Mezzo - Report

Delaney Granizo-Mackenzie, dgranizo@ (PM)
Mike Mulshine, mulshine@
Anna Ren, aren@
Steven Tran, stran@

"DuoLingo for music!" -- That has been our guiding motto throughout this project since we decided to make a musicianship web application in that first meeting together. By chance, three out of the four of us had prior experience with MacGamut through Princeton's intro music theory courses and commiserated over its terrible interface, clunkiness, and general awfulness as a piece of software. Over the last several weeks, we've strived to create something, to put it bluntly, better. After many hours of coding and meetings and deliberation, we've produced the beginnings of what we hope is an application that is useful, beautiful, and fun: Mezzo.

Our milestones were not met exactly according to plan, but it would have been very surprising if they had been. We each had very little experience working in a group to make a well-balanced, useful, functional, attractive website. Most of us had absolutely no web experience at all. Therefore, we didn't know how long things would take or how to best utilize our time at various stages of the project. It was hard to know where and when to get started on various tasks, especially given the numerous options we had open to us in terms of languages, frameworks, etc. Some of our group members were used to very structured COS assignments: "do this using this and this." It was a little difficult to adjust to the "do whatever you want with whatever you want" mentality of this class. Sometimes it feels like every class at Princeton teaches you how to Google in a better, smarter way, and this class definitely helped us a lot with learning what to search for when programming. We definitely underestimated how much work implementing everything we were imagining would be, but all of us learned a lot about the process of working with a group to produce a web application. There was a steep learning curve, but we all have a much better grasp on how sites are set up and how to make them work now. We all had a lot of fun running into and overcoming obstacles along the way.

We started using GitHub at an early stage and ran in to our first issues here. We should have put more thought into how we would all collaborate in its use. We tried to be responsible by implementing the policy that we always committed and pulled before pushing, but we still occasionally ran into issues that would set us back a bit, or make us hesitant to move forward with changes in functionality that were difficult to

implement in the first place. We were using git from the command line and may have benefited from using their GUI application: we didn't realize it existed until a few days before our demo! It also took us longer than it should have to implement the rule that one should push only after fixing all merge conflicts. Occasionally we had to do a hard pull or manually fix a lot of merged changes because we were not as regularly diligent as we could have been about committing and integrating changes as we developed on our independent local machines. Once we we came to terms with our use of GitHub, we began developing.

Our first goal was displaying a staff and notes on a webpage, which was met in a timely manner. At first we thought that we might use something like Sibelius, Finale, or Lily Pond to generate staff notation. However, the user would be required to download these applications (not to mention *purchase* them) if we wanted to dynamically generate music staff and notation. We quickly discovered a Javascript based notation API called Vexflow. Vexflow enabled us to produce arbitrary music notation in an HTML canvas with relatively little Javascript. All code could be executed from browser with no external dependencies. It took some time to get used to integrating Vexflow with HTML, but these difficulties were likely due to the fact that none of us had ever done much Javascript coding.

Audio functionality was completed with similar efficiency, but our approach was at first unguided. For a long time, we thought we would end up using some audio programming language like ChucK, developed by Ge Wang and Rebecca Fiebrink (who used to here at Princeton), or Max/MSP. We soon realized that this would take away from the portability and accessibility of our website, as it would likely require users to download outside software. This would lead to all sorts of difficulty with inter-hierarchical communication between layers of our code/website (browser, business logic, server, etc), and would also make us reliant on the user's expertise in using an outside application to guarantee the success of our website. We needed a browser based audio source that would allow for dynamic production of clean tones. We found in the duo of RiffWave.js and HTML5 Audio, as described in internals. Successfully implementing audio functionality was a big step for us and it certainly boosted the group morale with respect to confidence in our ability to meet the practical standards we had set for ourselves at the start of the project.

The first real roadblock that we ran into was actually the one that kept popping up. In the beginning, we weren't sure what to use for anything. We started out thinking we would use Django for our web framework, MongoDB for our database, and Node.js because we knew that they were really popular platforms and cool people liked them. After trying to learn about them and how to use them with each other, we realized that there was no reason for us to use MongoDB because the data we wanted to store fit well in to a relational database scheme. We also realized that there was no point in

integrating Node.js because we weren't building a very large/scalable application. After we threw out our buzzword mentality, we were still a little at a loss as to what would be the best set of tools for us to use.

Our initial decision to use Heroku for app-hosting proved to be a good one. The site was very intuitive and provided excellent tutorials for everything that we needed done. We learned that having one group member in charge of pushing to Heroku limited the number of conflicts we could have in merging changed files. This decision worked out very well. We encountered many issues, but none from Heroku. Moving forward from that decision, we decided we were going to use PostgreSQL, since that interfaced well with Heroku from what we could tell online. After thinking about it some more, we finally decided to use SQLite because it is built into Django. There was no reason for us to consider using anything else when Django makes it so easy to use SQLite, but the sheer amount of options that we had made it feel like we should explore those options and choose something cooler. We realized that it was a lot easier to stay with the preset tools that Django came with than to experiment with other stuff unless we really needed to use something else for functionality reasons. After skimming through tutorials that we found through directed searches and reading tutorial after tutorial about a lot of different products, many of which we did not use, we realized that it would be better to focus on getting things to work as quickly and simply as possible using only the means that we had and already understood, unless problems arose. Our work went a lot more smoothly from there.

Another lesson we learned about technology uses arose when we were trying to send messages between the database and Javascript. After 'asking Google,' it seemed like there were many ways to do this, and we started out trying to use Ajax. However, after spending a day or more with Ajax, we discovered that there was a free library called dajax that did exactly what we wanted to do with nice, simple examples in their tutorial. We probably could have figured out how to do the same thing using Ajax, but it would have taken a lot longer to search through documentation on how to do it, and it probably wouldn't have looked as clean in our code, either.

Looking back, databases ended up being really easy to work with, but we only figured out how to use them after spending a really long time trying things that didn't work. After trying to manually set up a database using Django models and forms for user registration, we realized that Django already has a standard User model that offered everything that we wanted. We realized that we didn't even have to make a new form and could just import one from Django. This is an example of a situation where we did nearly a full day's worth of research for a couple of lines of code, but got the functionality that we wanted.

The bigger pain with databases came later, when we were trying to store user level data in the database. We decided that we wanted to keep this information separate from the user registration information. At first, we thought that we needed two databases to do this whereas we actually just needed two tables in the same database. Because of this, we spent a decent amount of time trying to generate a new database using South migrations and adding more databases and database routers to settings.py. Later, when we realized that we could just add another table to the existing database and that Django would do everything automatically if we just specified a new model in models.py with all of the fields that we wanted in the database, life got a lot easier.

Through all of this research and work on databases, we learned that there already exist solutions to many of the problems we were bound to face, and that it's okay to use these solutions even if you don't understand every little detail of what is going on behind the scenes. Many people have already worked very hard on the same problems, and have produced and published valid software solutions online - it would be a shame for us to waste time doing the same things they did, thus stagnating our production, and for their work to go unnoticed or unused. Instead it's wise to "stand on the shoulders of giants," as we mortals are constantly reminded. So, all in all, we found that spending some extra time researching these products and finding tutorials is definitely worth it when you run in to issues, as we did with databases.

Graphical, user design, and branding decisions were made along the way and were thus constantly evolving, as we thought they should. It turns out none of us had any design background to begin with - just a sense of what we thought looked good from a layman's perspective. However, we were all eager to try it out and offer our own (differing) opinions.

But it turns out that having four individuals with four opinions each equally trying to contribute to the look and feel of the application was neither efficient nor effective. A large portion of at least one meeting consisted of us just sitting around and arguing (in a fun kind of way) about a good name for the project. Similar debates raged about color scheme, general aesthetic, and even mascot. (As evidenced in the final product, we ultimately abandoned the mascot idea and changed our name quite a few times.) Through these inane debates, our group eventually realized that it would probably be most efficient to have one designated person be responsible for the look and feel of the site. The other members would speak up if they had strong objections, and we would all work together to commend or criticize design decisions, but would leave the coding details and executive decisions to one individual - in this case, Steven.

Since none of us had design experience (web or otherwise), we decided to use Bootstrap to aid in our styling. We were definitely aware of the danger of using this

particular framework - because of its popularity, so many modern websites end up looking like the same, prepackaged Bootstrap site. Regardless, we kind of needed a boost in starting and Bootstrap provided us with that.

We knew from the beginning that we wanted our website to reflect a clean, modern, and attractive aesthetic. We wanted our application to look like something users would *want* to use. Inspired by the ever-increasing trend of flat design, as well as educational websites such as DuoLingo and KhanAcademy, we set out to create the Mezzo look. To get us on our feet, we started with an incredibly basic template downloaded off of the Bootstrap homepage called “Cover” consisting of a gray background and some centered white text. For a long while, Mezzo was built off of this very standard, slightly ugly template while the functionality and site architecture was being built.

Eventually, our designated designer, Steven, had had enough of the derivative CSS. He instructed our design team -- also Steven -- to overhaul the look of the website. We implemented a new, bright, color scheme (blues,whites, and pinks - dubbed ‘Coral Ice’ by Anna), making decisions for newer, more modern fonts, and adding animations and an improved layout. In a spontaneous break from coding one night, Delaney and Steven snapped and edited some pictures and the beautiful blurred piano background image was born, adding to our clean and modern aesthetic.

On the path from demo day to Dean’s Date, Mezzo was looking better than ever with its trendy new name and pretty-good-looking front end, but we knew there was still work to be done. We changed the layout of the game pages as well as the hub homepage, unified some color choices, and added frosted glass to each page to make the font pop a bit more. Finally, in an effort to unify our image and add a bit of branding, we came up with our slogan - “musicianship made easy”. By selectively un-capitalizing our slogan and certain keywords on the site, we hoped to better convey a unified tone.

From laypeople to laypeople-with-a-bit-more-knowledge-about-design, we’ve certainly learned a lot about what works on a website and what doesn’t. Feedback, both internal and external, has taught us basics about website layout, font and color choices, and how design and functionality influence each other.

We initially intended for our website to have a bit broader scope than it has now. We wanted to implement not only interval and melody training/testing, but also rhythms and chords. We learned early on that our intentions for the scope of the website were a bit ambitious, and we made it clear to our advisor TA (Chris) that we would focus on intervals and melodies. Incorporating all of the categories we had initially agreed on would essentially have required us to develop a whole entire music theory course online, and none of us are music theory experts, nor do we have

teaching skills in the field. We decided to cut down to just intervals and melodies because these two categories were most in line with our original intention of pursuing an *ear*-training program, which typically means a program that helps users identify intervallic relationships between sounding notes. The other game categories like rhythm and chords that would undoubtedly be useful to people interested in ear training and we will try to implement these in the future, but they are not part of our core functionality.

In testing our website, we asked friends and family to help us out, and particularly encouraged people with some type of musical background to take a look, as aspiring musicians will likely make up the bulk of our user base. Most were satisfied with the basic functionality of the website, but aggressive users and mouse-clickers uncovered some bugs where pages would load before database calls finished executing and therefore would not load properly. Thus began the database/javascript debugging process hinted at earlier in the paper. We initially mistook these conflicts as database specific issues - we thought that our database code was just not updating or was being corrupted. But after using Chrome and Firefox tools to debug, we found that our database was behaving as it should and, rather, that javascript was continuing execution of code even as database values were being set or retrieved, so that users could click a button prematurely and go to a corrupted page. This happened when new games pages were being loaded - the games being loaded depended on the user level, and the call to the database to get user level was sometimes slower than the page loading. We learned a lot about the ins and outs of the Javascript setTimeout function and used this to disable buttons for certain durations of time so as to allow for database storage or retrieval to occur.

If there were more time, we would have liked to implement additional versions of the intervals and melodies games. Right now we have tests in which the user hears an interval and has to select which interval he or she hears by selecting the corresponding notes on the correct music staff. We would like to have tests where a user hears an interval and is asked to select the correct interval name from four given interval names, without seeing the notes on a staff. We would also like to have training sessions, in which users interactively step through a few games or pages that teach music theory, intervals, melodies, and how best to use this website.

Looking into the future, we would like for this website to serve as a supplement to introductory and intermediate music theory courses at high schools, universities, and conservatories. We have talked about plans to reach out to the music department here at Princeton to see if they might be interested in using our website for MUS 105/106, the university's intro music theory courses. With their help we would tailor the website to serve the practical needs of the courses, perhaps implementing administrative accounts, class groups, and a grading/ranking system. If this sort of pursuit is met with

success, we could reach out to other educational institutions and see what they think. In addition, we hope to increase the gamified aspects of the site. A points/rewards system as well as possibly a social aspect would open up a whole frontier to be explored with Mezzo, truly bringing musicianship to a wider audience and a fun and accessible manner.

During this whole process, we also had to learn how to code in a group - two brains developing code on the same computer can lead to conflicts of intent or habit. Nevertheless we all got along extremely well (best friends forever!) and learned from each other's coding and working styles, allowing us to fine tune the code of the group as well as our personal styles and working habits as we progressed. We were able to schedule group meeting times effectively and actually had a *lot* of fun coding together, sometimes at the expense of efficiency; regardless, we learned a lot about each other and life in general along the way.

From our first meeting between four strangers on Piazza to our demonstration day and on to Dean's Date, both our group and our project have grown immensely. It has been a wild ride, filled with too many bugs, late nights, and perhaps more caffeine than our bodies know what to do with - but also a huge amount of learning, friendship, and a web app we are all proud to say we made ourselves. They say programming is like an artistic process - as four musicians/programmers, we can straddle that divide and say that Mezzo truly is like a work of art in progress. We've accomplished much in the past few weeks, and can't wait to see where the future might lie for us and for Mezzo.