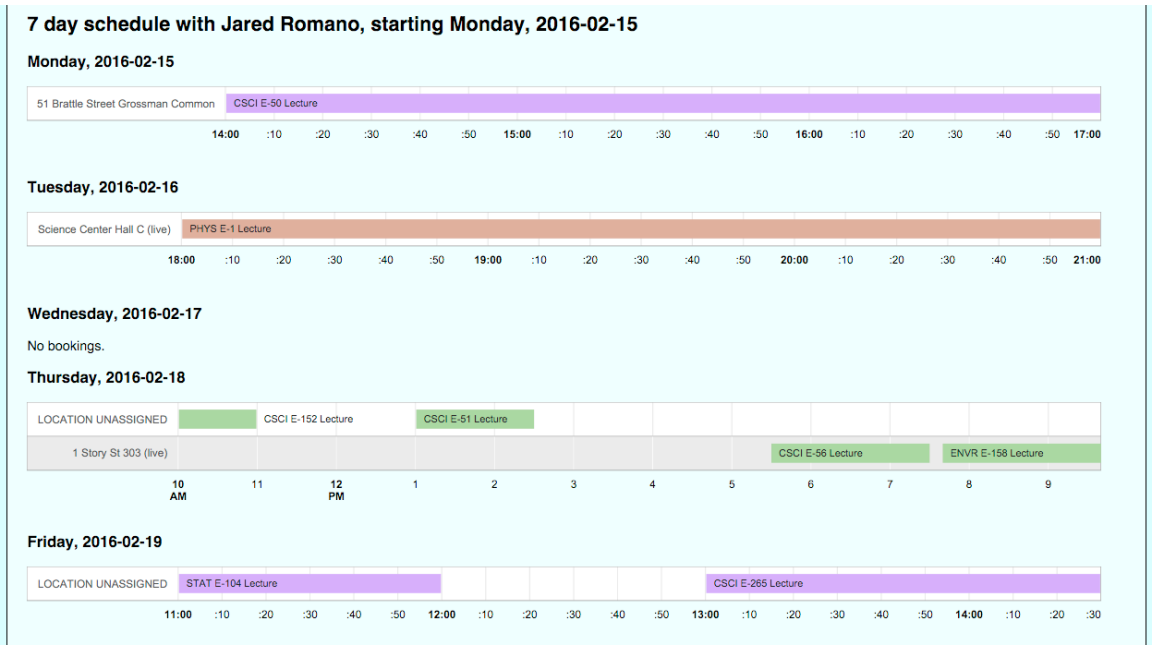

Figure 14 Timeline view: mismatched Y axis scale adversely affects visual comparison

One of the limitations of the Google Charts timeline is the inability to color code the bars differently within the same chart. For example, I would have chosen to color code canceled bookings as gray bars and active bookings as a brighter color. For certain views, it would be useful to use one color for live streaming bookings and a different color for those which are not. To compensate for this inability, I included more information in the text used as labels on the chart, such as including the text "(live)" next to the names of locations which have that ability, and the text "CANCELED: " before the label on a timeline bar for a canceled booking (Figure 13).

For most users, the "Power Search" form with every possible option available is more complicated than they wish to use. For these users, the filteredSchedule() function and the view partials which display its query results are still working for them, but in ways that are hidden. Every user's dashboard includes a set of "Quick Search" buttons which with one click take them to the information they most often need, such as "Today's Live Streams" or "My Shift Calendar." These simple buttons are actually hidden HTML forms which include the parameters for the filteredSchedule() function which will return the data and view which answers their question. The button "Today's Live Streams" is the submit button for a hidden form that requests bookings that have today's date, and have the attribute "livestream" set true, displayed as a sorted list. For "My Shift Calendar," the hidden form field for the person's ID is set to the user's own ID, the start date for the date range is today, the duration of the date range is 60 days unless otherwise specified, and the view is "calendar."

Because these Quick Search buttons are just small hidden HTML forms with the field values preset, they can be added or deleted from user views easily during the current beta release, based on user feedback we are receiving.

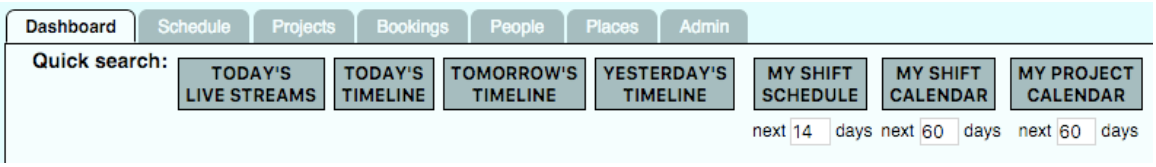

Figure 15 Example of Quick Search buttons on a dashboard

Problem Reports

Administrative users of the system are assumed to be the managers of the department who are using StaffStuff to organize their production scheduling, so their dashboard and views in the UI deal more with the big picture of the departmental logistics rather than taking a focused view of their personal booking assignments.

One of the challenges of running a complex department is identifying where the logistical problems are that need to be acted upon, and how urgent they are. Tools for this job were inadequate or completely lacking in the existing scheduling packages; while some of them would flag certain problems, none of them had a clean view of just problems to be solved, and none of them were able to clearly identify all of the types of problem an administrator needs to address.

In order to identify whether each booking has any problems to be flagged, it must be compared against any other bookings which are concurrent or within a minimum time interval of each other. I chose 30 minutes as adequate travel time between locations on

campus for this client; although it is not true for special circumstances such as field trips it will work for general scheduling. I defined a "problem" as being one of the following:

- A location with more than one concurrent booking: check whether both Bookings have the same location_id.
- A person with more than one concurrent booking or insufficient time between bookings: for each BookingRoleAssignment belonging to the first Booking, check if the user_id is also in any of the BookingRoleAssignments belonging to the other Booking. If so and the bookings are concurrent or overlapping, flag a problem. If they are not concurrent, but have less than the minimum amount of time between them and are not in the same location, flag a problem.
- A booking with staff positions defined which are not filled: null user_id in BookingRoleAssignment.
- A booking with no staff positions defined: no BookingRoleAssignments.
- A booking staffed by someone who is not qualified for the role as defined: a BookingRoleAssignment has a qualification_id which is not associated with its assigned User.
- A booking staffed differently than a booking in that location normally requires: compare the Booking's Location's recommended BookingRoleAssignments and their Qualifications to those that the Booking actually has.
- A booking with no location: null location_id.

- A booking in a location marked as inactive: check the Location's status attribute.
- A booking staffed by someone marked as inactive: check the status attribute of each User assigned to a BookingRoleAssignment.
- A booking marked as live streaming but booked in a location without that capability: check if the livestream attribute of the Booking matches that of the Booking's Location.

Later when I added the features which would facilitate the following calculations, I added them to the Problem Report:

- A person exceeding the maximum number of hours allowed to be worked in one week: for each User assigned to a BookingRoleAssignment, run the weekly_hours() function that is defined in the User model and flag totals higher than allowed for that User's Stafftype.
- A booking staffed by someone marked as unavailable to work at that time (see Availability feature for details).
- A booking in a location marked as unavailable at that time (see Availability feature for details).

For StaffStuff admin users viewing a schedule, or for any user who selects the "Problem Report" view in the Power Search form, these problems are identified and stated in red lettering for easy visibility.

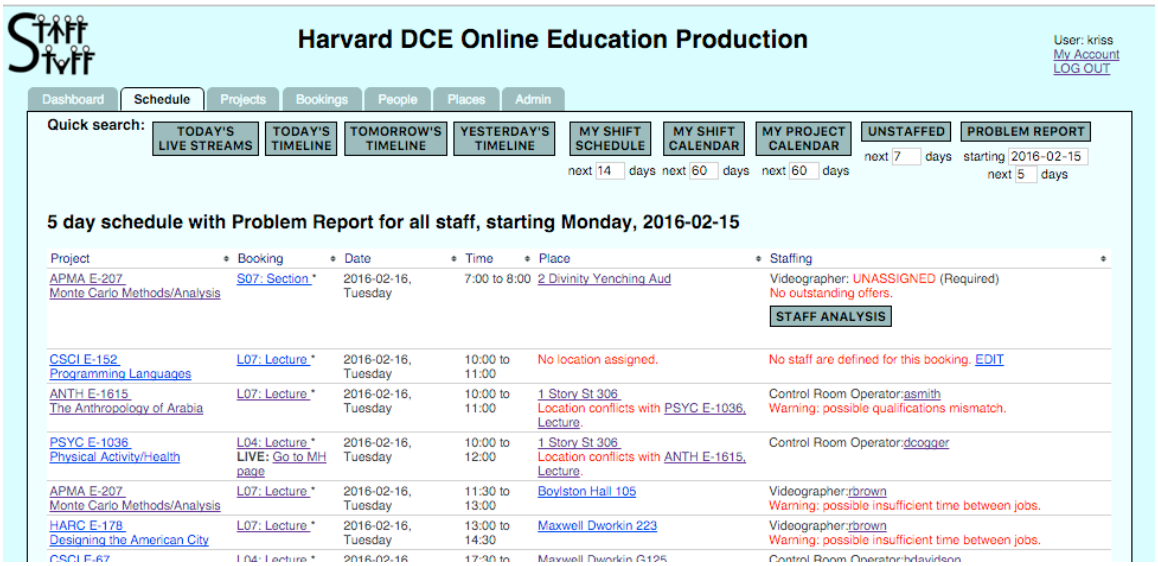

Figure 16 Problem Report

It might seem that some of these problems should not occur. For instance, if a booking has staffing defined which is not usual for that location, can we not assume that this is intentional? However, we cannot; if a class is relocated from a traditional classroom where a videographer would carry a camcorder and tripod to record it, to a classroom with a control room, the session is now staffed incorrectly because it needs a Control Room Operator instead of a Videographer. A booking might have had no staff defined when it was created because the location type was not yet known. A location or person might become inactive or unavailable after they were assigned to a booking. If the time or date of a booking is adjusted, this may cause double booked resources or for a person to be scheduled overtime.

The algorithm for making these comparisons is computationally intensive. If it is not to run an additional database query for each booking in order to find its temporally overlapping bookings for comparison, it must at least compare each booking to all the others in the given dataset to find those that overlap, and then if so, run the additional

comparisons between the two bookings to look for problems. This conflict analysis runs in quadratic time. Additionally, the calculation to check the number of hours worked by each person in that pay week requires additional database queries, as does the function to check the person's availability for the booking. The slow runtime of the function was not a problem while working with a test database with a small sample of projects and bookings. However, when scaled to the actual load of the client's production department, which can be 50 bookings per day and is projected to increase, it requires several minutes to run a Problem Report analysis for more than two or three days at a time.

In order to improve the runtime of this analysis, the results could be timestamped and stored in the database so that the function could retrieve the results of the latest analysis and then search only for possible changes since it had last been run. However, adapting to this algorithm would require restructuring large portions of the code. The comparisons between bookings currently happen during the sorting and laying out of the "Problem Report" view, since other views of the same schedule information also originate from the same function in the ScheduleController but do not require this processing. In order to save the results, the problem analysis logic would need to be moved back to the ScheduleController; as currently written, this would slow down the results for other views as the Problem Report analysis ran unnecessarily for any view. A better approach would be to split off the Problem Report function entirely from the general purpose filteredSchedule() function which powers the views that do not use problem analysis. Another option for a future release is to explore the techniques that would allow the page to load data in smaller chunks as the results of the analysis became available, rather than wait for the full analysis to be complete before loading the page.

To ameliorate the runtime problem until one of these potential solutions can be further explored, the Problem Report view was removed as a module included on the admin user's dashboard, so that it would not need to run every time the user returned to their "home" screen. The default number of days on the admin's Quick Search button for a problem report was reduced from 7 days to 3 days, and a field was added to select a start date other than today. This made it easier for an admin to run only a few days at a time to look for upcoming issues, without needing to navigate to the Schedule tab and use the more cumbersome Power Search form to run the next set of dates.

Setting Up a Semester: Data Import

Another of the shortcomings in most of the existing systems was the inability to upload or inject large data sets, either via an API or file upload. This presents a data entry problem not just as we set up the system for the first time but at the beginning of every semester, so this feature is high priority.

During our evaluation of the existing systems, our software developers dumped the Legacy System's database into CSV files of the data for all the classrooms, course instructors, and staff members. Each semester, we also receive a CSV of the results from a query run by the Registrar to get the data for most of the courses that we offer in a given semester, and when they will meet. Data for a few additional production projects had previously been stored in a Google Sheet which was also exported to CSV. This meant the simplest way to address our bulk data entry needs was to provide upload of CSV files. These functions are in the AdminController and are run from pages accessible in the Admin tab.

I restricted the ability to upload users and locations to not just admin users, but only to the user with ID number "1." The only time uploads of users and locations should be necessary is when a StaffStuff system is first installed and the administrator is migrating information from the previous system. The user setting it up will always be the original user "1," before additional users have been created. Restricting it so drastically allowed me to skip a number of checks on the data, such as the format or actual SQL injection attacks, which are necessary for features that will be needed by users who will not have any direct access to the database to correct mistakes, or might have malicious intentions.

For the data that will be uploaded at the start of each semester, however, I did include a number of checks on the input and detailed instructions on the upload page. This setup might be performed by any administrator on the system, several times each year. Uploads at the start of term might include lists of the courses to be produced, and lists of the booking defaults for lectures, sections, or other booking types. The instructions include the exact names of the columns and what order they should be in, and any file that does not match is rejected. If the CSV file passes this check, the data is read in and each field is checked for correct formatting. Required fields are checked for content.

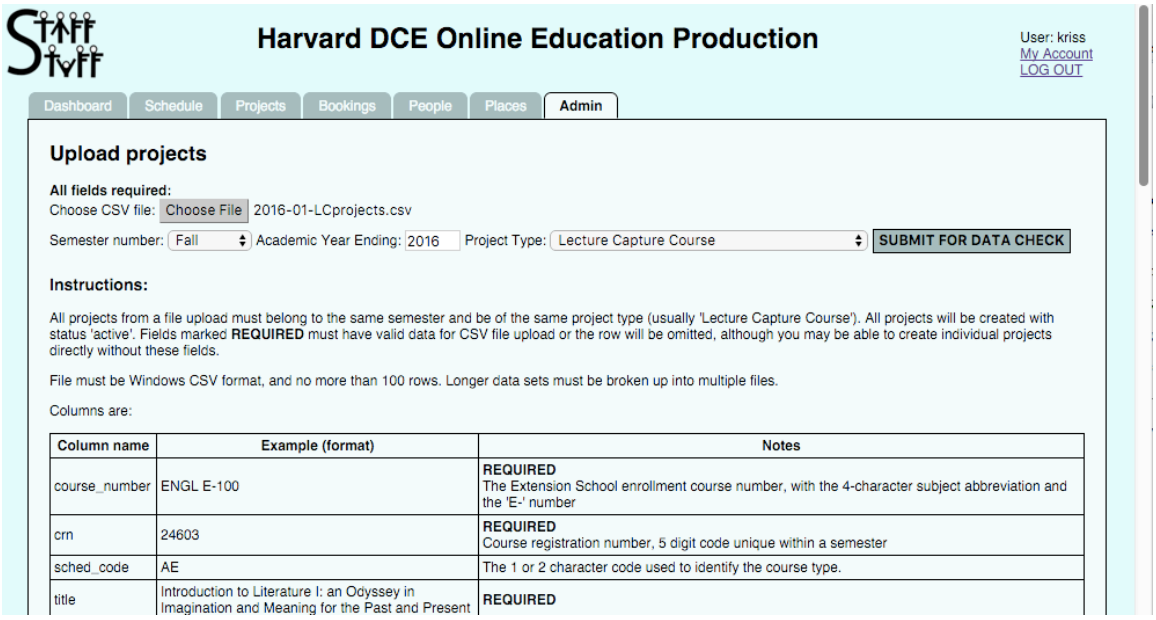

Figure 17 Upload screen for CSV files of Projects

For the upload of the semester's courses, the user selects what project type they will be, and the file to be uploaded. This original upload is saved to a temporary file, and the user is shown a table of the data with any problems flagged and how the system will handle them. In addition to the basic checks listed above, each row is checked for whether such a project already exists, by looking up whether there is already a project with the same semester, year, and CRN. Duplicate projects or rows with required fields that cannot be parsed will not be saved. The names of course instructors and the production lead are looked up in the StaffStuff users table and flagged if a match is not found or multiple matches are found. The user may at this point choose to cancel the upload or to save it. If saved, the system does one last check on the data as it is saved, to ensure that the uploaded file itself contains no duplicates. If not saved, the upload is canceled and the user may correct any problems in the data file before trying again.

Once the projects are uploaded, the user may upload CSV files of booking defaults for them, such as the meeting days, times and locations of weekly lectures. Each booking type must be in a separate file, i.e. one file of all the lecture defaults, one file of all the section defaults, and so on. The user again selects the semester and year to be uploaded, and the first field in the CSV file is the CRN of the course, so the project to which they belong can be identified. In a similar process to the project upload, the data is saved to a temporary file and displayed to the user with any problems or duplicates flagged after the checks are run, with options to either save or cancel the upload.

Setting Up a Semester: Generating and Updating Bookings Based on Defaults

The final step in defining the semester's parameters is for the admin user to define which dates are considered holidays, meaning that they should by default be skipped when generating bookings. This would include not only national holidays such as Labor Day, but also school breaks such as Spring Break. These dates are entered in the admin screens.

With the semester's Projects, BookingDefaults, and Holidays thus entered, the admin user may then with the click of a button generate a list of what the bookings for each project would be. For example, a course with a lecture default of Mondays from 10-11am, starting on August 31 and running through December 21, would generate a booking on every Monday during that range, except holidays. This process also checks for duplicates and will not create a booking where there is already a booking for that project which would overlap. As before, the user may review this data in the displayed table, and then choose to save the semester's bookings, or cancel the action. Because this

process checks for duplicates, if more courses or booking defaults are added to the CSV file later, it may be uploaded a second time and the semester's bookings generated a second time to fill in only the added data.

Bookings do have a booking_default_id field which relates them to the BookingDefault that generated them, if there is one. Bookings can also be created one by one for a project without involving a BookingDefault, in which case this field remains null. However, if the Booking belongs to a BookingDefault, this allows the user to update its information by updating the BookingDefault. For example, if a meeting location changes to a different classroom for the remainder of the semester, the user can simply update the BookingDefault. The function which updates the BookingDefault also checks for any Bookings it has which are not yet complete, and updates their location as well. Bookings whose end time is past are not changed.

Day Notes and the Relationship of Bookings to their BookingDefaults

The relationship between a BookingDefault and its Bookings, however, is more complex than it initially appeared. One of the reasons to have individual Bookings generated is so that individual Bookings can be changed, such as when one particular date meets in a different room or has different staffing. The question then is, when the BookingDefault is updated, should it also update that Booking which no longer matches the default values? I initially decided that it should not, and implemented this by nulling the Booking's booking_default_id field to completely decouple it from its default any time the Booking information was changed. However, depending what had changed, we may or may not want it to be affected by changing the default. For example, if one date

has a substitute staffer covering the usual assignee's vacation day, this should not prevent the booking from updating if the lecture moves to a different classroom for the rest of the term. If the location had changed for just one day though, that could indicate a special event which should not be reverted to the regular classroom if the lecture were to move.

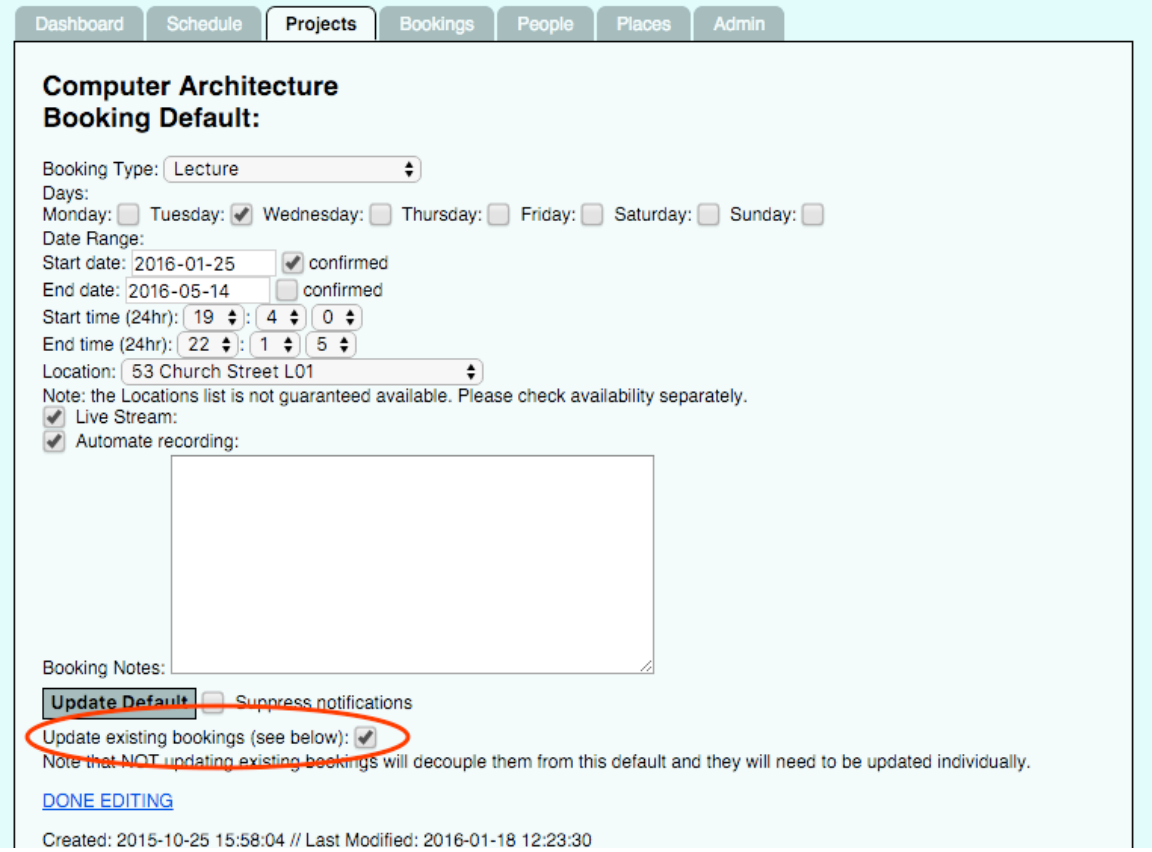

Figure 18 BookingDefault edit screen, with option to update the Bookings

Unable to devise an algorithm which could make such judgements accurately for all scenarios, I determined that we must rely on the user to make this decision. I kept the behavior of decoupling the BookingDefault from the Booking whenever the Booking was changed, but added a warning when a user changes a BookingDefault which lists all the Bookings of that BookingType which were not affected, so the user could note them and change them individually if desired.

I reconsidered this decision upon implementing the timeline view of the schedule. I had initially envisioned a view which would show visually where the day's actual schedule differed from the default schedule by color coding the bars representing bookings, or by overlaying the day's bookings with the default. However, upon learning of the limitations of the Google Charts tool, I realized I would need a different way to indicate clearly where the day's schedule was not the default, since I could not color code or overlay the graphics, or even control the axes enough to place two charts side by side for effective comparison. Technical Support workers who answer student questions refer to this information to explain anomalies in the video-on-demand listings for online classes, such as when a video is not listed for a certain date when there normally would be one. With visualization ruled out, I implemented a function to list the differences as text "Day Notes" next to each day's timeline.

The algorithm for these Day Notes first checks each BookingDefault that ought to have a Booking that day to see if there is indeed a Booking, and lists it if there is not, such as "No regular STAT E-101 lecture at 17:30." It then iterates through all the day's Bookings which are not associated with a BookingDefault, such as "Special HIST E-200 Film Screening at 19:00." The problem with the decoupled BookingDefaults and Bookings became clear, as they would be listed confusingly as two separate deviations from the default schedule: "No regular BIOS E-30 Lecture at 11:00" followed further down in the list by "Special BIOS E-30 Lecture at 11:00," simply because it was being held in a different room or was staffed by a different videographer. This was not correct, and I reasoned that ultimately the problem was not entirely in the algorithm generating

the Day Notes. It was that the relationship between the Booking and the BookingDefault objects was still not correctly modeling the real world.

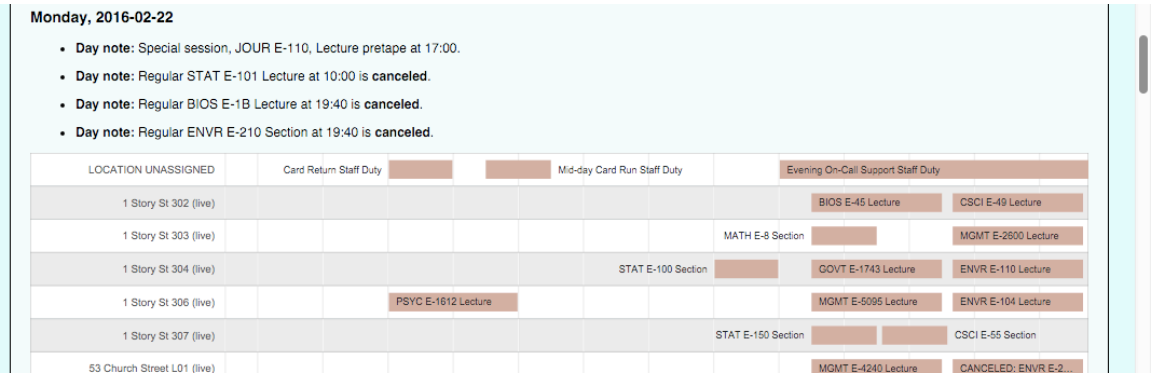

Figure 19 Day Notes in the Timeline view

I rewrote the function for updating a Booking that belongs to a BookingDefault so that they did not decouple for any change other than the actual date of the Booking, which would indicate an extreme deviation from the default schedule and ought to be listed as a special event in Day Notes. This would be the case, for example, if a lecture were to be postponed to a different date. I then rewrote the function for changing a BookingDefault so that when a field is changed, it only updates its Bookings where that field is an exact match for the old value of the BookingDefault, and alerts the user to cases where the fields did not match and were therefore not changed. Finally, I rewrote the Day Notes function so that it compares each Booking of the day to its BookingDefault if it has one, and announces any differences as a single statement, such as “BIOS E-30 at 11:00 located in Sever Hall 101 instead of Harvard Hall 104.”

Resource Availability

The availability of a person to work a shift can be thought of as the inverse of the blocks of time that the person is unavailable to work. I determined that if I represented this concept as unavailability blocks, this enabled me to use many of the existing models and functions to store and analyze this information. Each block of time that a person is unavailable is a Booking, which allows us to detect whether a person is available to work using the same functions that detect conflicts between two production Bookings. Similarly, availability can be set for other resources, such as a location, by assigning that location to an unavailability Booking.

I created a ProjectType and BookingType for system functions and a special Project to which Unavailability Bookings would belong. I adjusted the functions which display lists for users of Projects, Bookings, and search results that would include these models, to exclude these and any other system functions that I might devise.

Typically when I would request that staff give me their general availability for a semester, these messages would be phrased in terms most like a BookingDefault, a weekly template. One might say “I cannot work any Mondays, and I can only work after noon on Tuesdays, but I am available all day on Wednesday through Friday.” This seemed like a natural way to represent them in the database as well, so I began by converting the times and days submitted by the user in their availability form into a set of BookingDefaults, and from there generated the Bookings, again using the existing functions. These availability Bookings then conflict or not with the production Bookings that require staff.

I allowed the user to enter up to four blocks of unavailability per day: the earliest start time they will work, the latest end time until which they will work, and up to two additional blackout periods during the day. Thus, on a day when the user can come in by 10:00, has another commitment from 12:00 to 16:00, but can then work the evening until 22:00, there would be three BookingDefaults created to represent the blocks of time the user cannot work: 00:00 to 10:00, 12:00 to 16:00, and 22:00 to 23:59. I decided to allow the user to enter time blocks that overlapped, and combine them in the function that processes the form into the fewest number of Unavailability blocks for the day before creating the BookingDefaults.

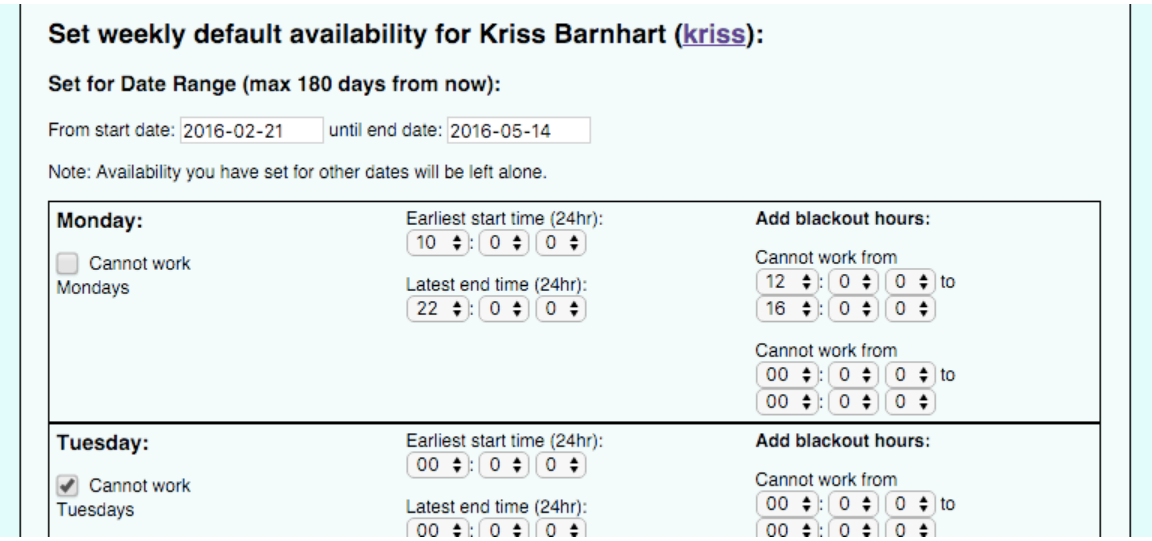

Figure 20 Personal availability form (partial)

The user also enters the date range to which this schedule applies. When the user enters a new availability schedule, it completely overwrites any other availability that was set for that date range. All previously existing Unavailability Bookings for that user are deleted for those dates, and the BookingDefaults to which they belong are adjusted to not overlap the newly created BookingDefaults.

To adjust the BookingDefaults' date ranges, I had to account for five scenarios. Let us consider a user who had previously set their availability for February 5-20. First, the newly created Unavailability BookingDefaults might not overlap an existing Unavailability BookingDefault at all, such as if the new range were for February 1-3, or February 25-28. Second, the new set might overlap the beginning of the old set, such as if the new range were February 1-10. Third, it might overlap the end of the old range, such as February 15-25. Fourth, it might replace a range in the middle, such as February 8-12. Finally, it might completely overwrite the old range, such as February 1-28.

Once the function has identified whether the newly submitted availability overlaps with one or more existing Unavailability BookingDefaults and which scenario has been created for each, it adjusts the existing BookingDefaults. If they are to be completely overwritten, they are deleted. If only the beginning is to be overwritten, the start date is adjusted, or if the end is to be overwritten, the end date is adjusted. If the new data falls in the middle, the old BookingDefault has its end date adjusted and then a duplicate is made with start and end dates to cover the period that would be after the date range of the new availability. As noted, any associated Unavailability Bookings that are no longer in the date range of their BookingDefault are deleted. In the last scenario, Bookings must be generated for the BookingDefault which was created to cover the portion of the old date range which falls after the range of the new availability.

In order for the availability analysis to be useful when searching for potential booking staff, the availability for each user must be kept current. To encourage frequent updates by the users, I placed links to view and to update one's availability on the LHT user's dashboard, as well as within their account profile page. The view is shown using

the calendar partial, and is translated from the Booking times back into more human-friendly language such as “Unavailable before 10:00.”

Notifications

One of the most time consuming tasks of managing a department with a large number of staff is communicating schedules and changes to each person in a timely way. While the schedule of LHT staffers is clear on the dashboard when they log in, they may not log in to check if they are in the habit of going to the same jobs each week, so changes also must be communicated directly. If the change affects a job on the current or next day, these changes should be communicated immediately such as via text message and email. If the change will not affect any immediate jobs, it can be queued and all changes sent as an email digest each night, to prevent multiple disruptive messages enumerating each minor change to a future booking as they are made.

Text messaging is accomplished by sending email to the user’s cell phone number at a designated domain for their carrier service provider. For example, a user whose cell number is 617-987-6543 and whose carrier is AT&T can be sent a text message on their phone by emailing 6179876543@mms.att.net. Storing the domain for each carrier company is the main purpose of the CellCarrier model, and these can be added or updated in the admin screens should they change.

The Laravel framework recommends using an emailer service to send messages (Otwell, 2015), rather than holding the next page load while the system actually sends them itself. It is preconfigured for the MailGun service, which at the relatively small scale required by StaffStuff is a free service, so I set up the recommended account.

I then implemented the notification features, starting with a notify() function in the User model which sends a notification to the user, either immediately by text and email or by writing it to the Notifications table in the database to be compiled into a digest later, depending on the parameters defined. I created notifyStaff() functions in the Booking and Project models which will send a notification to all of their own staff by iterating through their ProjectRoleAssignments or BookingRoleAssignments, respectively, and calling notify() for each user. With those functions in place, it was a matter of going through each Controller's CRUD functions, determining which actions merit a notification to which users, composing the subject and body of the messages that would be sent, and calling the appropriate function to send them.

For example, when a Booking is canceled, meaning the status attribute is changed from "active" to "canceled," the update() function in the BookingsController now not only saves the new information for that Booking, it also calls the functions to send a notification to the staff for that Booking, as well as the production staff for the Project to which the Booking belongs. If the Booking was scheduled for the current or following day, it includes the parameter which flags it for an urgent notification, and the email and a text message will be sent immediately to Booking staff instead of saved for later. If the notification is not urgent, it is saved to the notifications table to be sent out at midnight as a digest of all the changes that happened during the day which affect the recipient.

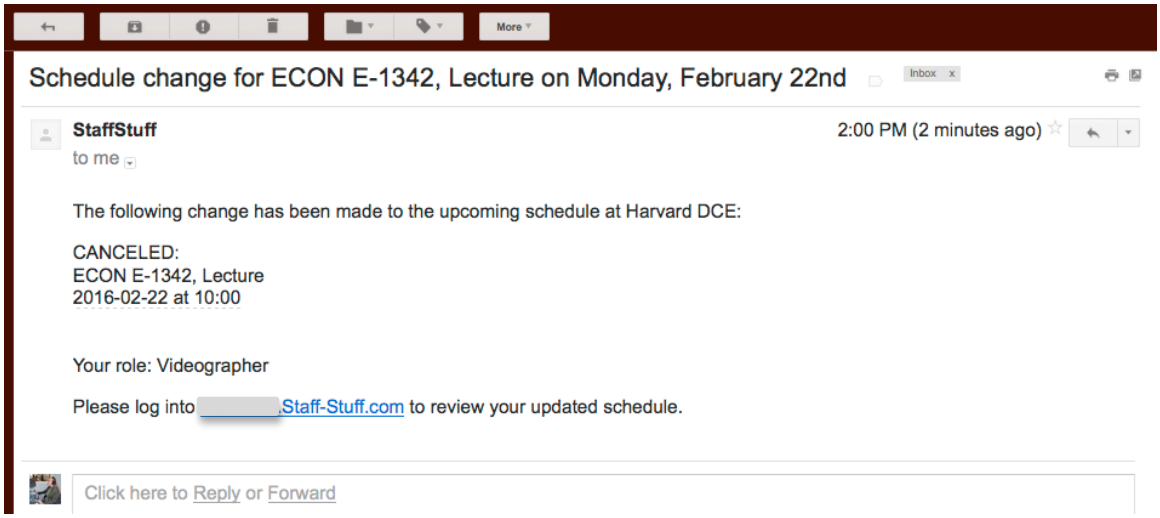

Figure 21 Notification email for an urgent schedule change

The Laravel framework provides a Kernel model which handles scheduled jobs such as sending daily emails, so the schedule() function in that model calls the function which composes the digests and sends the emails. A Crontab command must be set up on the server to call the schedule() function regularly to check if there are any functions that ought to be run. This is achieved using Artisan, the command line interface for Laravel.

In addition to the daily digest of schedule changes, users may subscribe to daily or weekly emails of their upcoming booking schedule. The function which handles these emails retrieves from the prefs table the profiles which have flagged a subscription to the daily or weekly schedule, as appropriate. For each, it composes the email body using the same functions and view partials as the website does to display the Users' personal schedules, but with the HTML internal links stripped out, and calls the User's notify() function to send the email.

Because each individual action which changes a Booking, Project, BookingRoleAssignment, BookingDefault, or BookingDefaultRoleAssignment, may

trigger notification messages to a number of users, I found when testing the system that I was not comfortable with having no control over whether these notifications would be sent. For example, I might be making a very minor change, or a change that would be reverted or changed again soon. I added a "Suppress Notifications" checkbox to each form whose action might trigger notifications, and updated the functions to check for this option before sending or queuing any texts or emails.

To round out the notifications features, I added buttons to the user's profile page which trigger the sending of a test email or a test SMS message. Thus a user can go to their profile at any time and check if they are properly receiving their notifications from the system.

Job Requests and Job Offers

The JobRequest model is simply a collection of foreign keys and one Boolean. It associates a BookingRoleAssignment, i.e. the object representing the job, with a User who has either requested or been offered the job. If the admin_id field holds a value (the ID of an admin level User), then that admin has offered the job to that User. If the admin_id field is null, then the User has requested the job but not been approved and assigned to it. The Boolean flag tracks whether the offer of the job has been declined by the user, rather than delete the record upon declining and have no record that the job was ever offered to that person.

This simple model allows the implementation of several features. First, from the LHT or producer's dashboard, they may follow a link to a list of available jobs in the next 30 days, meaning BookingRoleAssignments which have no User assigned, for which

they have the Qualification and have listed themselves as available at the time of the Booking. Each listing has a button to "Request" the job. The button is a hidden form, the processing of which creates and saves a JobRequest model, and notifies the administrators of the request. On the administrator's dashboard is a list of all the current requests, each with a button to approve it. If the administrator approves, the User is assigned to the BookingRoleAssignment, a courtesy notice is sent to any other users who had requested or been offered that assignment, and the relevant JobRequests are deleted from the database.

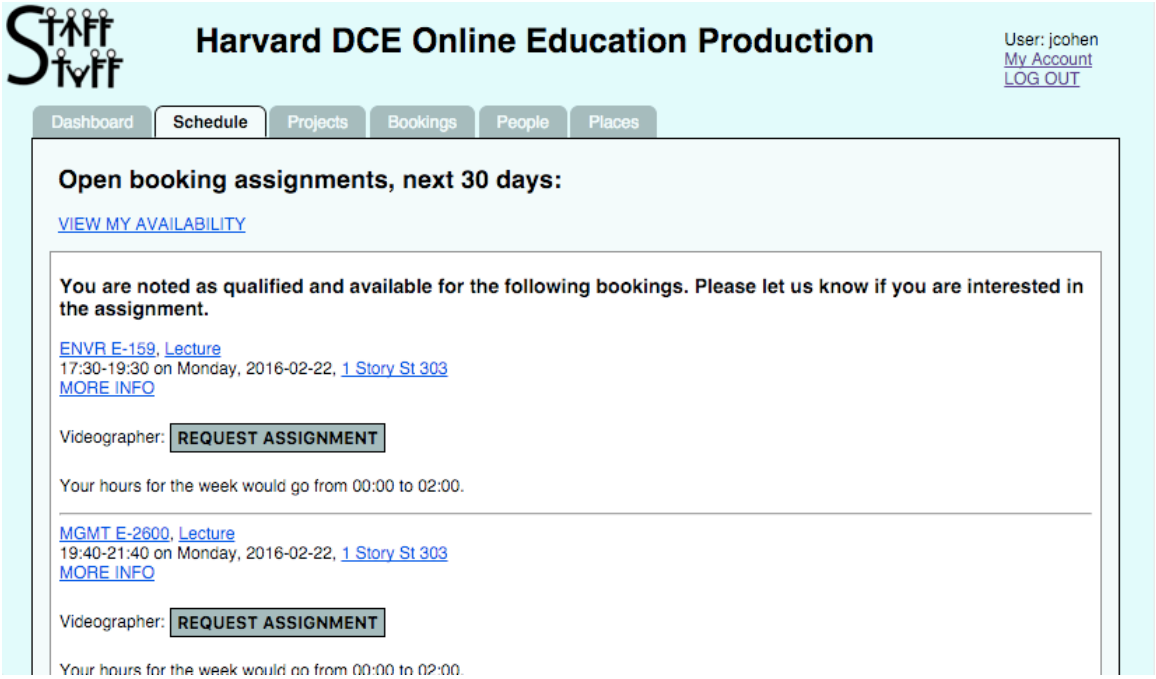

Figure 22 Job openings list, with option to request assignment

Approached from the perspective of the administrator, instead of simply assigning a User to a BookingRoleAssignment, they may select to send an offer to one or more users. The processing of this form creates and saves a JobRequest for each User, and sends them a notification by email, by text message, or both, depending on that User's

profile preferences as defined in the User's Pref object. When these users log in, any offers extended to them are displayed on their dashboard, with buttons to accept or decline each offer.

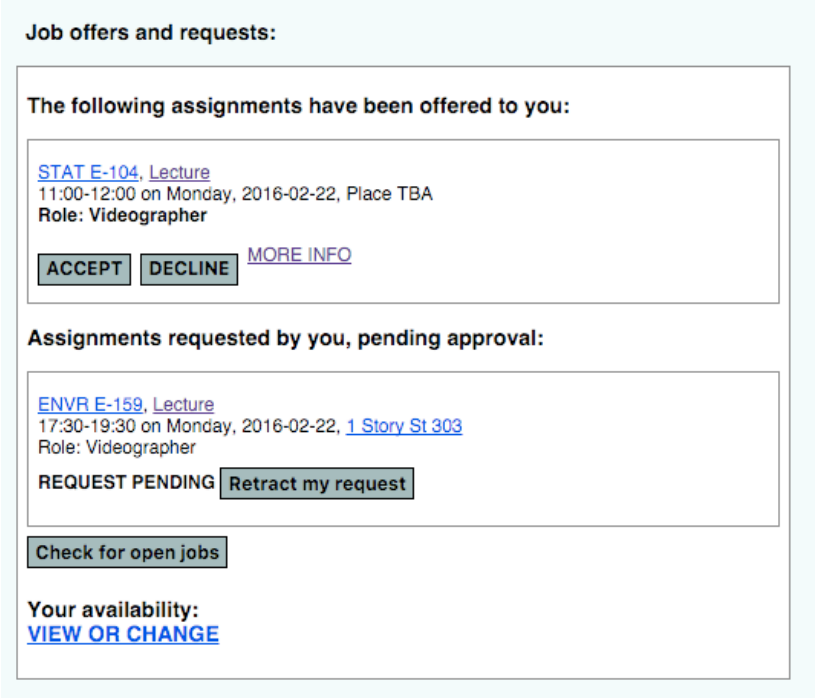

Figure 23 Job Requests and Offers Module displayed on user's dashboard

The first user to click "Accept" is assigned to the BookingRoleAssignment and sent a confirmation. The administrators are notified, others who had been offered the job are sent a courtesy notice that it has been taken, and the JobRequests are deleted from the database. Any other outstanding offers or requests for the assigned user which would conflict with the newly accepted BookingRoleAssignment are automatically declined.

If a user selects "Decline," this is noted in the Boolean field of the JobRequest rather than deleting the JobRequest, so the user will not be offered the job again in this round. Users who decline are automatically marked as unavailable during the Booking, unless they are already scheduled to work a different Booking at that time. This prevents

them from needing to decline repeatedly if the job were to come back up for assignment, such as if an assigned staffer later finds themselves unable to work the shift. They may go to their profile and set themselves available again during that time if they wish, although they are cautioned that this could result in their being offered the same declined assignment again.

Calculation of Hours Worked

One of the ways in which no existing third party product would be sufficient is in the ability to calculate paid hours for staff according to the complex rules of this particular production department, and use that information to warn the administrator when a given user would be in overtime.

Since most of the staff who work on production bookings are categorized as LHT, less than half time employees, they are legally not allowed to regularly work over 17 hours in a given week. A very time consuming aspect of staff planning was keeping a rough tally of how many hours each staffer would be working in a week and only offering jobs to those staffers who would not go over 17 hours if they were to accept them. The complexity of this task is compounded because different types of LHT staff have their hours calculated differently.

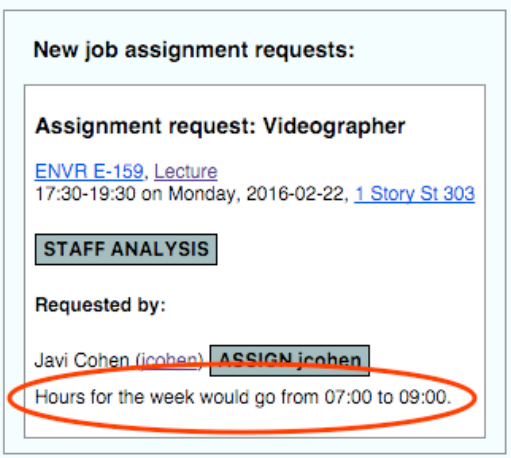

Figure 24 Example of paid hours calculation displayed in an admin user's view

LHT videographers who provide their own video camcorder, tripod, and accessories rather than using departmentally owned equipment are paid at a higher rate which includes the gear rental fee. The time calculation algorithm and pay rate are grandfathered into policy from the time that this type of employee was a freelance hire who billed as an independent contractor. They are paid for "tape rolling" time, meaning only the time that the event is in session and they are actively recording. However, there is a minimum shift of 90 minutes, which takes effect any time there is a gap of greater than three hours between shoots. For example, if the videographer shoots a class from 1:00 to 2:00 and then an event from 4:00 to 7:00 on the same day, that would be a total of four paid hours. However, if after the 1:00 to 2:00 class, the event took place from 6:00 to 9:00, there would be a four hour gap between the shoots and the minimum shift rate would apply to the one hour class; the paid total would be 4.5 hours.

Paid hours for LHT production staff who do not provide their own equipment are treated like regular office staff: time from clocking in to set up for a shift, until clocking out after shutting down or returning equipment to storage. In general, staff are allowed to

clock in up to 30 minutes before the start of a shoot, to allow time to set up the equipment and run necessary checks. They are allowed to clock out up to 15 minutes after the end of a shoot, to allow time to shut down equipment or return gear to storage. These allowances are flexible if the type of production requires it, but are considered typical. If the gap between events is greater than the 45 minutes assumed to be the total of the shutdown time of the first event and the setup time for the next, the staffer is required to take an unpaid meal break during this time which divides the series of events into two blocks of paid time. For example, an LHT working a class that meets 1:00 to 2:00 is paid for 1.75 hours to allow setting up and shutting down equipment, assuming a clock in time of 12:30 and clock out of 2:15. If that 1:00 to 2:00 class were followed by another at 2:30 to 3:30, the gap between is only 30 minutes and no break is assumed; the clock in time is 12:30 and the clock out is at 3:45, for paid time of 3.25 hours. However, if the classes were 1:00 to 2:00 and 4:00 to 5:00, they would be calculated as two separate shifts with setup and shutdown time, and unpaid time between: the first shift from 12:30 to 2:15 and the next from 3:30 to 5:15, for a total of 3.5 paid hours that day.

Thus, the paid hours for a given job may vary depending on the rest of the schedule of the user who works it. A videographer with no other jobs that day would be paid a minimum 1.5 hour shift for a one hour class, while another who will be recording another class shortly afterward would only be paid for the one hour. A booking in a control room which is already set up by an LHT operator and has another booking shortly before or after could eliminate some amount of paid setup or shutdown time that would be required were it booked into a different control room with a different operator.

In order to help the administrator assign staff to bookings efficiently and without unintentionally assigning more than 17 paid hours to an LHT employee, the calculation of weekly hours for each possible assignee must be done twice: once without the potential booking included and once with it. This shows the administrator which staff would not go over 17 hours and how much the job would be worth in their schedule.

After I had written the functions to calculate paid hours with or without a potential booking included, I added this information to the various views where an administrator might be assigning a shift, such as the job request approval form (Figure 24). I also designed an administrative screen for use when approving submitted time sheets for pay, which simply lists all staff and the number of paid hours calculated for a given work week. While the actual number of hours reported might vary slightly due to unexpected delays or production complications, having an accurate idea of expected hours is very helpful when the departmental manager approving payroll is not necessarily the same administrator who is assigning the shifts.

Staff Analysis Table

With all the components in place, I was able design the culmination of the StaffStuff system: the Staff Analysis table. Wherever an administrative user sees an unstaffed job listed in the system, it is accompanied by a “Staff Analysis” button which submits a hidden form for that BookingRoleAssignment. The view which is returned includes a table with each staff member who is qualified for the job categorized. LHT staff and full time staff are listed in separate columns.

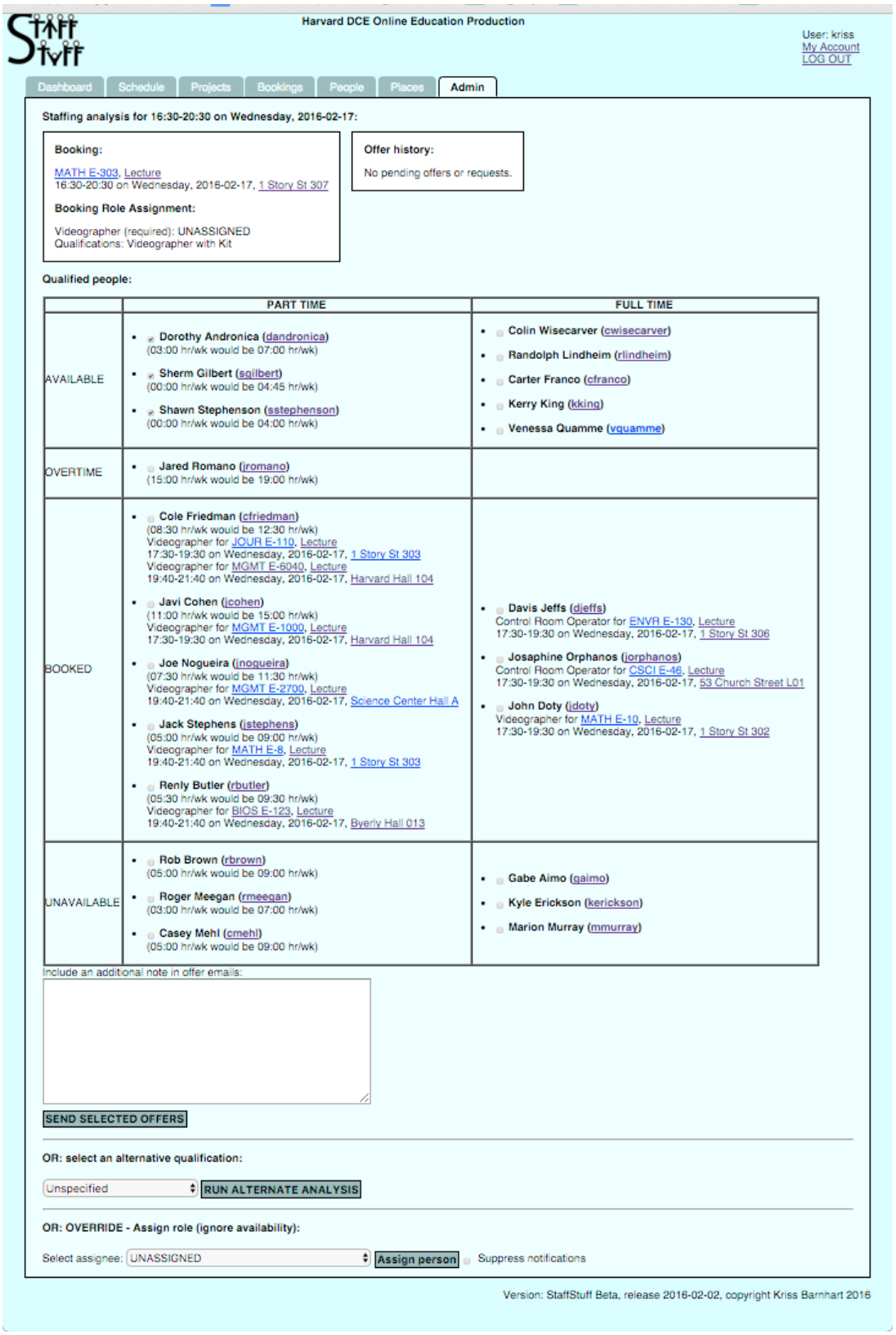

Figure 25 Staff Analysis Table

The rows are:

- users who are available and would not go over allowed hours
- users who are available, but would be over allowed hours
- users who are busy on another booking at that time
- users who are marked unavailable at the time

Next to each user, information is listed to help the administrator decide to whom the job should be offered, including the total hours with and without the assignment, if there might be insufficient travel time from another assignment, if there is a conflicting booking what that booking is, and whether the user has already requested or been offered or declined the job.

If any job offers or requests for the assignment are outstanding or have been declined, these are listed at the top of the page, along with the timestamp of when the offer or request was made. This information is useful in determining how likely it might be that the job will be taken by someone who already has an offer, and hence whether offers should be sent to additional users.

The whole table is also an HTML form, with check boxes next to each listed user. By default, all the users listed in the upper left cell are selected: the LHT staff who are available and would not be over allowed hours if they took the assignment. The administrator may select and deselect any users in the table, and optionally add some custom text to the notification message that will be composed. When the form is submitted, the system creates and saves the JobRequests and sends out the job offer notification to those users who are selected.

At the bottom of the Staff Analysis page are two additional options for the administrator. There is a form to run the analysis again but with a different Qualification. For example, an administrator who sees that there are no control room operators available who are qualified to work in a given room, might want to see an analysis of staff who are qualified to work in a different control room, or who are trainees. A second form allows the administrator to simply assign a user directly to the position, overriding whether they are qualified or marked available.

New User Orientation

While the existing staff records which would be uploaded to the users table typically did include a cell phone number and email address, almost none had the cell carrier information. Also, testing of the notifications system revealed that some email accounts, including many at the client's own institution, did not receive messages from the StaffStuff system, presumably due to spam filtering. These considerations as well as the need to force users to change their passwords when first logging in, led to a required new user orientation.

When a user logs in, the system checks if they have one of the system default passwords, and if so then instead of the regular dashboard the user is shown a welcome message and invited to begin the first of five orientation steps.

1. Review your personal information, especially your preferred email address, cell phone number, and cell carrier company.

2. Send yourself a test email message. If you do not receive it, consider editing your profile to use a different email account (return to step 1), or adjusting your spam filters.

3. Send yourself a test text message. If you do not receive it, check whether your information, including the cell carrier company is correct (return to Step 1).

4. Select your preferences for email subscriptions and how to receive notifications of job offers.

5. Set a new password.

The user is then invited to either go to the form where they can set their availability to work, or to return to the regular system dashboard.

On the Admin tab, there is an option which when run will secure all unclaimed accounts, defined as accounts which still have the system's default password. It changes the password on these accounts to a random string. If the user subsequently wishes to log in and set up the account, an admin user can navigate to the user's profile and reset the password to a known string.

Chapter 7 Deployment and Evaluation

The StaffStuff system was deployed to the intended server at a custom domain on January 2, 2016, and the client's real data was uploaded to populate the users and locations tables. Projects and BookingDefaults for the upcoming Spring semester were also loaded using the CSV upload feature, and administrative users began to use the system when the offices were opened on Monday, January 4. On Friday of that week, the rest of the full time production staff were invited to activate their accounts, and given a 45 minute training session to practice using the features they would require most and ask questions about the system. Selected members of the software development department, who need easy access to the production department schedule, were oriented and trained the following Monday, January 11. Student help desk staff with read-only accounts were trained on Thursday, January 14. Finally, the largest group of users, the LHT staff were oriented and trained on Tuesday, January 19. Full departmental production load began on the first day of the semester, January 25.

During these first three weeks of this Beta deployment, several bug fixes were required although none were severe and most could wait until a weekend coding session and Sunday evening release. The most significant included a logic flaw when the user accepted an offered assignment, which although the actual assignment was made correctly generated conflicting notification messages, and a logic flaw in combining the user's submitted availability times into simplified blocks. However, most bugs only

affected the few administrative users, and so could be temporarily worked around until fixed.

Based on early user feedback, I improved navigation around the UI by adding more direct links to other views, such as making course numbers clickable to jump to the project profile page. I also added some information to common views which I had previously thought to keep very simple, such as showing the title of a project rather than only the course number in the list view of a schedule, so it could be compared directly to the Matterhorn automated recording system's schedule which displays only the titles.

Producers requested a Quick Search button for "Today's Live Streams" in a list format, for use when they were on duty to monitor the feeds, so I added it. I also adjusted the text on the Quick Link buttons to clarify the difference between "My Shift Calendar," which is Bookings at which the user has a BookingRoleAssignment and must therefore be there in person to work a shift, and "My Project Calendar" which is the schedule of all Bookings for all of the Projects which the user is assigned to, such as all the recorded class meetings for all the courses where the user is assigned as Production Lead even though the producer does not necessarily need to attend every lecture in person.

When a bug in Matterhorn caused its listings page of the day's live streams to cease listing the links to them, I quickly added the links to the StaffStuff schedule list view; they are constructed from a common domain and file path plus the year, semester, and CRN of the course, so no new data fields were needed in StaffStuff to construct them. For the first full week of classes, production staff logged into StaffStuff to get the links to their video feeds instead of using Matterhorn's own pages.

The producers requested an easier way to view what time slots were available for the duration of the semester in classrooms which had live streaming ability, a common question when online courses are trying to book discussion section meetings. The "Power Search" for filtering a schedule was not suitable for this. I added a feature to the Places tab that when given a date range, returns a timeline view for each day of the week, displaying for each live classroom the booking defaults that overlap that date range, including any availability blackouts. The user can then click through on any room to view the full calendar of bookings in that room, to check for any special events that would interfere with a weekly repeating booking in a desired slot.

Some users requested that we keep the old color coded spreadsheet of our project information in addition to StaffStuff, and although I initially dismissed these requests as mere resistance to change, after only a few days I changed my mind and built the "Legacy View" page. This view places all active Projects and BookingDefaults into an HTML table in the style of the previous shared spreadsheet, including the color coding to which the staff is accustomed. I built this not because of the pressure from the users, but because I found that as a departmental manager I needed such a table to serve as an overview of the upcoming semester. Although this format had been a terrible layout for individual staff members to find their assigned bookings, it evolved the way it did over the past fifteen years because it was a very good layout to see patterns and problems in the overall semester's weekly schedule.

Some user feedback was helpful in identifying where I had not gone far enough in following the business logic of how the department runs. For example, one LHT user pointed out that when I extended him multiple job offers, the system allowed him to

accept all of them even if this would mean that he worked significantly over 17 hours. The system did not catch the problem as I made each of the offers, because any particular one of those offers would not have brought his total over 17. I agreed that this was not the ideal behavior, and I adjusted the algorithm such that when users accept an offer that would put them in overtime, instead of assigning them to the BookingRoleAssignment, it instead reverts the offer to a request requiring an additional admin user approval.

Comments from the LHT users indicate that they do find the StaffStuff system easy to navigate and that their work schedules are clearly communicated, unlike they had been under the old practices. The main complaints are from users whose email accounts or cell phones do not receive the automated StaffStuff notifications.

For the first few weeks of the term, users contacted me frequently after receiving notifications or the daily digest of changes, to make sure that they were reading and interpreting them correctly. They almost always were, and these contacts have tapered off as users became familiar with the system. Some producers have needed a bit of extra training to correctly interpret the timeline views of a schedule, and some extra practice to efficiently find the answers to their common questions, but have expressed that overall the new system has been worth learning and is much easier once they get used to it.

For administrative users, of which I am the main interested party, the measure of success is not only in time saved and errors prevented, but also in the reduction or elimination of analog and manual supplemental practices to keep track of the schedule, which do not scale. These include the large whiteboard grid of weekdays and staff with their availability and schedule written in, paper notes that can be mislaid, or simply attempting to remember details. The office whiteboard still displays the grid from Fall

term, because I did not need to draw one for Spring in order to figure out what staff are able to work at a given time or where someone is. Very few paper notes exist, and those only as a temporary measure when someone did not have a networked device with them.

A case could be made for designing the system with more general use in mind, by not accommodating the highly specific business logic such as the esoteric methods of calculating paid hours, using terminology such as "section" that is particular to a specific set of academic institutions, or carrying over terminology and concepts from the depreciated Legacy System. A design which did not consider these details would be less likely to require significant revisions or become obsolete as the client department grows and changes over the coming years, and might have marketability beyond this particular client. However, as discovered when evaluating the numerous existing products, a more general system which does not track the complicated details unique to this client would still require a great deal of manual labor and cross referencing of sources to reflect the real world logistics, which have these details whether we model them or not. The elimination of these secondary processes is indicative of the ability of StaffStuff to meet the client's needs completely and with high accuracy, in one system.

Chapter 8 Summary and Conclusion

When considering some of the more convoluted aspects of the business logic we modeled, advisor Larry Bouthillier likened the process to the old adage of "paving a cow path," which although it often has a negative connotation was an apt metaphor. One reason to pave a cow path is if there are historical landmarks scattered along the area that the road must traverse, and it is impractical to destroy or relocate them. The builders could simply place the new highway in a different area without inconvenient obstacles, where it could be straight and efficient. But this would not serve the needs of the local population who would still be without a road. At some point, a road must be built which takes the most efficient path while accommodating the history, and so must a customized system such as StaffStuff be built when there are traditions and organizational structures that do not fit into a generic set of parameters.

When designing the user interface, I focused on user stories; my goals were efficient, intuitive navigation for the majority of users, and simplicity which would eliminate the need for extensive training. For the former, I tested my designs by evaluating how many steps were required for a user to answer their most common questions in their user story. For the latter, I consulted with the colleagues who would be the system's primary users to make sure they could accomplish basic activities such as creating a booking or looking up information, with minimal coaching.

The underlying data model for StaffStuff is not significantly different from the concepts used by existing systems: production projects which have bookings at particular times and locations, users who staff them, and so forth. Even the functionality for automatic communication with users and the offering and accepting of work shifts is not unique to this product. The custom improvements lie in efficient methods for setting up and making adjustments to a large new batch of projects several times each year, and in the analysis functions which utilize algorithms based on the unique business logic of the client. Only with this precise fit to the client's needs can supplemental practices, such as tracking details in separate documents when they don't fit the system, be eliminated.

Further development of the system could provide an API for Opencast Matterhorn or similar systems to access a schedule and the metadata for automated recordings. Other possibilities include improvements to the user interface such as custom Javascript timeline graphics instead of the flawed Google graphics timeline plug-in.

After the success of the StaffStuff system in the Spring 2016 beta release, the client's intention is to continue use through Summer into the next academic year, and we are currently in discussions regarding potential purchasing or licensing of the system for longer term use.

References

Agile Alliance and Institut Agile (2013). User Stories. *Agile Alliance Guide*. http://guide.agilealliance.org/guide/user-stories.html, retrieved February 20, 2016.

Bach, C. (2007). Tablesorter Flexible Client-side Table Sorting. http://tablesorter.com/docs/, retrieved February 20, 2016.

FarmersWIFE (2015). FarmersWIFE. http://www.farmerswife.com/, retrieved August 18, 2015.

Flynsarmy (2015, February 5), Creating a Basic ToDo Application in Laravel 5 – Part 1. https://www.flynsarmy.com/2015/02/creating-a-basic-todo-application-in-laravel-5-part-1/, retrieved August 18, 2015.

Google, Inc. (2015), Timelines | Charts, Google Developers, https://developers.google.com/chart/interactive/docs/gallery/timeline, retrieved October 26, 2015.

Laravel.io (2014, July 8). Laravel Vs CodeIgniter a Difficult Choice. *Laravel.io Forum*. http://laravel.io/forum/07-08-2014-laravel-vs-codeigniter-a-difficult-choice, retrieved August 18, 2015.

Nerdmom (2012, August 2). Comparing Laravel, CodeIgniter, & CakePHP. https://nerdmom.wordpress.com/2012/08/02/comparing-laravel-codeigniter-cakephp/, retrieved August 18, 2015.

Opencast (2016). Opencast Matterhorn. http://www.opencast.org/matterhorn, retrieved March 5, 2016.

Otwell, T. (2015). Laravel 5.1 Documentation. https://laravel.com/docs/5.1/, retrieved February 20, 2016.

Planday (2015). Employee Scheduling Software. http://planday.com/, retrieved September 14, 2015.

ScheduALL (2015). ScheduALL Operational Management Module. http://www.scheduall.com/modules.html, retrieved August 18, 2015.

WebCheckOut, Inc. (2015). WebCheckOut Personnel Scheduling. http://www.webcheckout.net/personnelschedule.html, retrieved August 18, 2015.

Web Revisions (2015). Best PHP Framework for 2015.
http://webrevisions.com/tutorials/php-framework-the-best-php-framework-for-2013/#.Vq_10DYrI_U, retrieved August 18, 2015.

Xytech Systems (2015). Xytech MediaPulse.
http://www.xytechsystems.com/products/mediapulse/, retrieved August 18, 2015.

Appendix 1: User Story Questions and Functions

When opening the StaffStuff application, the user needs to perform a function or answer a question based on the data. When designing the user interface, I listed the following as likely questions and functions for each type of user, and focused my design efforts on making fast, intuitive paths to the answers.

Producer users

1. What classes am I producing next term? What is their default weekly schedule? What is their actual booking schedule?
2. What class videos am I supposed to publish today?
3. I need to book a live section. What slots are available?
4. I need to book a non-live section. When can we staff one? Can we staff this specific time slot request?
5. I need to book a live special review. What slots are available?
6. I need to book a non-live special review. Can we staff it? Can we staff this specific time slot request?
7. Who was the videographer for this class yesterday? Was it the usual person?
8. I am on night support. What is the schedule? What is different today? What live streams should I check? What staff are fill-ins?
9. I am on night support. What classes have special needs today?

10. I am on night support. I need to contact the videographer that is filming a specific class right now.

LHT (part time staff) users

1. What is my default schedule?
2. What is my schedule today/this week? Is anything different from usual?
3. Is there anything special about this shoot or location I should know?
4. Am I live streaming or just recording this shoot?
5. When is spring break? Will classes meet on a particular holiday? When is semester start/end?
6. My availability has changed, and I need to update it.
7. My personal info or contact info has changed and I need to update it.
8. I can't work one of my shifts, and need to request a substitute.
9. I want to pick up some extra work. I need to find shifts I can do and indicate interest in them.
10. I was offered a shift and I need to accept or decline it.
11. I'm not sure I am getting my notification emails or texts. I need to check that and try to fix it.

Student Technical Support (read-only) users

1. A student complains that there is no live stream or video recording posted. Did/will class meet?
2. What live streams should I see running tonight? Is anything different today?

3. I don't see anything on this stream that I am monitoring. Is the class meeting? Has the class ended?
4. Does this specific course generally live stream this semester?
5. When/where is this special session going to meet? Will it live stream?

Management (admin) users

1. What is today's schedule of bookings? Where are my staff?
2. What is today's schedule of live streams? I need to compare the list of live streams with what is scheduled in the automated recording system to make sure it is correct.
3. Are there any conflicts or other problems with today's bookings or those coming up soon, which I need to address immediately?
    a. Location conflicts or no location assigned.
    b. Staffing conflicts or no staffing assigned. Assigned staff no longer available to work.
    c. Incorrect staffing for the location or type of booking.
    d. Unqualified staffing with no training scheduled.
4. I need to create a new project, and define what staffing it will need.
5. I need to assign staff to roles on that project. Which staff have the skills for the role? Who has worked on similar projects before? Which producers have worked with a particular course instructor?
6. I need to create a new booking for a project, and define what staff it will need.
7. I need to create a series of recurring bookings for a project, skipping holidays.
8. I need to change one of the bookings in a series without changing all of them.

9. I need to make a change to all of the remaining bookings in a series on this project, without affecting the ones which are complete, so the integrity of historical data is maintained.
10. What are the upcoming bookings which need to have staff assigned? What roles need to be filled on each booking? What skills are needed in those roles?
11. What people with the correct skill set are available for those bookings? Which of them would be in overtime, and by how much, if they were assigned the job? Are any of them particularly interested in the assignment? Are any staff already in the booking's location due to another booking before or after? If so are they available during this booking, and if not what are they doing?
    a. I need to approve an interested person for the role, and notify them.
    b. I need to efficiently offer the job to one or more qualified people.
    c. I need to see how many of my offers have had responses, and what they are.
    d. I need to skip the offering process and just assign someone to the job, and notify them.
    e. There are no qualified people available and I need to find someone with similar skills who can be trained.
12. I need to check the information on a particular booking for a particular project. Is an upcoming special request scheduled correctly? Who staffed a given booking?
13. I need to make a change to a project's information, and notify the affected staff.
14. I need to make a change to a future booking, and notify the affected staff.
15. I need to make a change to one of today's bookings and immediately notify the affected staff by text message.

16. I need to make a minor or temporary change to a project or booking without sending automated notifications.
17. A member of the staff will be absent. I need to find what bookings they were supposed to do, remove them from the assignments, and find alternate staff to fill in.
18. I need to find a time and place which I will be able to staff for a requested booking within certain parameters, such as live streaming capability or a limited time frame.
19. I need to update or deactivate a user account.
20. I need to create or activate a user account for a new hire or client. I need to orient new users to the system and have them set their password.
21. I need to reset a user's password.
22. I need to check whether the automatic notifications are working for this user.
23. I need to update a user's availability to work on one or more dates.
24. I need to know if someone's updated availability conflicts with any shifts they are assigned to work.
25. I need to check how many hours an employee worked or will work in a given week.
26. I need to load in a large amount of data for an upcoming semester:
    a. I need to create a project for each course in the semester and assign the instructors and producers to each project.
    b. I need to enter the information for the lecture and section video recording default schedule for each project.
    c. I need to create the individual bookings from the defaults, define the staffing needs, and assign staff to the bookings.

System administrator

1. I need to enter a large number of system users, with or without login access.
2. I need to enter a large number of booking locations.
3. I need to enter a large amount of historical data: projects and bookings from previous semesters, and who staffed them.

Appendix 2: Full QA Checklist

Before each release with major changes to the code, the following Quality Assessment routine was executed in addition to testing the newly implemented features, to ensure that no major bugs had been introduced to previously working functions.

- Set up main DB using CSV uploads for users, locations, projects, and booking defaults. Generate semester in bulk.
- Search for a user
- Create a user
    - Try to put in bad information
- Edit a user - info, status
    - Try to put in bad information
- Edit a user's preferences
- Edit a user's qualifications
- Reset a password
- Send a test email
- Send a test SMS
- Set user's general availability
    - Try to put in bad information
- Set a vacation week availability

- Change a single date availability
- Change availability for a period that overlaps two or more sets of dates
- Create conflict between booking and availability
    - check that notices are sent
- View the user's schedule for accuracy
    - view LHT dashboard
- Search for a place
- Create a place
    - Try to put in bad information
- Edit a place
    - Try to put in bad information
- Change default role qualifications for a place
- Set the place's availability
    - Try to put in bad information
- Set a vacation week availability
- Change a single date availability
    - Try to put in bad information
- Change availability for a period that overlaps two or more sets of dates
- View live room default availability
- Search for a project
- Create a project
    - Try to put in bad information
- Add staff to project:

    - add role
    - add person to role
        - check that notices are sent
    - change person
        - check that notices are sent
    - Edit project info
        - Try to put in bad information
        - check that notices are sent
- Create booking defaults, with one that will affect tomorrow morning
    - Try to put in bad information
- Generate bookings
- Create single booking for TOMORROW MORNING
    - Try to put in bad information
    - check that notices are sent
- Edit booking default
    - Try to put in bad information
    - check that it propagates
    - check that notices are sent
- Change booking-from-default date
    - Try to put in bad information
    - check that it decouples
    - check that notices are sent
- Add staff to booking default

    - add role
    - add person
    - change person
    - delete role with person in it
        - check that it propagates
        - check that notices are sent

- Add role to single booking, with no user assigned yet
    - check that notices are sent

NOTE: at this point you need to log in as different types of users

- (as LHT #1,2,8) Request the booking
    - check that notices are sent
- (as admin) Offer the booking to at least 5 other people - LHT #3,4,5,6,7
    - check that notices are sent
- retract offer #7
    - check that notices are sent
- (as LHT #8) Retract your request for the booking
    - check that notices are sent
- (as LHT #3) Decline offered booking
    - check that notices are sent and availability is updated
- (as LHT #4) Change availability to conflict with booking
    - check that offer is declined
- (as LHT #1) Change availability to conflict with booking
    - check that request is retracted

- (as LHT #5) Accept offered booking
    - check that notices are sent and remaining requests (#6,#2) are handled
- (as admin) Delete person from the booking role
    - check that notices are sent
- (as LHT #4 and #5) Request the booking
- (as admin) Offer the booking to someone else (LHT #2)
- Grant request LHT #4
    - check that notices are sent and other requests are handled (#2,#5)
- Delete person from the booking role
- Offer the booking to someone (LHT #3)
- (as LHT #1) Request the booking
- (as admin) Assign role directly to LHT #5 at booking.edit
    - check that notices are sent and other requests are handled (#1,3)
- Cancel a staffed booking.
    - check that notices are sent and requests are handled
- Un-cancel the booking.
    - check that notices are sent, staff are deleted but offers are re-sent
- Cancel a staffed booking.
- Set the staff as unavailable or booked to something else.
- Un-cancel the booking.
    - check that staff are deleted and no offers are re-sent
- Re-assign roles to the un-canceled booking and make LHT requests
- Delete the booking.

    - check that notices are sent and requests are handled
- Change the videographer for just one of the bookings-from-default, like a fill-in.
- Cancel the project
    - check that bookings are canceled and all default and fill-in staff are notified
- Delete remaining bookings from each default with button that appears
- Delete a project
    - check that bookings are canceled and all default and fill-in staff are notified
- View schedule with filters, try different searches.
    - Test all Quick Search links
- View as timeline, calendar, list, problem report.
- Log in as producer and check dash and schedule views.
- Log in as read-only and check dash and views.
- Create problems! Do these show up in report?
    - Double book a person
    - Double book a room
    - No person assigned to booking
    - No room assigned to booking
    - Inactive room assigned to booking
    - Live class assigned to non-live room
    - Mismatch between role qualifications and person qualifications

- Mismatch between room's recommendations and booking roles or qualifications
- Insufficient interval for person
    - unless in same location
- Staff is over 17 hours