– Independent Work Report Spring, 2016 –

Tiger**UHR**:

A Management System for Undergraduate Resources

Lance Goodridge

Advisor: Jérémie Lumbroso

Abstract:

Several universities are beginning to employ undergraduates to help meet the demand caused by climbing enrollment rates. However, these student hires are typically not integrated into existing HR systems, which results in self-organized, and often poorly designed management setups. In this paper, we introduce TigerUHR, *an application for administrating undergraduate resources. Our system priorities user-friendliness, and aims to streamline the hiring and registration process by removing data redundancy and intelligently inferring information wherever possible. The app also provides powerful management and scouting tools for position administrators.*

Contents

1 Introduction

1.1 Motivation

Princeton's Computer Science department is growing faster than it can handle:

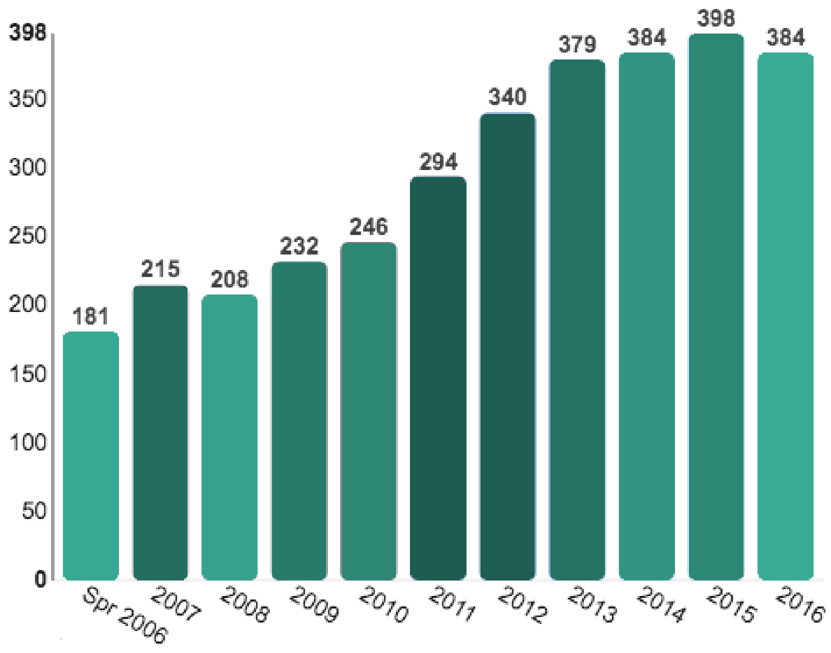


Figure 1: Final Spring Semester COS 126 Enrollment Numbers by Year
Darker color indicates higher enrollment difference from previous year

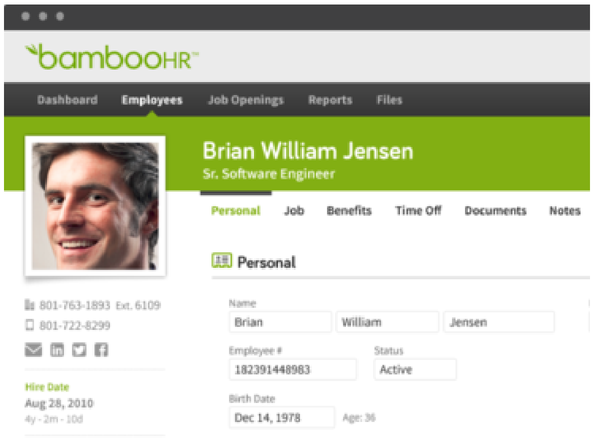
In Figure 1, we see the enrollment rates for the department's introductory course, COS 126: as you can see, the number of students enrolled has more than doubled over the last ten years, with nearly 400 students enrolled at the time of this paper. The department's other courses are also growing, with another 400 students enrolled in its intermediate courses: COS 217 and COS 226 [7]. Supplying the proper amount of attention and assistance to these numbers of students is infeasible with the department's current faculty size. Our university and others are realizing that the only way to have enough manpower to handle the climbing enrollment rates is to hire from the pool of undergraduates.

A scalable, intuitive system for managing the student resources is also needed; a poor HR system will only result in more chaos as the number of hires begin to rise. Currently, undergraduate resource management at Princeton is performed by hand across several Google Forms. This requires the maintenance of redundant information, is difficult to navigate, and does not perform adequately at scale. As the course enrollment numbers continue to grow, and correspondingly, the number of undergraduate hires, administrating student resources will itself become an unnecessarily difficult and complex task without a proper system in place.

1.2 Related Work

There are currently no applications that specifically aid the hiring and management of student resources. There are general HR systems, such as BambooHR [1], shown in Figure 2, that provide standard management tools for employee administrators. However, these apps provide much more functionality than is required, such as management of job benefits and tax information, and cannot leverage information provided by the university, leading to unnecessarily large registration forms.

There are also systems that make the work performed by undergraduate hires more efficient: such as codePost [2], which streamlines the grading process for multiple users, or the Lab TA Queue [6], which provides an organized scheduling mechanism for undergrad TAs to service their clients. However, these systems still require qualified people to be selected and hired for the position. This is the gap that TigerUHR aims to fulfill.



Brian William Jensen
Sr. Software Engineer

801-763-1893 Ext. 6109
801-722-8299

Hire Date
Aug 28, 2010
4y - 2m - 10d

Name: Brian | William | Jensen
Employee #: 182391448983
Status: Active
Birth Date: Dec 14, 1978 Age: 36

Figure 2: Bamboo HR

1.3 Goal

The department needs a system that consolidates the hiring and management of all undergraduate positions under a single application, and provides an easy-to-use interface for doing so. The goal of my project is to implement that system: a webapp that will collect the relevant information for all Computer Science undergraduate resources, and streamline the application and hiring process by providing a clean and intuitive UI.

2 Approach

Our app was designed with ease-of-use as the primary focus. In order to make the app as pleasant as possible for resources, administrators, and future developers, we defined the following to be guiding design philosophies:

2.1 Intelligent registration and data collection

Lengthy registration sequences are often notable deterrents for a product. The resource registration process should require as little information as possible:

Since we have access to data published by the University, the application should intelligently infer as much information as possible using those sources. In particular, objective information about a student, such as their name, class year, email address, and so on, can be collected from public websites such as Tigerbook [9] given their netid. A student should only have to enter subjective information, such as how many hours they're willing to work each week, and their approximate schedule.

Ideally, we would be able to collect their grades automatically as well. However, we do not have the authority required to query grades. Getting proper permissions to do so would require lengthy negotiations with the registrar, or having course staff manually input grades. The former is undesirable, given the scope of this independent project, and the latter violates the premise of having "intelligent", automated data collection in the first place.

Instead, we concluded that having students upload their transcripts, and parsing grade information from those was an adequate solution. While parsing the transcript, we can ensure that we receive up-to-date and correct information, and it also removes the risk that we might not have the grades for a particular student. Therefore, a transcript should be the only other data required during registration.

2.2 Painless application and hiring process

The job application and hiring processes should also be as simple as possible. Students should be able to choose which positions to be considered for at registration, and be automatically applied for those jobs. Hiring managers can then see who is interested in the position, and either reject them, extend an offer right away, or defer and request additional information from desirable candidates. In the last case, upon providing the additional information, be it in the form of an interview, an online form, or some other medium, the hiring manager should then have a second chance at either extending an offer, or rejecting the student. Upon receiving an offer, the resource can either accept or reject the invitation for employment.

It is critical to make the process as smooth as possible, as a student who feels burdened by the application process likely won't apply to many positions, or worse, decide not to use the app at all. Thus, special attention must be given to ensure that applying feels as natural and stressless as possible.

2.3 Powerful and intuitive management tools

Similarly, a hiring manager that feels encumbered by the management interface, or thinks the app lacks the necessary tools for the job, is also unlikely to use the site. Therefore, hiring managers should be able to view and sort candidates by attributes important to their specific position. Interviewers and student assistants should also be able to see necessary information about the candidates, but with sensitive details, such as grades, abstracted away.

There should also be a system for evaluating resources. When grading responses to an online form, or doing performance reviews at the end of a semester, the hiring managers will have to perform evaluations for many students at once. There should be a built-in system for administrators to this easily, efficiently, and collaboratively.

2.4 RESTful API

An intuitive API goes a long way when integrating with other apps or automating certain tasks. Having an accessible API means that other potential applications could use and benefit from our data, provided they have authentication to do so, of course. REST [8] is a popular design scheme for web APIs, and is both widely known, and very easy to interact with using `http`. We also felt it made the most sense given how easily the app lends itself to object-oriented construction.

2.5 Extensible Code Design

Finally, we intend to extend this app to integrate with other applications, and anticipate needing to add new features quickly and often. Maintaining a code layout that is "Open for extension, and closed for modification", will help ensure the feature adding process is safe and painless for present and future developers.

3 Implementation

3.1 Overview

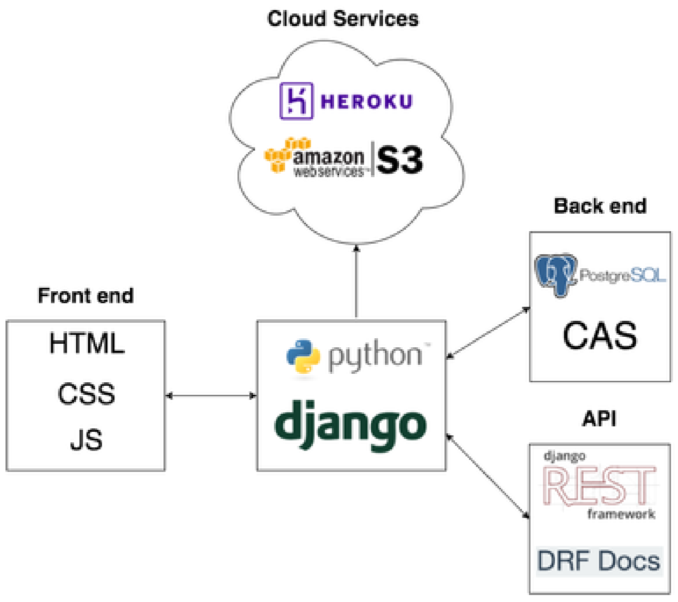


Figure 3: High level architecture of TigerUHR

The website was built using the Django Web Framework, and is currently hosted on Heroku. The backend is primarily written in Python, using Django, and the front end is written with standard web development languages: HTML, CSS, and Javascript. CAS is used for authentication, and PostgreSQL is used for database management. The app uses an Amazon S3 bucket for storing static files and media uploads. An interactive, RESTful API was also constructed using Django REST Framework, and documented using DRFDocs.

Behind the scenes, the app makes use of Heroku config variables, functionally equivalent to environment variables, to store sensitive information needed by the app, such as access keys and authorization parameters. For development, we also constructed a virtual environment with Vagrant that allows the user to run a developer build of the site on their machine, without installing any additional software.

3.2 Models and Database Design

Django models abstract database tables into python objects, which allows you to design and interact with your database in an object-oriented fashion, without worrying about writing specific queries [3]. This is very convenient, especially considering we planned to build a RESTful API.

The database was designed with the API in mind: the two focal points of the app, the Resources and the Positions, were given their own database models, `Resource_Profile` and `Position` respectively, and form the "roots" of the database. Most of the other models branch off them, and describe a more specific aspect about, or relationship between the more general root objects. The rest describe independent entities not necessarily tied to a resource or position.

A brief description of the most important database models are provided below:

- `Resource_Profile:` The root of the Resource model tree, this model contains the objective information for a student, including their associated `User`, netid, graduation year, picture, and transcript.
- `Resource_Repute:` Contains information that is either inferred by our systems, or is uploaded by others, including `auto_clear`, which indicates a student has been strongly recommended for hiring, academic standing, course remarks, and interview remarks.
- `Resource_Position_Info:` Contains information relating a resource to their time hired at that position. Includes the start date and end date, along with their performance rating and comments, if they exist. If the resource is currently hired, `end_date` will be `None`
- `Resource_Application:` Represents a resource's application to a position. Contains the time the application was submitted.
- `Position:` The root of the Position model tree, this model contains all critical information about a position, including its name, its slug (a unique identifier),

the `User` corresponding to a staff contact, and tables of resources in various stages of the application process.

- `Position_Offer:` Represents a hiring offer presented to a resource. Contains the time the offer was extended.
- `Remark:` Represents a general comment about a resource, and may be used for several topics. Stores the author and receiver of the comment, along with the time it was written.
- `Time_Slot:` Represents a half-hour block of time used by the scheduler (See "Scheduler" section). Stores the resource or position it is describing, and whether the time slot is "active". For Resources, "active" means the the student is free at that time, and for Positions, it means that block is required.

3.3 Authentication and Permissions

Users are required to login through CAS before being able to access the site. It is authenticated against Princeton's system, so users just use their standard netid and password. Authentication for the API is currently session based, and requires the user to log in using an account associated with a head manager or site administrator. Eventually, we plan to change this to use token based authentication instead.

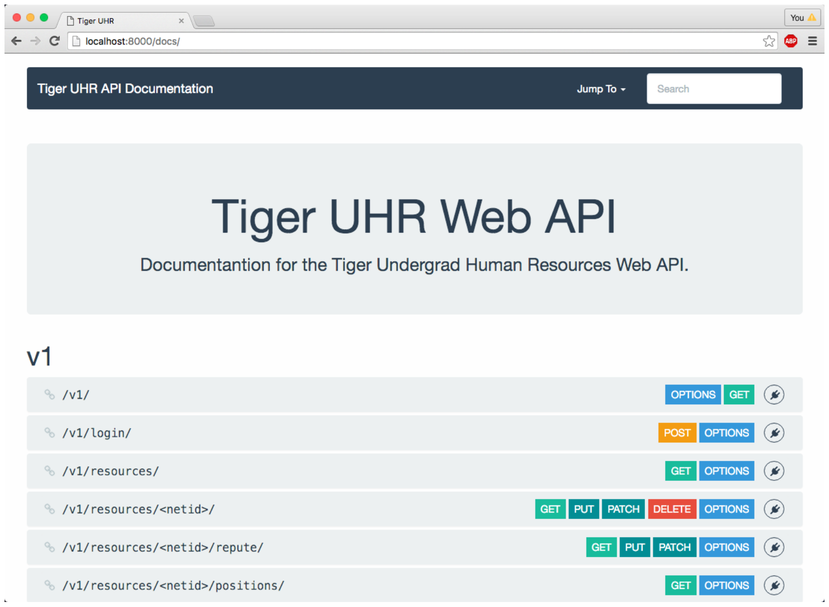
Permission handling is performed using Django's default User permissions system. There are five permission levels, detailed below:

- `Admin:` Has access to everything, and permission to perform everything.
- `Head Manager:` Has access to everything on the site, the API, and permission to add new Positions.
- `Manager:` Can view all information for resources, and can access management pages for the positions they control.
- `Assistant:` Can view abstracted information for resources, and can view, but not edit, management pages for the positions they assist.
- `Resource:` May only view and edit their own profile page.

3.4 API and Documentation

A REST-compliant API was built using Django REST Framework [4]. The framework works very well with databases designed in an object oriented manner: it provides a `Serializer` class that wraps each of your models, and allows their content to be parsed and rendered from several content types such as JSON. The serializers, and the provided class mixins, can then be used to autogenerate REST API methods, which can be customized by overriding them.

The framework also features a live endpoint system that allows to you to interact with the API like a typical website. When you visit an active API URL, such as


Tiger UHR

localhost:8000/docs/

Tiger UHR API Documentation

Tiger UHR Web API

Documentantion for the Tiger Undergrad Human Resources Web API.

v1

```
/v1/                                 OPTIONS GET
/v1/login/                           POST OPTIONS
/v1/resources/                       GET OPTIONS
/v1/resources/<netid>/               GET PUT PATCH DELETE OPTIONS
/v1/resources/<netid>/repute/        GET PUT PATCH OPTIONS
/v1/resources/<netid>/positions/     GET OPTIONS
```

Figure 4: Documentation of TigerUHR's API

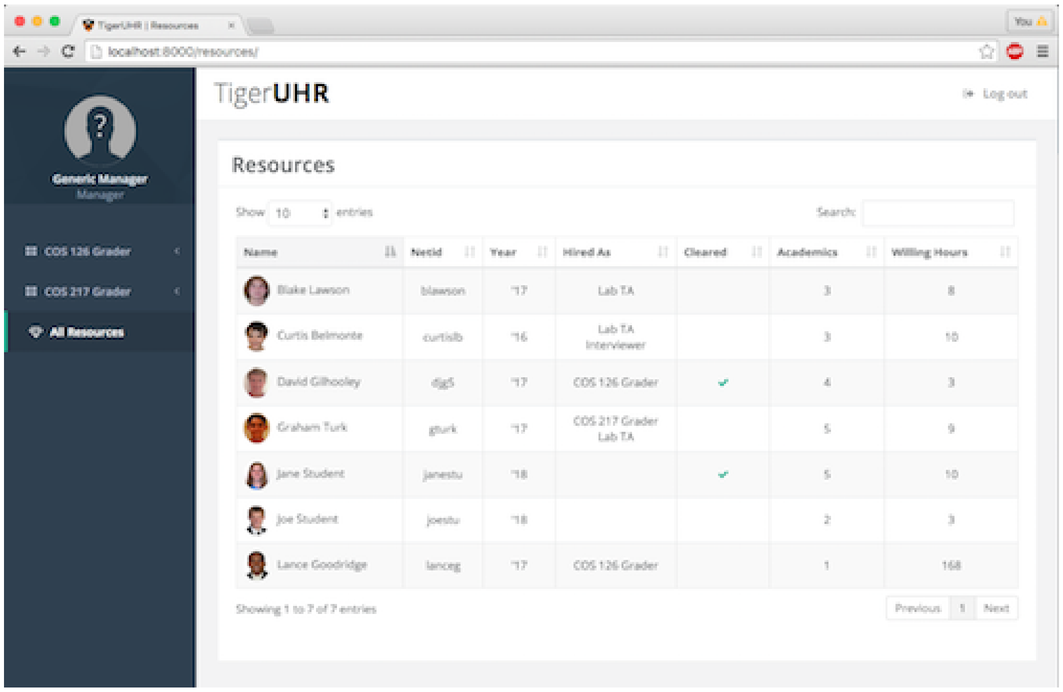
`/api/v1/resources/` for example, you will be presented the `GET` data for that request presented in JSON format, along with parameters to send a `POST` or `PUT` request at the bottom as well.

Finally, we recognize the importance of having clear documentation for an API, so we used its companion library, DRF Docs [5] to do that. Upon visiting `/docs`, you will be presented with all active API URLs, along with a description of what they do, which HTTP methods are allowed, and the parameters each one expects. An image of the documentation page is presented in Figure 4.

3.5 All Resources Table

Figure 5 shows the "All Resources" table, viewable by hiring managers and assistants. Here, you can view, sort, and search through all registered resources at once. This table provides the identifying information for each student, their job status, and a general indication of their preferred time commitment and academic standing, without giving sensitive details away. The *Academics* column on this page just shows a reflection of a student's average COS grades; eventually, we plan for hiring managers to be able to specify which courses are important, and how much weight each one holds for their own specific academic standing calculation.

Clicking on a resource's name takes you to their profile page, where you can see more detailed information (See "Profile Page" section). Clicking on a table's column header allows you to sort by that column in both ascending and descending order.



Resources

Show 10 entries Search:

Name	Netid	Year	Hired As	Cleared	Academics	Willing Hours
Blake Lawson	blawson	'17	Lab TA		3	8
Curtis Belmonte	curtislb	'16	Lab TA Interviewer		3	10
David Gilhooley	djg5	'17	COS 126 Grader	✓	4	3
Graham Turk	gturk	'17	COS 217 Grader Lab TA		5	9
Jane Student	janestu	'18		✓	5	10
Joe Student	joestu	'18			2	3
Lance Goodridge	lanceg	'17	COS 126 Grader		1	168

Showing 1 to 7 of 7 entries

Previous 1 Next

Figure 5: All Resources Table

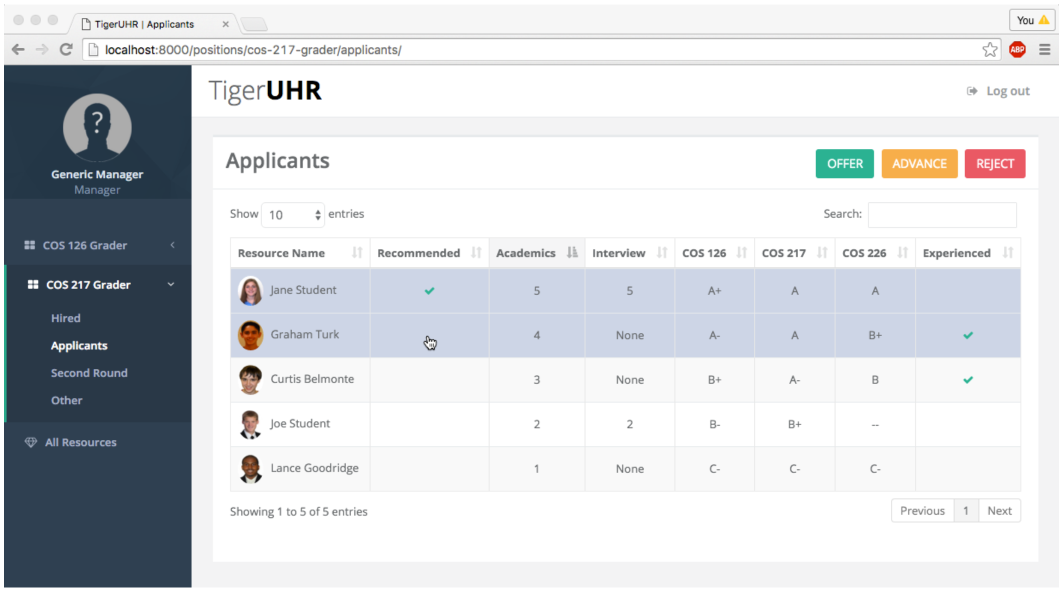
On the left side of the page, there is a navigation menu with links for managing specific courses. A hiring manager will only be able to see and access pages for the position they manage.

3.6 Management Tables

Figure 6 features an example management table for a position. Specifically this is the "Applicants" table, which you can reach by clicking on the nav link for a position, then clicking on "Applicants" in the menu that appears.

As explained in "Approach", hiring is a three stage process:

1. At the beginning of a hiring season, resources apply to all positions they're interested in.
2. First round selection begins. Hiring managers can see the list of applicants for their position, and either extend them an offer immediately, reject them, or advance them to a second round, where additional information may be requested. This additional information may be in the form of an interview, or an online form for example.
3. Resources deferred to the second round provide requested information, and the hiring managers review them. Then second round selection begins, where the hiring managers must choose between extending an offer or rejecting each student. Resources may either accept or reject position offers they receive.



Applicants

Resource Name	Recommended	Academics	Interview	COS 126	COS 217	COS 226	Experienced
Jane Student	✓	5	5	A+	A	A	
Graham Turk		4	None	A-	A	B+	✓
Curtis Belmonte		3	None	B+	A-	B	✓
Joe Student		2	2	B-	B+	--	
Lance Goodridge		1	None	C-	C-	C-	

Showing 1 to 5 of 5 entries

Figure 6: Example Management Table

This is the table of first-stage applicants, so the three options of extending an offer, advancing them to the next round, or rejecting them are available. Using the table is very intuitive: just click on the resources you want to select, then press the appropriate button. The button press will asynchronously call the corresponding API function, and update the table.

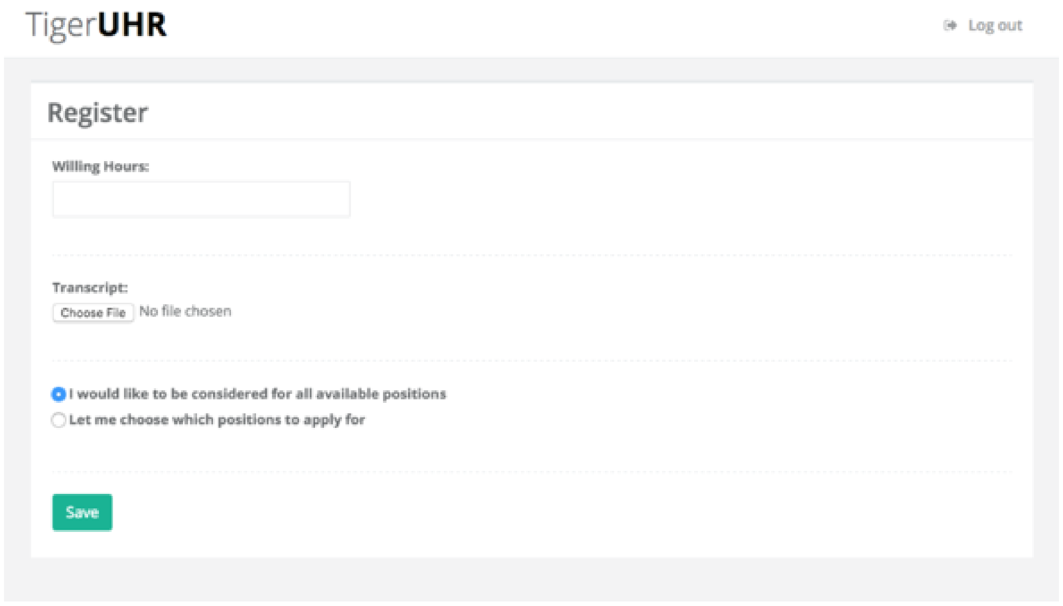
In addition, there are three other management tables for each position:

- **Hired:** Displays the resources currently holding the position.
- **Second Round:** Displays the resources who completed the second-round requirements.
- **Other:** Displays resources who have been extended an offer, but not replied yet, as well those who have been fired.

3.7 Registration

Figure 7 displays the registration screen. Registration consists of two pages: the first, shown in the figure, asks the student to enter the number of hours they're willing to work in a week, which positions they would like to be considered for, and to upload their transcript. The next page has the resource fill out a WhenIsGood-esque schedule of their free time.

Thus, registration is incredibly quick, and the student has to provide very little information; everything else is automatically collected by the system. A number of submodules were needed to accomplish this:



TigerUHR

Log out

Register

Willing Hours:

Transcript:
Choose File No file chosen

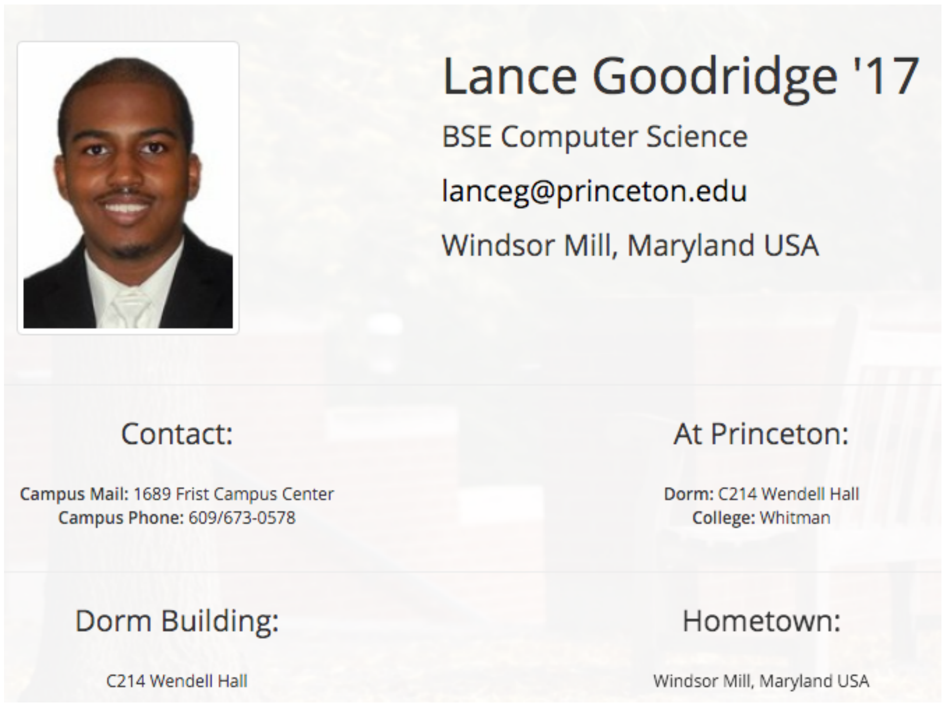
[✓ I would like to be considered for all available positions]

Let me choose which positions to apply for

Save

Figure 7: Resource Registration Page

3.7.1 Resource Scraper



Lance Goodridge '17

BSE Computer Science

lanceg@princeton.edu

Windsor Mill, Maryland USA

Contact:

Campus Mail: 1689 Frist Campus Center
Campus Phone: 609/673-0578

At Princeton:

Dorm: C214 Wendell Hall
College: Whitman

Dorm Building:

C214 Wendell Hall

Hometown:

Windsor Mill, Maryland USA

Figure 8: A Student's Page on Tigerbook

First, the resource scraper, which acquires the student's full name, email address, class year, and prox photo from their netid. Most of a student's identifying information is publicly available, provided you have something to search for them by, and can authenticate into the site of your choosing. Figure 8 shows the information displayed about a student on Tigerbook, with a similar setup used for Princeton's College Facebook. The student's netid is retrieved when they authenticate through CAS, so all we need the module to do is actually perform the web scraping.

First, the script opens a connection to Princeton College Facebook, authenticates, and searches for the provided netid using python's `urllib2` library. It then uses `Beautiful Soup` to scrape the desired information from the results.

In addition, it creates a folder for the student in the app's Amazon S3 bucket, and stores their picture there. The module uses the `boto` library to establish a connection with the bucket, and the authentication parameters for both the bucket and Princeton College Facebook are stored inside Heroku configuration variables for security.

3.7.2 Transcript Parser

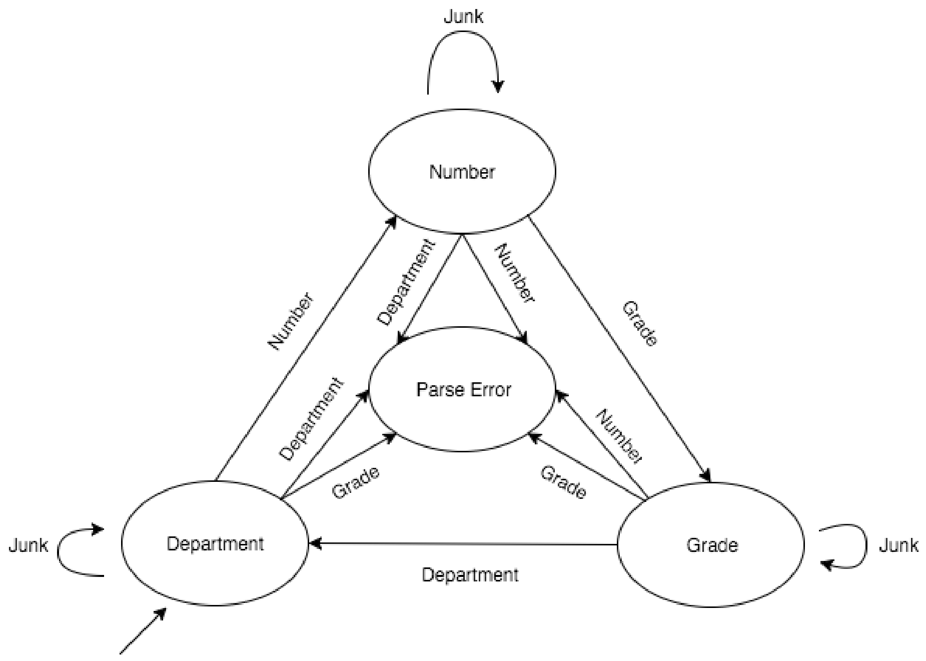


Figure 9: Simplified FSM of Official Princeton Transcript Parser

Next, the transcript parser, which extracts a student's grades from the transcript they upload. First the text is extracted, and the program verifies the transcript's authenticity. We wanted to support both official and unofficial Princeton transcripts, so the module then identifies which type of transcript it is, and calls the appropriate parser.

Both parsers use a state machine to keep track of the information it receives. Since there are certain invariants on the order in which information arrives, we can detect a parsing error if something arrives unexpectedly, or throw away data we know is unwanted. For example, Figure 9 shows a simplified FSM of the state machine used by the official transcript parser. For this type, we know the text will be extracted such that a course department will always come first, followed by the course number, then the grade received in the course, with junk data interspersed between the three pieces of information. We can throw away anything that isn't one of the three pieces, because it isn't important to us. If we receive one of the pieces out of order however, we know that something unexpected has occurred, and the parsing is no longer reliable, so we throw an exception and exit.

The module returns a dictionary relating each course to the grade received, and the semester it was taken.

3.7.3 Scheduler

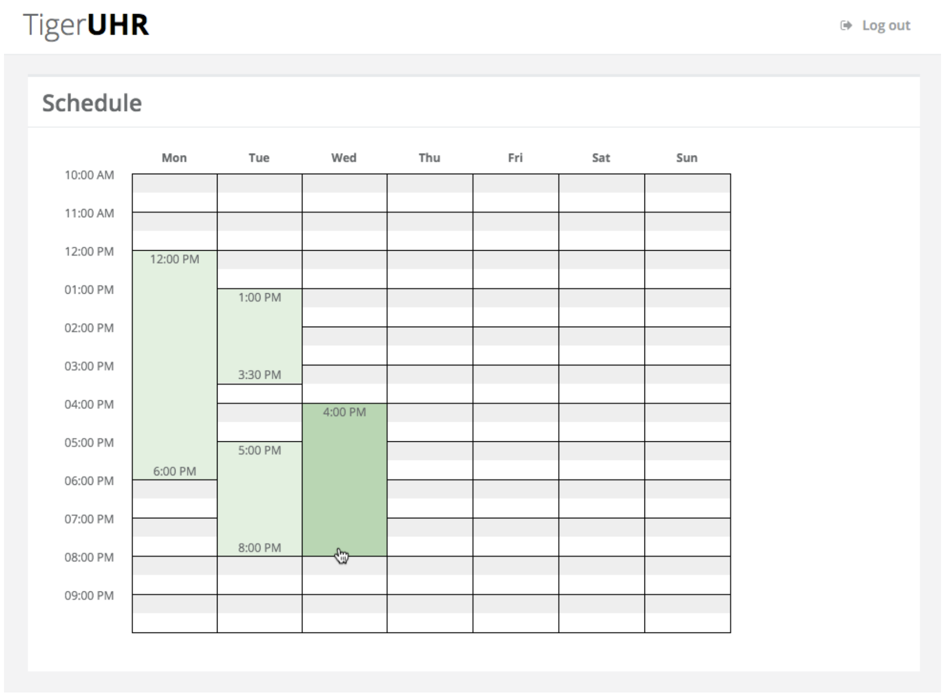
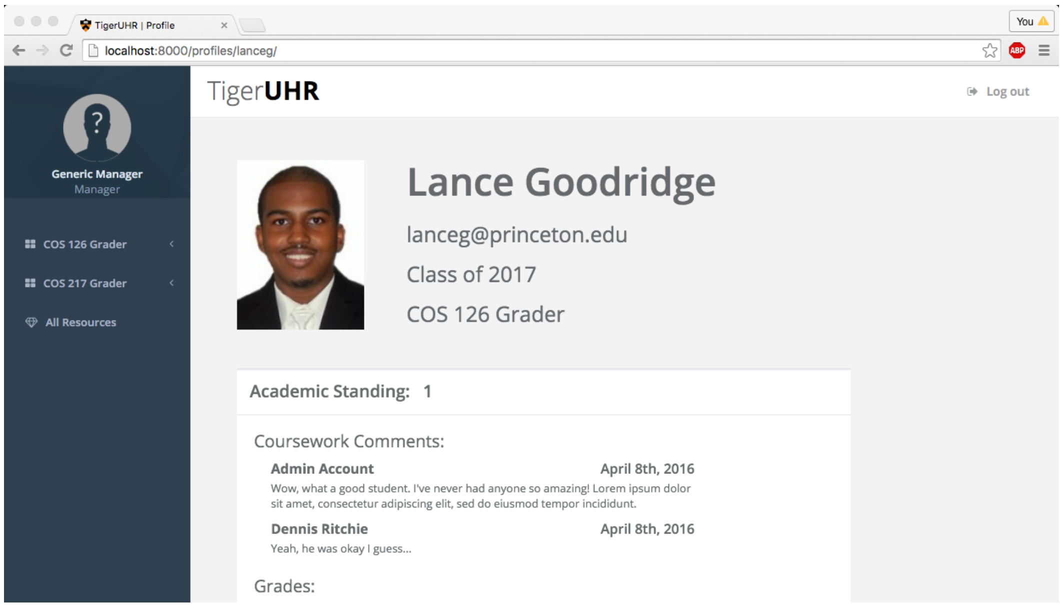


Figure 10: Scheduler Page During Registration

Finally, we designed a WhenIsGood-esque view for entering and displaying schedule information. We wanted users to easily tell what hours were required for a position, and input their available work hours as quickly and naturally as possible, so we modeled it after a WhenIsGood, since most people are familiar with how it works, and the design is very clean and easy to use. Our scheduler functions exactly the same way: you click and drag along the time slots your available, and it displays your selections in a slick, compact format. Figure 10 shows the scheduler in action, on the second page of the registration form.

These modules were implemented as standalone programs that could be used outside of TigerUHR if desired.

3.8 Profile Page



Lance Goodridge

lanceg@princeton.edu

Class of 2017

COS 126 Grader

Academic Standing: 1

Coursework Comments:

Admin Account April 8th, 2016

Wow, what a good student. I've never had anyone so amazing! Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

Dennis Ritchie April 8th, 2016

Yeah, he was okay I guess...

Grades:

Figure 11: Profile Page while logged in as a Manager

Figure 11 shows the resource profile page, while logged in as a manager. Here, a manager can view the student's complete identifying information, schedule, grades, and comments on their previous interviews and job performances. Interviewers and assistants can also view resource profile pages, but sensitive information, such as the student's specific grades, will not be displayed. Managers and assistants will be able to add comments on the resource's job or interview performances, provided they have the proper privileges to do so.

Resources can also visit their own profile page. There, they can reupload their transcript, update their schedule information, or re-select which positions to be considered for. They can also accept offers and complete second round form requests. The resource will be notified via email whenever they have something to respond to, such as an offer or online form, so they don't have to keep checking the site.

4 Conclusion

4.1 Evaluation

The result is a webapp that meets my goal and guiding design philosophies:

1. **Intelligent registration and data collection:** Achieved through the Resource Scraper and Transcript Parser modules. Registration should take no more than 2-3 minutes, and should not discourage students from using the app at all.
2. **Painless application and hiring processes:** On the resource side, application is reduced to selecting which courses you are interested in; more work is only required when advanced to a second round. On the manager side, hiring decisions are performed through a point and click interface, with systems in place for requesting and reviewing further information.
3. **Powerful and intuitive management tools:** Management is done largely through sortable and searchable tables with clear purposes and functions.
4. **RESTful API:** Successfully constructed and documented.
5. **Extensible Code Design:** Code was written modularly and as loosely coupled as possible to support future feature implementations.

4.2 Future Work

This system is still in development, and is flexible enough to be extended in several aspects.

We plan to eventually integrate with other apps, which allows the system to collect more useful data and provide more powerful features. For example, we could connect with codePost, and track average number of comments per grader, which would allow us to match students who desire more feedback with the appropriate graders. We could also link with the Time Collection system, to automatically record and display work hours. We also plan to introduce data analyzing algorithms, which opens the door for autonomous performance evaluation, and advanced management scouting features. There is even the potential for generalizing the app, and marketing it to other interested universities. The possibilities are endless.

The current plan is to have the COS 126 Grading team use the system for hiring next semester, which will give us real data and user feedback to respond to. From there, we will expand to the rest of the department and beyond.

4.3 Closing Thoughts

We believe this app will be of tremendous use to the university. For the immediate future, the product will allow the COS department to be more liberal in hiring undergraduate workers, and will allow more students to get relevant work experience on campus. Eventually, we hope the system will be able to permeate and benefit other departments and universities as well.

4.4 Acknowledgments

I'd like to thank Jérémie Lumbroso for his advice and creative ideas throughout the project. I'd also like to thank Benjamin Parks, who provided the base implementation of the college facebook scraper.

6 References

[1] “Bamboo Testimonial.” HR Software for Small & Medium Businesses. Accessed April 15, 2016. http://www.bamboohr.com/

[2] CodePost. Accessed April 15, 2016. http://codepost-labs.herokuapp.com/

[3] Django Models. Accessed April 15, 2016. https://docs.djangoproject.com/en/1.9/topics/db/models/

[4] Django REST Framework. Accessed April 15, 2016. http://www.django-rest-framework.org/

[5] DRF Docs. Accessed April 15, 2016. http://drfdocs.com/

[6] Lab TA Queue. Accessed April 15, 2016. http://www.princetonlabtas.com/

[7] Office of the Registrar. Accessed April 15, 2016. https://registrar.princeton.edu/course-offerings/search_results.xml?submit=Search&term=1164&subject=COS

[8] “Representational State Transfer.” Wikipedia. Accessed April 15, 2016. https://en.wikipedia.org/wiki/Representational_state_transfer

[9] Tigerbook. Accessed April 15, 2016. https://tigerbook.herokuapp.com/