uCount - Software Development Challenge

TSA Conference @ Kaysville, Utah - Updated March 15th, 2016

Mitch Talmadge
Co-President

Email: mitch@mitchtalmadge.com
Cell: (801) 949 0103

Kevin Thorne
Co-President

Email: kevint@thedigitalnative.space
Cell: (801) 819 5480

Table of Contents

Problem Research

Every year, Bingham High School hosts the Sweethearts dance, which is preceded by a school assembly. Bingham High School's Technology Student Association (Bingham TSA), by tradition, takes responsibility for planning and running this assembly, as well as judging the nominees and picking a King and Queen. This year, the Sweethearts Committee agreed to allow the student body to vote on certain matters regarding the dance.

Allowing the student body to cast their votes securely, quickly, easily, and reliably requires a dynamic and robust web-based Polling System. Bingham TSA decided that the best way to ensure that the Polling System performed to laid out specifications and requirements was to create the system from the ground up. We decided to call the system *uCount*.

We looked at many different technologies while deciding which languages to use and how the system should be structured. We considered PHP, but found it difficult to manage database connections. PHP required us to write our own SQL queries[1], which is prone to human error and therefore not guaranteed to be reliable. Additionally, it is difficult to make relational mappings in the database using PHP. We also looked at Groovy on Grails and Ruby on Rails, but decided that we did not have enough experience in either language to ensure a bug-free system by the deadlines we had set. Java EE[2] was chosen for multiple reasons: our team has years of self-taught experience creating Java SE applications using Swing and JavaFX among other popular libraries, and Java EE seemed like a logical progression for our skill-set; Java EE's *Java Persistence API (JPA)*[3] provides a great object-oriented interface (also known as a object-oriented database management system - OODBMS) for modifying and working with the database, and is proven to be very reliable; The Java EE specifications have a large variety of modules for more expandability in the application if such is needed, including multiple front-ends if ever required; and Java EE is backed by a large community which could be helpful should issues arise.

---

[1] http://stackoverflow.com/questions/3483080/php-developer-looking-for-solutions-equivalent-to-java-ee-architecture
[2] http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html
[3] https://en.wikipedia.org/wiki/Java_Persistence_API

Problem

The main problem we faced was providing a solution that allowed students to vote easily and securely, while also ensuring that each student only got one vote. Although there are plenty of free voting systems on the internet, such as Google Forms, we could not use these systems because it is far too easy to vote more than once. We needed a system which could tie into the school's databases to authenticate Student IDs, allowing us to keep track of who has and has not voted.

In the past, Bingham TSA ran a Polling System based on PHP, and had to rely on the students' good-will that the entered Student IDs were valid, as we were unable to ensure that an ID entered by a student was an existing ID, rather than one made up by the student. After looking at the results in the database, it was found that some students would enter random IDs repeatedly, or in a pattern. It was obvious that students were abusing the system, and an accurate representation of the student body was unable to be guaranteed because of these nuisance students.

Solution

To combat the issue of student authentication, we started off by visiting our school's administration for assistance. We wrote a proposal for the IT Department of Jordan School District outlining multiple possible solutions to our problem, however, due to privacy concerns, Jordan School District was unable to provide us with a method of authentication that could be implemented with minimal effort on the student's part. Still in search of a solution, we visited the USOE's Information Security Department, and we were given permission to use any public methods of authentication.

Using Java Enterprise Edition, we began work on uCount, which took advantage of public authentication methods in order to validate Student IDs. uCount was designed specifically to allow students to vote, and provides ways to hook into schools' various methods of authentication.

We were able to authenticate all Student IDs through Jordan School District's public Overdrive library system, found at https://jordanut.libraryreserve.com/. This library system only accepts valid Student IDs, which fit our needs perfectly. Implementing Overdrive into our system was as simple as simulating the web headers sent by an average web browser, essentially logging each student into the library system every time they cast a vote, and checking whether the login was successful or returned an error, such as in the case of an invalid ID.

Social & Educational Benefits

uCount is designed with schools in mind, and can be used in a wide range of scenarios. uCount is useful in situations where a student vote is important or useful; for example, when voting for a favorite talent, voting for an SBO, or voting for a new school policy. The interface is very simple to use, and multiple polls can be created so that they are ready to go whenever needed.

uCount allows students to get involved in the democracy of their school in a way that may not have been previously possible, or that was difficult and time-consuming, such as when using *Scantron* forms. We believe that although uCount is in its infancy, there is much potential to turn this system into one that is usable and customizable by any school who finds interest in it. The fact that uCount is electronic, quick, and easy to access is bound to show a greater turnout at school polls than was previously seen through Scantrons and other forms of voting. uCount could easily be setup to run on multiple iPads during lunch, which means students don't even have to pull out their phone to vote; they can vote in seconds right as they enter or leave the lunchroom.

uCount also has the potential to be used in many other situations requiring a vote, not just those that are a part of an educational setting. Many categorical statistics could be gathered using uCount.

Plan of Work Log

Date	Task	Time Involved	Member(s)	Comments
1/8/16 - 1/9/16	Database Design	2 Days	Kevin and Mitch	We made many revisions, and gained a great understanding of database design.
1/12/16 - 1/13/16	Entity Bean Creation	2 Days	Kevin and Mitch	This took us a while because this was our first time ever using J2EE.
1/19/16 - 2/16/16	Web Interface Development	1 Month	Kevin and Mitch	This was an on-going task which we cannot ever really call “finished.” There is always room for improvement.
2/6/16	Design Logo	1 Hour	Mitch	The logo was very simple and easy to create.
2/7/16	Implement Overdrive Authentication	6 Hours	Mitch	Overdrive authentication worked very well.
2/7/26 - 2/10/16	Implement Picture Uploads	4 Days	Kevin	We ran into some issues involving AJAX and the file upload form.
2/11/16 - 2/16/16	Live Results Page	6 Days	Kevin	The Live Results Page was very successful and allowed students to see the results on the projector in real-time.
2/12/16 - 2/13/16	Testing	2 Days	Mitch	We used loadimpact.com to simulate hundreds of virtual users connecting within one minute. See the testing section for more details.

Methodology

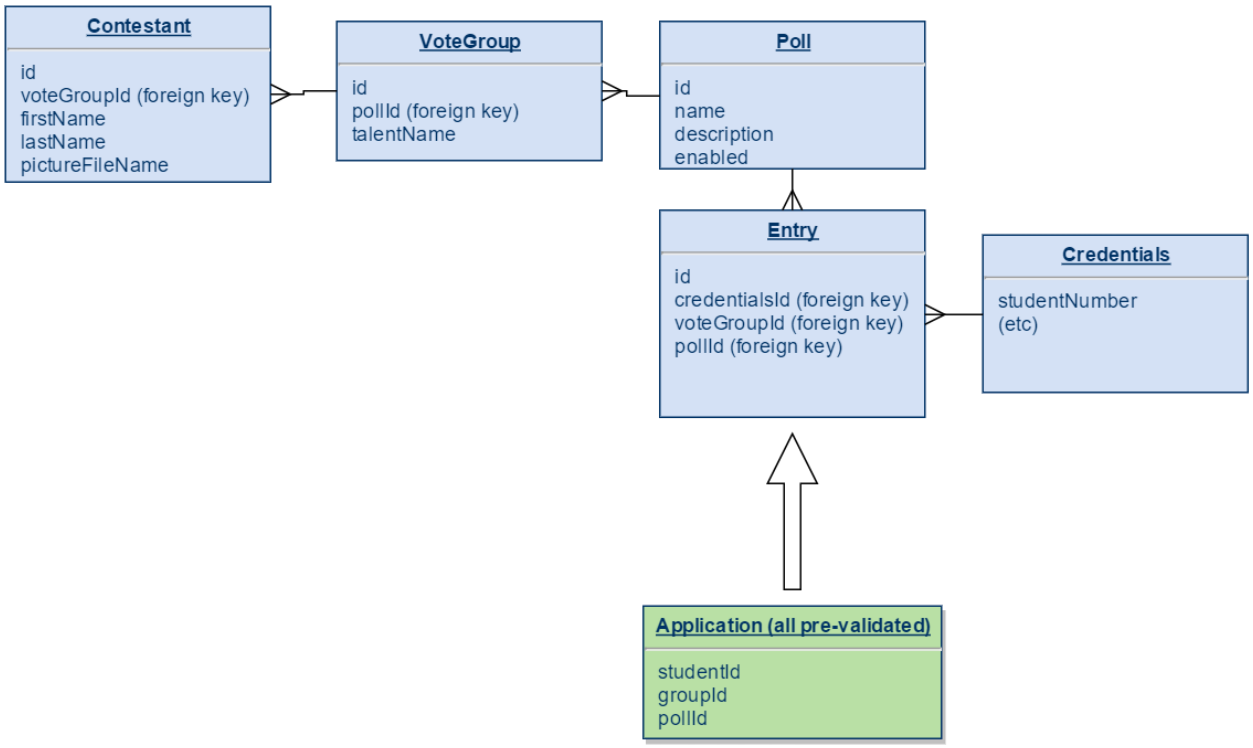
Project Requirements

Expressed in the most brief way, uCount is to perform as a polling system, allowing up to hundreds of students to access the same poll at the same time, from any device they have on hand. It is designed with schools in mind, and should provide a means of authentication that is reliable, quick, easy for students to use, and trustworthy. Students should not be allowed to vote more than once on any given poll. It is imperative that the interface is easy to use and not cluttered with unnecessary features or buttons, and that there is no confusion on how voting works.

High-Level Software Design

Most of the design, in the concern of deadlines, was more specific than it was dynamic. This didn't hurt the application, however, refactoring is planned.

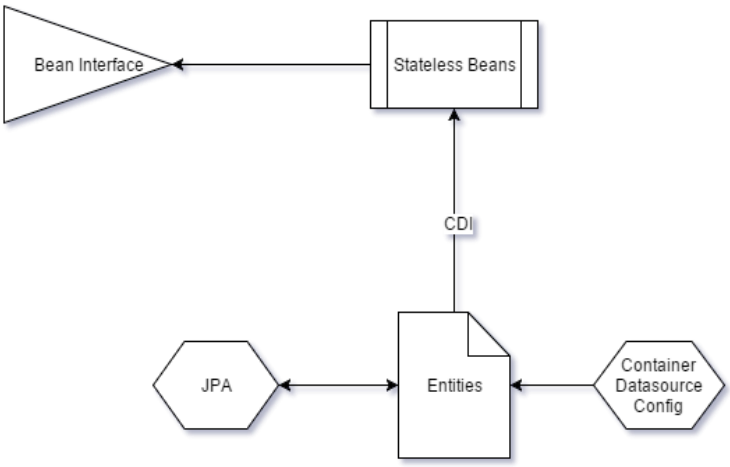
Database Design



The database design is normalized in the 3rd Normal Form (3NF)[4]. In a nutshell, this means that all non-identifying fields are dependent on the primary key. As previously stated, this database design

[4] https://en.wikipedia.org/wiki/Third_normal_form

does offer some flexibility, as we are not storing specific “nominee couple” contestant table entries. Rather, each person is stored as a Contestant within a VoteGroup.The VoteGroup table is generically named for a reason; you can organize your Contestants in anyway you prefer. That was the main idea for most of the database design: leaving an open structure so that the application can be dynamically used in many scenarios.



Enterprise JavaBean Context Design

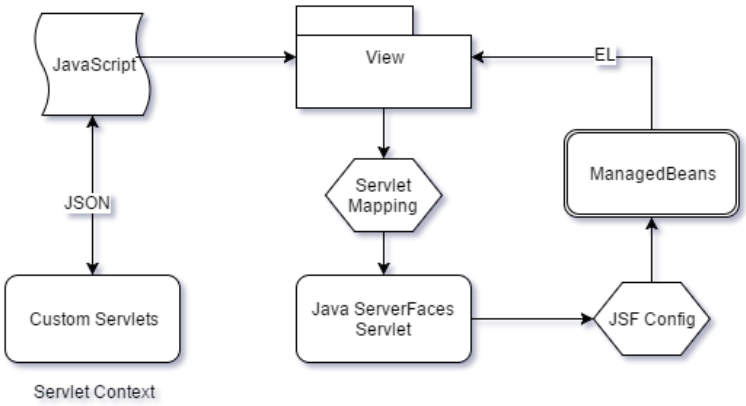
The EJB (Enterprise JavaBean[5]) context design was kept simple. There is a Stateless[6] bean tied to each Entity, which is a Java Object representation of a table. These beans are managed by the Java Persistence API (JPA) and administered to their respective tables. We call these beans “Services.”

To further optimize the backend, each bean extends a common *Service* class. This Service class has a type parameter that each implementation class wraps its respective entity type to, and contains methods to **C**reate, **R**ead, **U**pdate, and **D**estroy entities and database entries. This is commonly referred to as the **C.R.U.D.** operations.

Each Service implementation has its own set of specific methods containing queries specific to the Entity it owns (e.x. “findUserbyName(String)”). One of the libraries implemented in uCount to help generate accurate and reliable queries was QueryDSL[7]. This made the queries typesafe, and rather than entering raw, prepared query strings, QueryDSL will automatically generate Query types of each Entity. This in turn changes the ambiguity of a prepared statement string into syntax-validated methods.

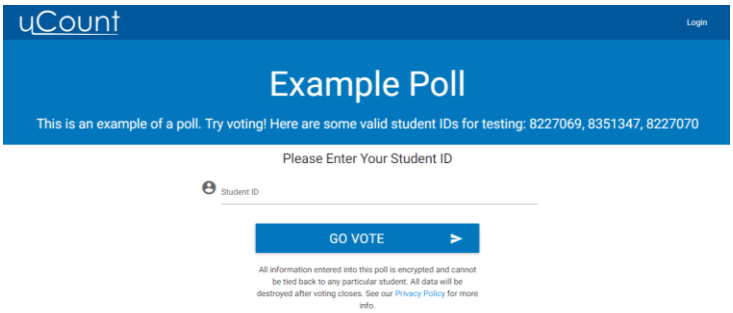
[5] https://en.wikipedia.org/wiki/Enterprise_JavaBeans
[6] http://docs.oracle.com/javaee/6/tutorial/doc/gipjg.html#gipin
[7] http://querydsl.com/

JavaServer Faces and Servlet Web Page Design



We chose to use a web page as our front-end because it is very easy to access from any device, and requires no installation.

We also decided that the JavaServer Faces technology should be used. JSF gives us the ability to abstract requests and responses and to bind properties to Managed Beans so that development is smooth and quick. JSF already follows a Model-View-Controller structure and made coding to convention easy, resulting in a beautifully structured piece of software. Any time we needed to dynamically load information from JavaScript, we can easily do so by accessing a class model interpreted into JSON from a servlet.



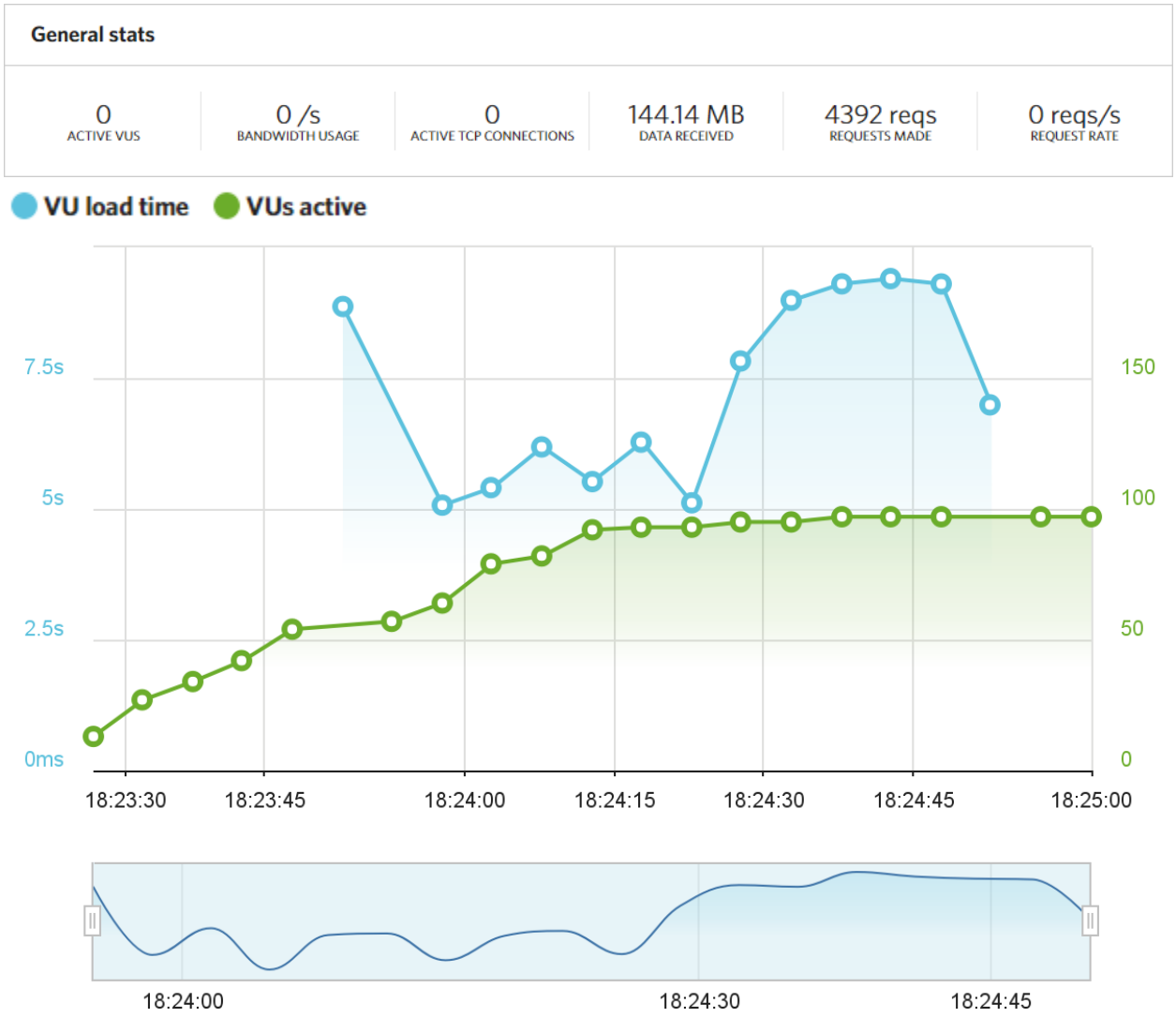
When designing our front-end web pages, we decided to follow closely to Google's Material Design Specifications[8]. To help port the specifications into CSS and Javascript, MaterializeCSS[9] was used. MaterializeCSS greatly aided the readability, appearance, and feel of the application. MaterializeCSS also helped us to quickly design a responsive system which looks great on computers, tablets, and our main target group: smartphones.

[8] https://www.google.com/design/spec/material-design/introduction.html
[9] http://materializecss.com/

Testing

Due to the large audience of this application (over 2800 students at the maximum), a majority of the testing done was through comprehensive load tests. To ensure that we could handle a large audience, we simulated 100 virtual users connecting within one minute. All other testing had to be done manually in order to meet our deadlines. (JSF is quite difficult to unit test). In order to get accurate load tests, we designed a page which, when connected to, would simulate a user entering a valid ID, picking a Vote Group, and casting their vote, with all the required SQL queries involved.

To perform our load testing, we used Load Impact[10]. Here are the results of the four major tests we performed[11]:

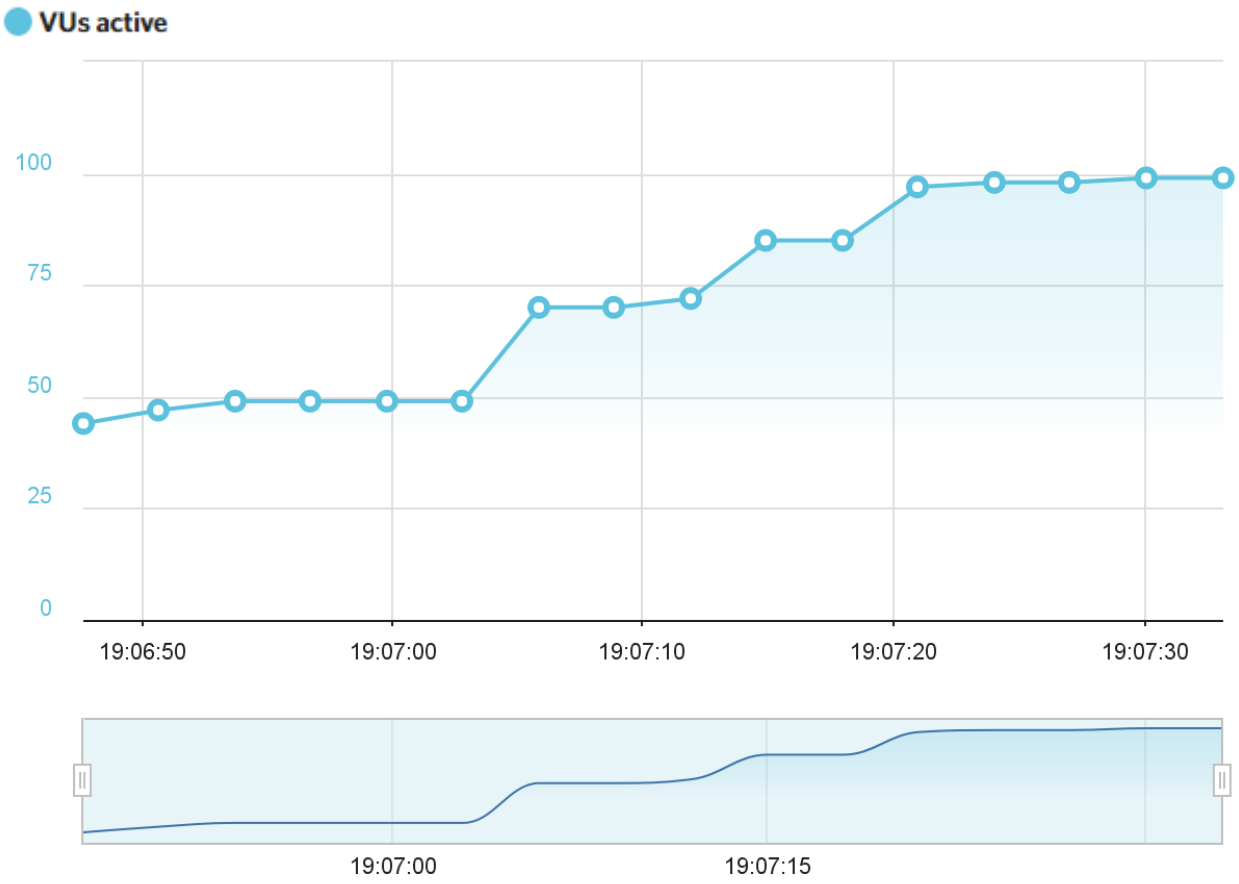


[10] https://loadimpact.com/
[11] http://bit.do/uCountTest1

Test 2:[12]

General stats

0	0 /s	0	77.28 MB	2352 reqs	0 reqs/s
ACTIVE VUS	BANDWIDTH USAGE	ACTIVE TCP CONNECTIONS	DATA RECEIVED	REQUESTS MADE	REQUEST RATE

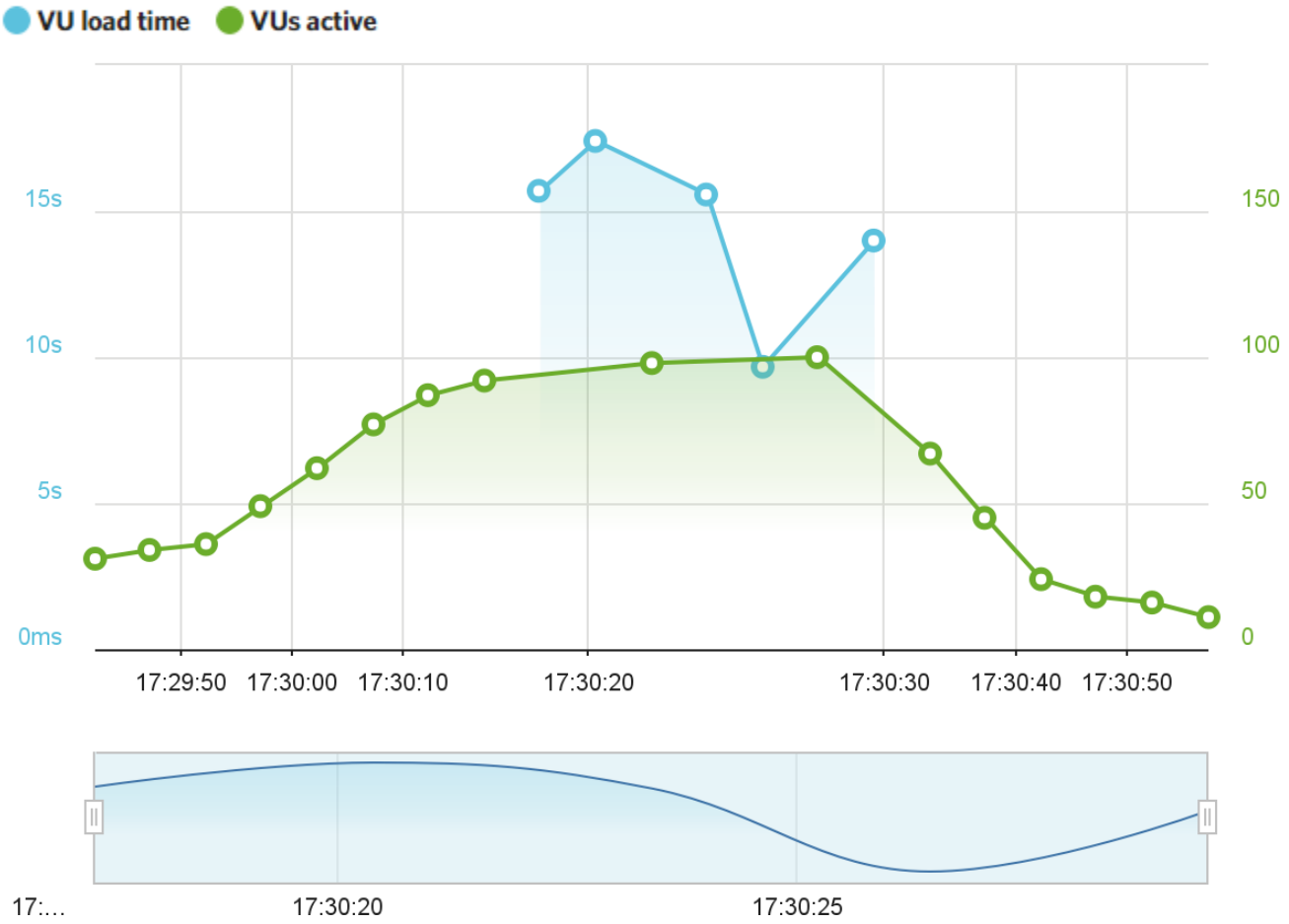


[12] http://bit.do/uCountTest2

Test 3:[13]

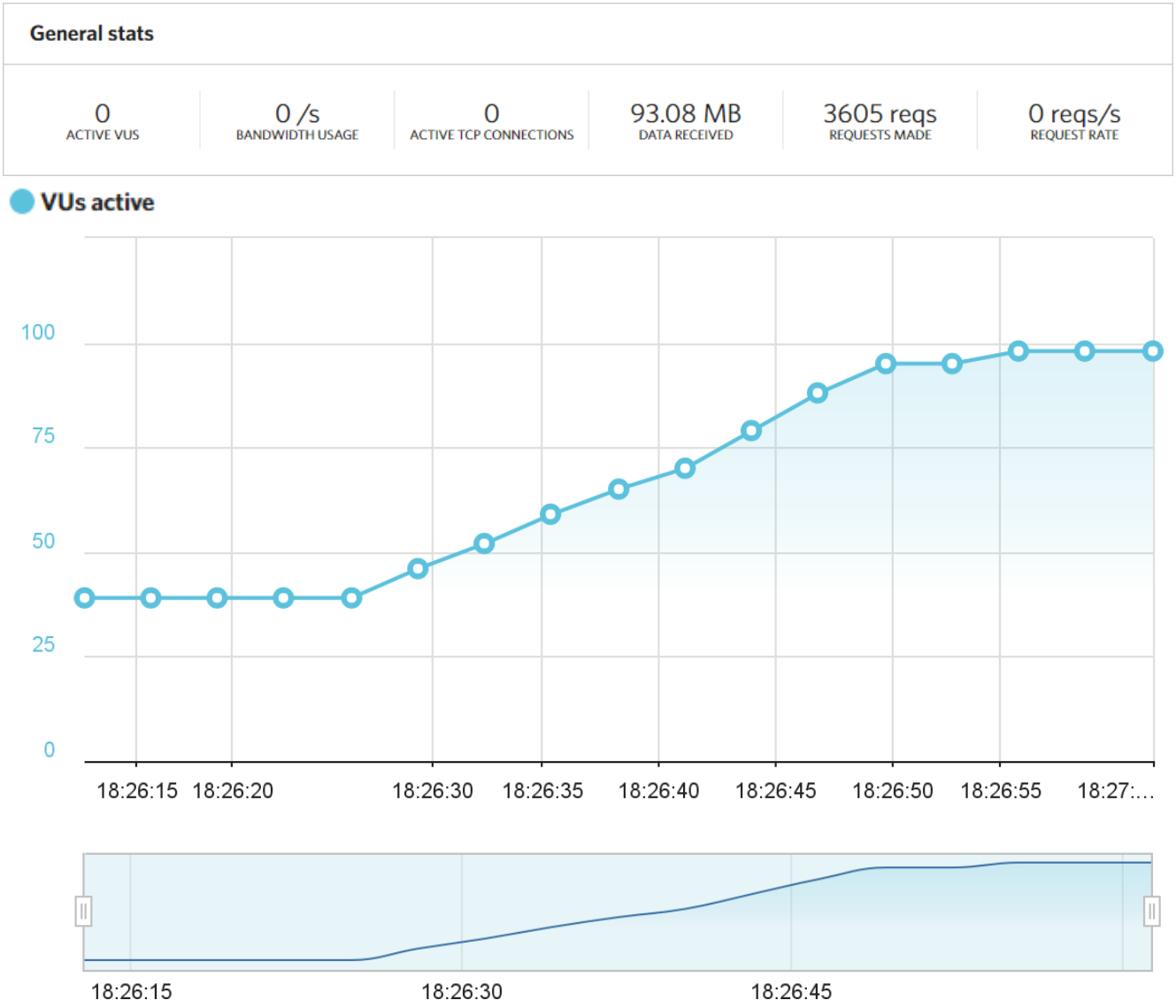
General stats

0	0 /s	0	74.02 MB	2234 reqs	0 reqs/s
ACTIVE VUS	BANDWIDTH USAGE	ACTIVE TCP CONNECTIONS	DATA RECEIVED	REQUESTS MADE	REQUEST RATE



[13] http://bit.do/uCountTest3

Test 4:[14]



These load tests were extremely valuable, and revealed major issues in how our beans were being used in the backend. They showed that we were not properly asynchronously threading our beans, which resulted in many clients completely timing out due to the beans being occupied by other clients. We solved this problem in time for the live voting with the help of these load tests. Additionally, we were able to improve loading times by integrating our website with CloudFlare's Content Delivery Network, which helped us cache images so that they would load fasted on connected devices.

[14] http://bit.do/uCountTest4

Project Evaluation

Overall, uCount worked very well and performed as we intended. Unfortunately, we encountered a large issue at the very last minute, and the quick patch we uploaded introduced yet another problem. Just before the application was to go live, we noticed that the ballot was not randomized, which gives the first Vote Group an advantage over other Vote Groups because of the students who choose them simply because they appear first. This is a real world issue and we needed to correct it, and did so by randomizing the ballot for every user. After quickly and stressfully redeploying the application *during* the assembly we were in charge of, it appeared that all problems were solved. However, later in the day after many votes had been cast, we noticed that the number of votes in each Vote Group were uncomfortably similar. After close examination with multiple test votes, it became clear that our previous patch introduced another bug that made every entry go to a random Vote Group rather than the one chosen. This was overlooked in the stress of the moment and taught us a real world lesson on why ample testing is so important. We decided to run a re-vote, which, after we had fixed all previous issues, gave us the results we expected. We are extremely pleased with the results of our hard work, and feel amazed that we knew very little about Java EE before starting this.

Future Development

The future of uCount will be mainly focused on expanding the types of surveys that can be taken. uCount is currently limited to “Choose One” or “Choose Many” type surveys, and we feel that much more could be accomplished with our program in terms of the types of surveys that can be taken. We would also like to be able to allow users to rank Vote Groups, or answer free-response questions using text. Other improvements that will be considered include the ability to run multiple polls at the same time; allow polls to be public or private; implement drag and drop editors; provide the administrators with a settings page; and potentially provide an app that can access the backend. Because we chose to use Java Enterprise Edition, we have the flexibility to implement all of said features with far less effort than would be required of other programming languages, like PHP.

References

1. http://stackoverflow.com/questions/3483080/php-developer-looking-for-solutions-equivalent-to-java-ee-architecture
2. http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html
3. https://en.wikipedia.org/wiki/Java_Persistence_API
4. https://en.wikipedia.org/wiki/Third_normal_form
5. https://en.wikipedia.org/wiki/Enterprise_JavaBeans
6. http://docs.oracle.com/javaee/6/tutorial/doc/gipjg.html#gipin
7. http://querydsl.com/
8. https://www.google.com/design/spec/material-design/introduction.html
9. http://materializecss.com/
10. https://loadimpact.com/
11. http://bit.do/uCountTest1
12. http://bit.do/uCountTest2
13. http://bit.do/uCountTest3
14. http://bit.do/uCountTest4