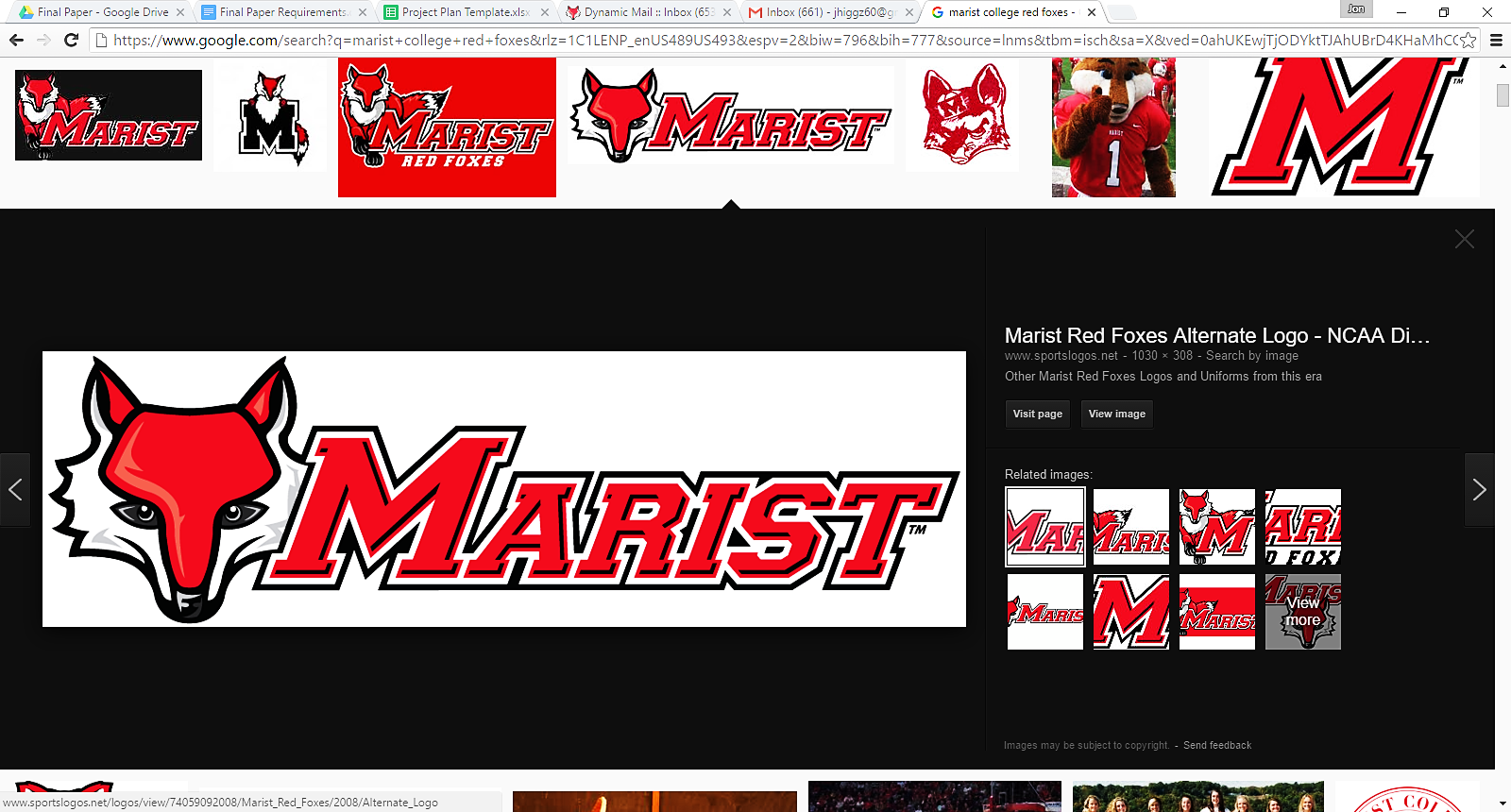
|  |  |
| --- | --- |
|  | **2015** |
|  | Team HERBY  Kevin Bruce, Jaime Engl, Jonathan Higgins, Charles Ropes, and Weon Yuan |

|  |
| --- |
| **[Foxify]** |
| Foxify is the Capstone project that Team HERBY has designed based on the requirements set forth by Chris Doyle and the Overseeing professors. This is a Program designed to make transferring to Marist College easier for the prospective students and Admissions. With course mapping and the calculation of credit hours needed for a selected major ,‘Foxify’ will really cut down on look-up times for the admissions staff allowing them to do more look-ups and be more productive. |



Team Herby

Professor Algozzine

CMPT475/477

CS/IT/IS Capstone Project

Credit Checker

# Foxify

## Introduction:

As a group we decided to name our team ‘Herby’ because it consists of the first letter from each of our last names. Our team consists or three computer science students: Charlie ropes, Jaime Engl, and Weon Yuan. We have one Information Technology major: Kevin Bruce and one information systems major: Jon Higgins. We decided to name our credit checker capstone project ‘Foxify’ because it is a transfer student credit checker to help prospective students turn into red foxes. In a sense out team is creating an end product to Foxify prospective students.

We split up the project into different roles that each person has to do to participate and to do their part in making this project come to life. Weon worked on the front end coding the graphical user interface with node.js and angular. Jamie and Charlie worked on the database and the API which communicates with and connects the front end to the backend. Kevin set up the servers and installed the operating system on them. Jon Higgins documented the weekly status reports and helped all around when needed whether it was helping with code, meeting with professors, doing research on node.js and API’s, and Linux command line commands.

## Executive summary:

Foxify is a quick and easy way to do class look-ups for prospective transfer students coming from Dutchess Community College. Students can enter their classes that they have taken at Dutchess and it will automatically show them how many credits will transfer and be accepted at Marist. They can also select what major or minor that they are interested in at Marist and it will show them what percentage they have already fulfilled with their already earned credits. This is a much more in depth report for interested students so they know what exactly they will need if they attend Marist. This will be a very fast and easy way for the admissions office to evaluate incoming transfer students instead of having to manually look up one class at a time. This will save a lot of time on both ends of the spectrum, students and Admissions office staff. Later on down the road our end product is scalable and can be updated and expanded with no problem to include more schools. In the end, more credit, look-ups can be done on a yearly basis making it easier for more prospective students to attend Marist.

### User requirements:

Throughout the meeting with the client, and the client interview our group put together an extensive set of user requirements. We selected what we thought pertained the most to what the client wanted in an end product. We did have to condense it a little because there was talk about future uses and adjustments, but we created this list based on the need of the client and excluded what we thought to be scope creep.

**1.0.** User Interface

**1.0.1** GUI should follow standards of the physical appearance of other Marist web applications.

**1.1** Web Portal for Students

**1.1.1** The client is looking for a system that allows prospective transfer students the ability to compare the credits they have already taken at their current college to the classes needed for a specific major at Marist. Prospective Transfer students can only transfer in with one prospective major.

**1.1.2** The tool will generate reports detailing how many credits transfer over and how many credits remain in order to graduate in the major they have selected.

**1.1.3** The application will automatically let the prospective student know if they are already close to attaining a minor.(i.e. An alert that states: ”You are 75% done with ‘Minor’ already.”

**1.1.4.A**. Prospective Transfer students do not need a pathway if they have over 36 Credits.

**1.1.4.B.** Prospective Transfer students need 24 credits to not have to do a Freshman Forum.

**1.1.4.C.** Prospective Transfer students can only transfer in a maximum of 70 credits up to level 200 classes.

**1.1.5** The system should be capable to save the classes that they have inputted under their current session, either by having the student create an account or another method.

**1.1.6** Students would select classes from a previously set list specific to their school.

**1.1.7** If one of their classes is not an option the student should be able to submit a request to add their class via another form to be consulted by the Admissions staff.

**1.1.8** Students seeking information, which may or may not be present in the system, can request to contact the Admissions staff.

**1.1.9** The tool would also record user information such as name, email, major of interest, current college, and semester they would intend to start at Marist which would be available to the Admissions staff.

**1.2** Administrative Portal for Marist Admissions. 7 Admission Counselors and 10 top people that would need access.

**1.2.1** The Admissions staff would be able to register an account in order to update program requirements and course equivalencies as needed.

**1.2.2** The application should allow the Admissions staff to respond to open support tickets via e-mail (from students).

**1.2.3** The Admissions staff should be able to review form submissions from students and have the ability to approve/reject the submissions.

**1.2.4** The Admissions staff should be able to review partnership requests from Affiliate Admissions and have the ability to approve/reject the requests.

**1.3** Administrative Portal for Affiliate Admissions.

**1.3.1** The application should allow for other colleges and universities to apply for partnership to become an affiliate in order to be added to the tool.

**1.3.2** This system will allow for Marist to develop close partnerships with numerous two-year colleges as it serves an efficient and convenient way for students looking to transfer.

## IT Requirements:

Based off the list we deducted for the user requirements our team created a list of what we needed for the IT requirements that could create and host this system based on the client’s needs.

1. **Server Platform (for each “server” required)**

We decided to go with Linux Ubuntu 14.04 LTS as the server operation system as it would work better with the database and what we are using on the front end.

* 1. **Physical system requirements**

Our images will easily fit within a single physical Server, but will need to think about physical equipment requirements for the final version after the prototype is accepted.

* + 1. **Storage capacity**

1. Maximum amount of Memory - 4GB
2. After meeting up with Chris and deploying the Virtual Servers, he explained to me that since this is a Virtual environment and not the real world, we shouldn’t worry about too much memory. (App/Database/Backup - Roughly 4GB of storage memory each)

**Speed requirements / response time parameters**

Reducing Traffic – Improves Server Response Time

1. Our Servers runs in a Virtual environment (VMware VSphere Client). Which is also running on a physical server downstairs of Hancock.
   1. **Virtual system requirement**

Application/Database Server - Virtual Linux Machine (Ubuntu 14.04 LTS) 4GB memory

Backup Server - Virtual Linux Machine (Ubuntu 14.04 LTS) 4GB memory

* + 1. **Number of images expected**

Our thoughts on the Virtual Images

5 in total. 2 for the Web Server, 2 for the Application Server, 1 Backup Server

More than 1 Server for each mainly for redundancy - only 1 Server for Backup.

After meeting with Chris we came to a conclusion that 1 image for each Virtual Server is all we need. I will set up the Backup Server to take redundant snapshots. These snapshots will be taken daily, weekly and monthly at specific time frames and stored in a backup folder on the server.

* 1. **Connectivity**
     1. **Network considerations**

The Server Platform should be able to communicate with any TCP/IP connected devices using the Network Requirements that I will set up during the deployment of the server.

1. After the Network Requirements are in place we will be able to access the Servers even through third party software such as Putty. Putty is a program that helps a user get access to an operating system (OS) they created through a command-line interface terminal.
   * 1. **Interconnection to what other systems**
2. The benefits are broad in scope. The IP addresses can be used since we wish to connect the Application to the Database Server, as well as accessing the Application Server from the web. And to store snapshots of the App/Database to a backup folder the Backup Server. This is why the Network Requirements are crucial to this entire project.
3. **Reliability**
   1. **Service Level Agreements**
4. Redundancy and Multi-site (Servers – Virtual)
5. 1 Virtual to host the Application/Database Server
6. 1 Database Virtual Server - Redundant Snapshots

**2.2. Uptime requirements**

1. We didn’t exactly know how big this project was going to be. But if we applied uptime requirements to the real world, the servers should be up and running majority of the time. There should and will be maintenance hours for specific time frames.
   * 1. **Response time requirements**

1. Expectation - Less than a second response time for user satisfaction while browsing.

2. After a few weeks into the projects issues arose regarding the database server. There were a lot of problems that we tackled but we weren’t successful at resolving the issue. The team ended up agreeing to host both Application and Database on the same server. Great things ended up happening like response time, and less work to get the app to talk with the database.

1. **Recoverability**
   1. Where are things backed up? How often?

Application/Database Admin (IT) – Data will be frequently backed up to Backup Server

* + - 1. Snapshots - The Backup Server will be configured to take snapshots which will be saved in a folder called backup.
  1. **Access to backups?**

(IT major) – Anyone can access the backups and view the backups, but I will have full admin rights to the backups. It is my job to see that there is that the server are back and running again when something goes wrong.

* 1. **What data is transient and doesn’t need to be stored longer term?**

(Student Data) – Potential transfers will be deleted from the database once the admission process is over. After their final decision to transfer to Marist or not. This will make space for potential students for the next term.

1. **Security and Privacy**
   1. **Database**

Who should have admin access?

7 admins to reply to emails/ students questions and submissions

1 important admin that overlooks everything at admissions

* + 1. **Access controls by userid / roles**

1. Are we going to use LDAP? Lightweight Directory Access Protocol

(LDAP servers can look up entries in a wide variety of ways. LDAP servers index all the data in their entries, and "filters" may be used to select just the person or group you want, and return just the information you want.) - Yes we can consider implementing this protocol into our database.

2. What code will run on what server to let users create accounts and login? (Apache, Mysql)

* 1. **Update vs. Access**

Roles of users and what will each be able to perform? Edit, update, read only, reports, database admin, etc.

* + - 1. Admins – edit, update, (1 database admin)
      2. Super users / Admission users – answer questions, lookups, read only
      3. Students – submit questions / enter transferable credits for feedback
  1. **Account information**
     1. **User data**

Personal / registration

User data will be kept in the database and retrieved when needed.

* + - 1. **Saved courses information**
    1. **FERPA considerations**

Student information will be stored since this is just a prototype, but student sensitive data will have to be addressed before the prototype can become a production system.

* 1. **Admin access controls**

Same as FERPA considerations.

* + - 1. What data should various admins and super users have access to?

1. Amins - edit, update, (1 database admin)
2. Super users / Admission users - answer questions, lookups, read only
   * 1. **Adding new users, deleting old**

IT System Admin will use (Active Directory) as a tool to add and delete users

1. **Maintenance** 
   1. **Planned down time requirements**

How often will the system be down for maintenance?

* + - 1. Potentially it will probably be down from midnight to 1 am nightly window. This will Improving user satisfaction/ eliminating frequent unexpected down times while a student is browsing the web.
      2. It only applies to the servers that need to. Let’s say backup failure for instance.

**What happens if a CS member messes with the some of the code and happens to cause down time?**

* + - 1. This issue will be looked at during the downtime window.

Also a CS member can notify the IT guy downstairs when they want a downtime to polish up some code that will cause some issues. This shouldn’t happen frequently but is acceptable. (This will be a service level agreement with joint study lab owner – Chris Ahmed/planned outages).

* + - 1. Professor Algozzine will be on the list so we can give him heads up not to mess with the prototype at this time frame.
    1. **Database maintenance**

Database Admin - will provide technical support for the Database maintenance and disaster recovery. Available during school hours.

* + 1. **Updates to course information**

This will be done by the Database Admin

* + 1. **Times of year when IT does maintenance**

1. IT Maintenance Plan - will be designed to save us time and money by preventing system downtime.

2. IT staff (Joint Study / Chris Ahmed) will be available on school hours to provide support /repair if something unexpectedly breaks down.

* + 1. **Times of year when Admissions systems are not available?**
       1. Once a semester for maintenance
       2. We should work out a time frame that's convenient for Admissions.

## Project plan:

Team Herby created a project plan that listed dates and milestones where the group decided on realistic goals to achieve week by week. We were very strict in following this and it came out very well. There were a couple of set-backs when we had a connectivity issue with the API, but we quickly figured it out by seeking help from Professors and colleagues and we were able to quickly steer back on track. Below is the Project plan where the group documented our goals and our progress throughout the course of this semester.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Milestone** | **Tasks** | **Responsible** | **Notes** |
| 8/31/2015 | Project Start | Form team: make introductions; inventory skill strengths and weaknesses; formulate and document initial roles and responsibilities (to be modified as needed later). | All |  |
| 8/31/2015 | Project Start | Schedule 1st team meeting - and attempt to choose recurring team meeting times that work for everyone (can be established later); establish a communications plan (ie. share emails, cell#'s, and establish how you will stay in touch) | All |  |
| 8/31/2015 | Project Start | Further develop this project plan (it is part of HW1) - THINK through your time management and what you will accomplish as the weeks progress. | All |  |
| 9/1/2015 | *Team Meeting* | *First Team Meeting. Break down Homework 1 into 4 manageable sections. As a group decide which questions need to be asked of the client at the next class. Think about project from all 3 majors point of view. Review project description and decide the user requirements. Compile into an executive summary which can be edited after meeting with client. Choose next meeting.* | All | Had first team meeting and worked on HW#1 including initial setup of project plan and brainstorming for project. Sent into PM for feedback. |
| 9/4/2015 | *Team Meeting* | *Second Team Meeting. Work on Project Plan to ensure work is completed in timely fashion. Create possible deadlines for future tasks. Create two UMLs based on the two different scenarios discussed previously. IS student creates status update for the week.* | All | Received feedback from PM and created appropriate changes to HW#1 |
| 9/7/2015 | Team Meeting; 1st homework started | Have met at least once by the end of the day on 9/7; have started Homework #1 minimally - add more items to this project plan in the empty lines below and have altered this line to be something more meaningful than my babble… | All |  |
| 9/14/2015 | Homework #1 Due! | Have HW1 done; Have this project plan completely filled in for review and comments back so you will know if you have a good plan to get to end-game;   Have your client questionnaire finalized and be prepared to interview the client. | All |  |
| 9/15/2015 | *Team Meeting; Begin planning ER diagram* | *Discuss what went well with the process for Homework 1. Discuss the answers that the client gave us from class and adjust user requirements, UML documents, and project plan as needed. Begin thinking about which tables need to be created for database.* | CS (creators) IT and IS (contributors) | Decided not to meet on this day and just meet on Friday 9/18. |
| 9/18/2015 | *Team Meeting; Work on Homework #2* | *Finalize database tables. Create Entity-Relationship diagram. Discuss different design ideas and choose the most effective one. Develop documentation for the ER diagram. IS student creates status update for the week.* | CS (creators) IT and IS (contributors) | Discussed what went well on Homework 1 (completed all goals during each meeting, had good communication with project manager). Started on homework #2 by creating the ERD. Sent to Project Manager for review. Edited and revised Hw#1. Created weekly status update. |
| 9/21/2015 | Homework #2 Due | E-R diagrams complete with supporting documentation if explanation is required for anything you document in your diagrams. | All | Homework #2 given an extension by Project Manager to Friday 9/25. Homework #3 given extension to October 25th. Addressed credit hours issue. Decided on goals for 9/22 meeting. |
| 9/22/2015 | *Team Meeting; Begin working on database design* | *Discuss what went well with the process for Homework 2. Update previous documents as needed. Discuss design for GUI/interface. Review design principles. CS students begin work on prototype of database.* | IT and IS (Wire-framing) CS (database) | Updated Project Plan. Finished updating user requirements. Created additional document describing ER diagram. Finalized ERD. IT student began working on IT Plan. Going to start on Homework #3 on Friday 9/25. |
| 9/25/2015 | *Team Meeting; Work on Homework #3* | *Create 5 wireframes/mockups that depict design discussed in previous team meeting. Confirm with all team members this is the best design. CS students continue work on prototype of database. IS student creates status update for the week.* | IT and IS (Wire framing) CS (database) | Wrote create statements for tables in Database prototype. Started wireframe mockups. Continued work on IT plan. Updated Project Plan. Edited ERD. |
| 9/28/2015 | Homework #3 Due | Complete mock-ups (ie. wireframes) of your user interface. | All | Just to reiterate this due date has been pushed back to Friday 10/2. In addition Project Manager informed us that Major Concentrations are out of scope. |
| 9/29/2015 | *Team Meeting* | *Discuss what went well with the process for Homework 3. Update previous documents as needed. CS students continue work on prototype of databases. Confirm security permissions. IT and IS students finalize UML diagrams.* | Mostly CS students IT and IS (UML) | We as a group decided CAS was out of scope but will document it as a future opportunity. Finished wireframes. Updated project plan. Came up with activities and sent to Project Manager. |
| 10/2/2015 | *Team Meeting* | *NEW: HOMEWORK 3 DUE DATE. CS students continue work on prototype of database. Meet with SME as needed. Begin creating documentation for the database. IS student creates status update for the week.* | Mostly CS students | Finished up final wireframes after feedback from Project Manager. Decided on plan for next week. Updated Project Plan and status report. |
| 10/5/2015 | UML dialogs final; project plan final | Final Project Plan - updated with all input from instructor Final UML Diagrams | All | Checked in PM to make sure project is going well. |
| 10/6/2015 | *Team Meeting* | *CS students continue work on prototype of database. Discuss stored procedures and whether or not they will be needed. Continue documenting.* | Mostly CS students | Set of local copy of postgres and ran create statements to make sure they work. Worked on GUI design on paper. Jon worked on ROI calculations. |
| 10/9/2015 | *Team Meeting* | *CS students continue work on prototype database. CS students begin work on GUI. IS and IT students help debug as needed and work on final paper. IS student creates status update for the week.* | All | Started researching new technologies such as Node.js and Express.js. Finalized some design details and updated ER diagram. |
| 10/12/2015 | Database design draft | Draft database design | CS Students | Checked in PM to make sure project in progressing accordingly. |
| 10/13/2015 | *Team Meeting* | *CS students are still working on database and GUI. IS/IT students review work of CS students and continue paper.* | All | Kevin created 3 servers, 1 for database, 1 for application, and 1 for backup. Decided on majors that are going to be used for testing and collected data accordingly. Assigned "fake" credit hours to data. |
| 10/16/2015 | *Team Meeting* | *CS students are still working on database and GUI. IS/IT students review work of CS students and continue paper. IS student creates status update for the week. Discuss what has gone well and what hasn't gone well as a team so far. Address issues as needed before peer reviews.* | All | Did not meet. Mid-semester break. Weon worked on GUI. Kevin continued work on servers. |
| 10/19/2015 | Mid-semester peer reviews due | Complete initial (ie. first pass feedback) peer reviews | All | Checked in PM to make sure everything is going well. Asked for Kevin to send IT documents for initial review. |
| 10/20/2015 | *Team Meeting* | *CS students still work on database and GUI. IS/IT students try using GUI and see what needs to be changed.* | All | Cancelled meeting because still waiting on client access to servers. |
| 10/23/2015 | *Team Meeting* | CS students still work on database and GUI. IS/IT students try using GUI and see what needs to be changed for easier usability. Discuss disabilities that might need special consideration for GUI such as colorblindness. IS student creates status update for the week. | All | Charlie worked on selecting and cleaning up the small portion of data we will need for testing and creating csv files. Jaime looked into postgres commands for the server to create the database and insert data. Weon continued work on GUI. Kevin working on getting NGINX onto the application server. Jon working up uploading database dump file to server. Updated documentation. |
| 10/26/2015 | Draft IT requirements, including network design | IT student discusses IT needs with CS and IS students. Begin drafting IT plan. | Mostly IT student |  |
| 10/26/2015 | *Team Meeting* | *CS students complete initial database prototype and meet with SME to discuss. CS students also continue work on GUI.IT student continues work on IT plan. IS student continues work on paper.* | All | Jaime and Charlies created database and tables on database server. Kevin showed everyone how he setup servers. Weon continued work on GUI. Jon began research on Node.js API. |
| 10/30/2015 | *Team Meeting* | CS students finish database prototype and documentation. IT student continues work on IT plan. IS student continues work on paper and status update for the week. | All | Jaime and Charlie created SQL files to load data into database. Kevin changed backup server into test server to test API on. Weon continued work on GUI. Jon continued learning Node js. |
| 11/2/2015 | Homework #4 Due | Database prototype complete | CS students |  |
| 11/2/2015 | Team Meeting | CS students finish work on GUI to review with whole group. IT student finished IT plan. | All | Database has enough test data in order to do simple functionality. Jon, Jaime, Charlie, and Kevin continue work on API. |
| 11/4/2015 | Team Meeting | Create Node js API. | All | Had an additional meeting because we are having trouble creating API. Tried to connect to server but had a lot of errors. Weon continued working on GUI. |
| 11/6/2015 | Team Meeting | Group meets to practice demo. | All | Still working on API and still having trouble. We are pushing off practicing demo to Sunday. |
| 11/8/2015 | Team Meeting | Finish API and practice demo. | All | Trying to get API working and practice demoing. |
| 11/9/2015 | First Demo of Prototype | Complete project prototype first-pass demo ready | All |  |
| 11/9/2015 | Team Meeting | Continue Node.js API work. | All | Met with SMEs to try and get API working. Server having issues. Weon still worked on front end. Jaime and Charlie continued the API. Kevin and Jon worked on fixing up documentation. |
| 11/13/2015 | Team Meeting | Try and get Node.js API working. | All | Meeting cancelled because of school lockdown. Continued individual work. Decided to get rid of database server in order to resolve connection issues with API and server. |
| 11/16/2015 | User Validation & Test plan finalized | Test plan for all aspects of the prototype complete User validation tests to be performed by client documented |  |  |
| 11/16/2015 | Team Meeting | Test API | All | Finally got API working on application server when database was moved to that server. Jaime and Charlie sat down with Weon to go over all queries that need to be made in order for front end to function correctly. Jon and Kevin worked on documentation and setting up paper. Weon changed framework on front end from React to angular. |
| 11/20/2015 | Team Meeting | Connect Database and Front End | All | Charlie continued writing post and gets for API. Weon worked on making sessions restorable. Jon organized all current documentation for final paper. Kevin is still working on documentation. Jaime updated ERD and ERD documentation along with project plan. |
| 11/23/2015 | Client Visit | Updates to user validation complete (as required) One key question to ask the client documented |  | Cancelled. No meetings or class during Thanksgiving week. |
| 11/30/2015 | Draft Documentation | All documentation required for the project in Draft form |  |  |
| 11/20/2015 | Team Meeting | API, Front end, and Paper work | All | Jon and Kevin worked on paper and documentation. Charlie and Jaime continued working on API and cleaning up the database. Weon worked on session storage, saving it and returning to it. |
| 12/4/2015 | Team Meeting | API, Front end, and Paper work. | All | Jon is still working on the paper. Kevin updated IT requirements. Weon worked on connected the front end and the back end. Charlie is still working on API JSON posts. Jaime changed some SQL queries in the API and the database structure slightly removing one column from Admins table. |
| 12/7/2015 | Paper Due - Final Documentation | All documentation required for the project in FINAL form |  |  |
| 12/7/2015 | Final Peer Evaluations | Complete final peer evaluation forms and submit | All |  |
| 12/14/2015 | FINAL PRESENTATION! | Git er dun! | All |  |

## Weekly Status reports:

Based off of the set project plan we would closely follow our set mile-stones and goals. Each week we would document our progress and any changes or challenges that we faced. We would bring our weekly status reports to the project director (Our Capping Professor, Professor Algozzine) each weekly meeting to report our Progress, set-backs, and questions or concerns. Seen below if you compare the Project plan to the weekly status reports you can see that we worked very hard to stay on track and for the most part we hit every goal in a timely fashion compared to the milestone date. There were a few small issues that we ran into along the way and some pieces that took a little longer than predicted, but we met with various professors and authorities to receive help and we did as much as possible to keep up with deadlines .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RGY** | **Item** | **Description** | **Mitigation / Help needed** | **Date** |
| Green | Homework #1 | Created Group. We created 2 UML's and filled out the Project plan and came up with questions. We also clearly documented the set of user requirements based on the information that we have received. |  | 9/4/2015 |
| Green | Homework #1 | Received feedback from Professor Algozzine. Met on 9/8 and 9/11 to edit our work and submit our project. |  | 9/7-9/12 |
| Green | User requirements | Met with the Client and asked questions. Then on 9/15 we updated our user requirements. |  | 9/14 and 9/15 |
| Green | Homework 2, rough draft and weekly progress report due | We finished the rough Draft of our ERD, and submitted our weekly progress report. | We need help in getting and understanding exact credit hours for each class. We believe that it will be a huge issue in transferring credit hours. Also, How will the majors table transfer and separate courses into core and elective credits. (Answered/Fixed) | 9/21/2015 |
| Green | Homework #2 | Wrapped up Homework #2. Finished up Homework #2. We submitted the work and worked on the wireframe for homework #3. |  | 9/25/2015 |
| Green | Homework #3 | We are continuously working on HW#3 (due October 2nd) and we have even started working on some of HW#4. The wire framing is going well and we are creating a very aesthetically and ergonomically pleasing UI for the user. | Would you like any documentation for the wireframes? If so, how would you like it presented? (answered) | 9/28/2015 (Class) |
| Green | Activity Diagram | We finished everything on HW#3, but we could not complete activity diagram because Project Manager never contacted us about which one to do. | We are waiting on the Project Director to tell us what Activity Diagram to create.(Answered) | 10/2/2015 |
| Green | Homework #4 | We have started HW#4 and work is under way. We are continuously having our meetings and doing work. IT launched a windows virtual server to begin construction of the DB. As our team still waits to hear from the P.D. we have discusses how we would create many on the different activities on an activity Diagram, but we just need the go ahead on which A.D. to create. | Currently still waiting on the word from the Project Director for the go ahead on which Activity Diagram to create.(Answered/fixed) | 10/5/2015 |
| Green | Homework #4 | We have been working on the database and we have also been cleaning, and fudging the data a little. We have also been working on the GUI. | We are having issues with the data being terrible, outdated, and it not making sense, so we are fudging it a little because our client isn't able to give us more accurate/cleaner data. | 10/9/2015 |
| Green | Postgress | We keep working on the database and started working on the front end and back end of the program by downloaded node and express which is node’s framework. |  | 10/19/2015 |
| Green | Foxify is growing | The 3 servers are up! We have the DB sever, the front end (application server) and the backup server. We have downloaded node.js and express.js on the application server. we are also creating a dump file of the DB to throw into the DB server. | We are waiting for a response from the project director in regards to pathways. We are having trouble using a dump file to transfer our database, so we have to type in our database into the DB server to recreate it.(Resolved) | 10/23/2015 |
| Green | More Progress | We have continually been working on the front end and back end. The API has been very time consuming getting it to connect the front end and back end together. | We need help with the API. We have sought the help of other groups and the project director. We are continually working on everything, but we have shifted more of the focus on the API. | 11/2/2015 |
| Green | Presentation | We have made a lot of progress on both the front end and back end. The GUI is coming together very nicely and is mobile device compatible. The backend and DB are looking very god as well. | After seeking help of various professors, students, and other outside sources we are still having a difficult time with our API. The code seems to be fine it’s just there is something not letting the API make the connection. We have tried many different things to get it to work, but still no connection. We are planning to meet with Chris Ahmed this week to see if he can help us solve our connection issues. | 11/9/2015 |
| Green | Meeting and presentation | We did not meet on Friday because we decided to cancel our meeting as the campus was on lock-down. Although we did not meet we have made progress with connection issues between the front end and back end with the API. We met with Chris Ahmed to help us fix some of the connection issues and also we sought help from Alan. | We have made an executive decision to put the API and the DB on the Application server. We started fresh and re-installed postgres, because we believe that our issues stemmed from not properly installing postgres on the DB server initially to meet our needs. Between the help from Chris Ahmed and Alan we have learned a lot and made progress with the connection. We have also had a great break through with the front end by not using react and anything react related. The framework was very time consuming to learn and was confusing. Instead we switched to angular. This will make course selection and filtration easier. | 11/16/2015 |
| Green | no meeting with group or professor | After meeting a couple times in the previous week we decided not to meet on 11/23 because a few of the group members would already be traveling. We assigned roles for each person to work on during the week of vacation and we constantly talked and checked on each other’s status. | Our team telecommuted and discussed progress and each other’s assignments and goals for the week of break. Everyone made progress on what we were all working on and assigned. The week of the break was not a setback or a loss of time at all. Jon worked on the final paper. Kevin, worked on solidifying and editing the documentation for the paper. Jaime, worked on the database and some DB documentation. Charlie, worked on the API and the DB.Weon, kept on working on the front end and touched up on some documentation as well. | **11/23/2015** |
| Green | Meeting with professor, status report and presentation or current work. Team meeting. | We are continually working on everything as a group and we are meeting to show what we all worked on over break and to move forward. As we get closer to the wire we are all pushing forward to finishing up and testing. | With much progress made over break we are meeting to help each other out and to move forward to finishing up and then onto testing while we finish up the final documentation and brushing up the paper. | **11/30/2015** |
| Green | Final Status update and Demo with no dry run of the final presentation. | As a group we have made a lot of progress in every realm of the project. We are wrapping up the Final paper, finishing up the logic with the call functions with the API. Connected to the front end and back end through the API. We are also putting finishing touched on the GUI to make it look as pretty as possible. | As a group we are sprinting to the finish putting finishing touched on the Paper and project. The DB is looking good and the call functions are working in the API connecting the front end to the back end to complete the course look-ups that are requested. The paper is coming together very well as we update and correct some of the documentation. Also, the GUI is starting to look beautiful as we make some last tweaks! | **12/7/2015** |
| Green | Final Presentation | We wrapped coding the API, GUI, and DB. Finished the Paper and documentation. Created our slides and went over presentation. | Ready to present Foxify! | **12/13/2015** |

## Cost/ROI:

**Preface-**

Throughout tracking the return on investment on this project our group has concluded that it is advantageous and profitable for Marist to back the development of this project. There are many benefits for Marist College that come from this Capstone project course.

The benefits about having this course mandatory for seniors to graduate include the following: Marist College is receiving payment for credits by students to develop this application. A Marist student pays $634 per credit hour and this is a four credit course. There are 26 people in our section and our group is assuming that the 3 sections are broken up evenly.

$634 per credit hr x 4 credits x 26 people in a section x 3 sections = **$197,808**

Marist College is roughly receiving **$197,808** from this course and receiving an application that they need to make admissions more productive when it comes to looking up transfer student credits.

**Transfer Course Look-ups-**

Currently, admissions has 5 personnel that perform 10 credit reviews each for DCC, and 90 reviews each from other colleges for a total of 100 reviews per person. Each of these reviews takes 30 minutes each.

5 people x 90 reviews + 10 DCC reviews x 30 minutes (.5 hours) = **250 hours** of time spent on look-ups.

If these reviews were not being done manually, then they could potentially be able to do 250 look-ups for DCC and 2500 reviews for other colleges a year.

250 DCC look-ups + 2500 other college look-ups = **2275 look-ups**.

This beats the:

5 people x 10 DCC look-ups + 90 look-ups = **500 look-ups** that need to be done by hand and it saves

2275 look-ups x 30 mins = **1137.5 hours** saved with an increase in the amount of look-ups by 4.5 times.

2275 number of look-ups / 500 previous number of look-ups = **4.5 times**

**Hours of production-**

Here is the production cost measured in hours that we put in of our time to produce this product:

Each class is 2.5 hours and our group meets 2 times a week for roughly 1.5 hours and we have 16 weeks/ class meetings.

2.5 hours a class x 2 meetings a week x 1.5 hours x 16 weeks x 5 people in our group = **600 hours** total that our group will spend on this project. This does not include the amount of time that we work on this project outside of class and during our meeting. Also, this doesn’t take into account if someone misses a meeting or class.

**ROI-**

ROI(%) = (Gain-Investment)\*100 / Investment

(1137.5 saved hours – 600 hours) x 100 / 600 = **89.58%**

**BEP-**

The BEP is on the 300th look-up.

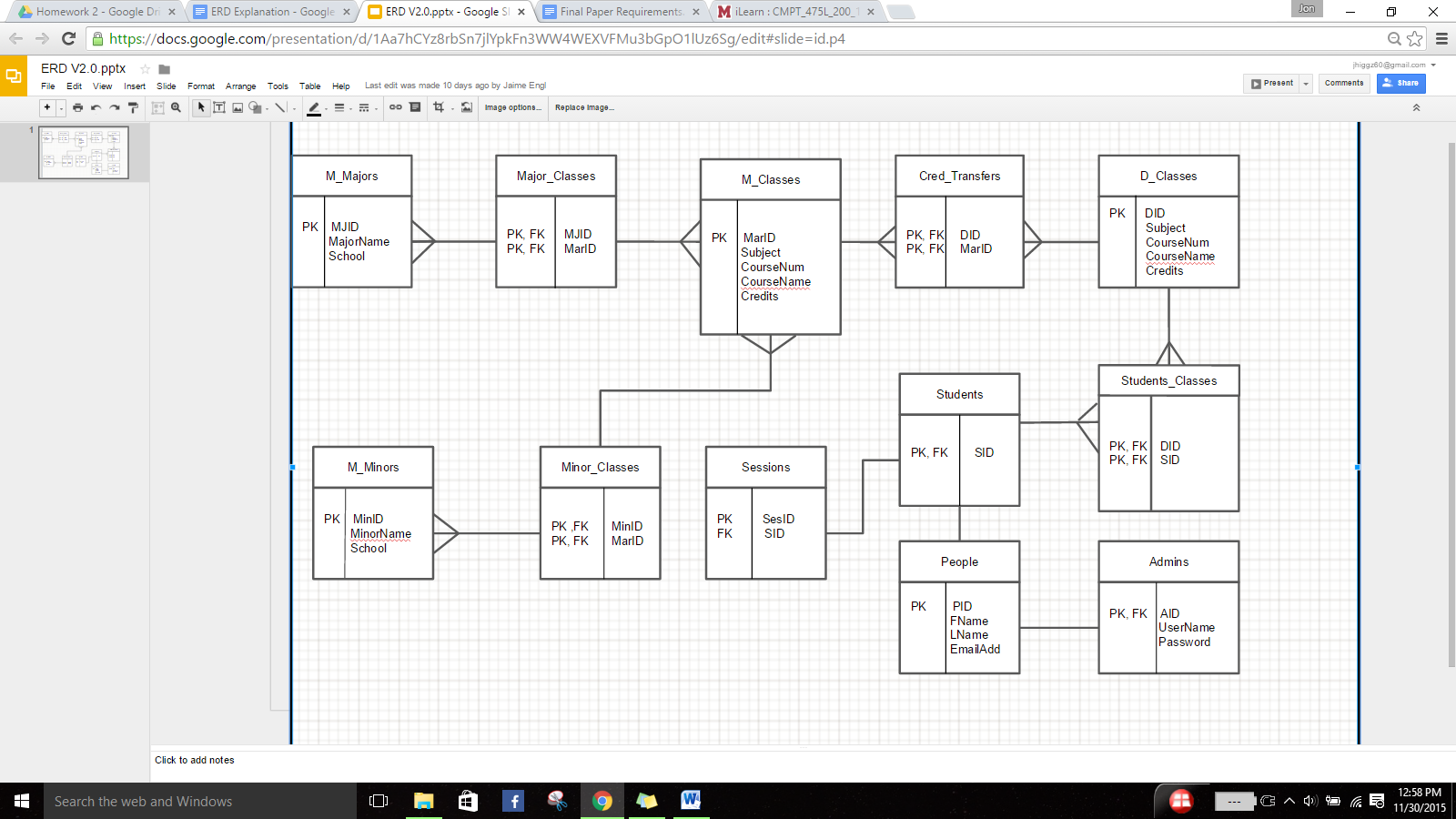
600 hours of development x .5 hours saved per look-up = **300**

And since there is an estimate that there will be 2275 look-ups a year with the implementation of the application then 300 lookups (ROI) / 2275 projected look-ups = **13%**

ROI Conclusion-

I conclude that this application will be a huge benefit for the efficiency to the people in admissions to do class look-ups for prospective transfer students. Marist is guaranteed to get their ROI and BEP in the first year. Marist also is not really investing anything into the project by having a mandatory course where student create the end result. Marist is actually turning a profit with minimal investment because they are receiving money from students to take this course and the College is benefiting from the free labor. In the end Marist is receiving roughly **$197,808** off of this Capstone Project Course and by having students develop this application Marist will be saving **1137.5 hours** of work by the admissions staff and the College will gain **2275 look-ups** of interested transfer students a year.

## Application Design: ERD



**People Table**

The ***People*** table will hold basic information such as First and Last names, Email, and a PID to identify a user. It will have a one to one relationship with the ***Students*** and ***Admins*** tables.

**Admins Table**

The ***Admins*** table will hold information about the admins. It will store their Username, Password and an AID which will be taken from the ***People’s*** PID.

**Students Table**

The ***Students*** table will store only the Birthday of the student to be used to determine if prospective students will be considered a regular student or an adult student. It will also store a SID which will be taken from the ***People’s*** PID.

**Sessions Table**

In our design, we are going to have session that saves a randomly generated number with a student’s ID so they can log back into a session without losing all of the work they have done so far. The **Sessions Table** stores that generated number with the student’s email address. This table has a one to one relationship with ***Students***.

**Student\_Classes Table**

The ***Student\_Classes*** table will hold the information connecting a student to a class. This is because many classes can have many students and many student can have many classes.  This table will store the DID (Dutchess ID) with the SID (Student ID).

**D\_Classes Table**

The ***D\_Classes*** table will hold all information about Dutchess Community College courses. Subject, Credits, Course number, and Course name will all be stored along with a DID to identify classes. It will have a one to many relationship with ***Has\_Taken*** and ***Cred\_Transfers***.

**Cred\_Transfers Table**

The ***Cred\_Transfers*** tablewill hold all the information connecting a Marist class to a Dutchess Class. This will match the classes that transfer from Dutchess to Marist. This table will store DID (Dutchess ID) and MarID (Marist ID). It will act as a composite table since it has a many to one relationship with ***M\_Classes*** and ***D\_Classes***.

**M\_Classes Table**

The ***M\_Classes*** table will hold all information about Marist College courses. Subject, Course number, Credits, and Course name will all be stored along with a MarID to identify classes. It will have a one to many relationship ***Cred\_Transfers***. It also has a many to one relationship with ***M\_Minors Table*** and ***Major\_Classes Table.*** This means that a Marist course can count towards several different Majors and Minors.

**M\_Minors Table**

The ***M\_Minors Table*** will hold information about Minors at Marist. It will store the Minor name and the school the Minor belongs to. It will use a MinID to identify Minors. This table will have a many to one relationship with ***Minor\_Classes*** because a class can be part of several Minors.

**Minor\_Classes Table**

The ***Minor\_Classes*** tablewill hold information about classes that count towards specific Minors at Marist. It uses a composite key of MarID and MinID from the ***M\_Minors*** table to the ***M\_Classes*** table. The ***M\_Classes*** table has a one to many relationship with the ***M\_Minors*** table and the ***M\_Classes*** table this is because many classes can have many minors and many minors can have many classes. We need the ***Minor\_Classes*** composite table to denote this since you cannot have a many to many relationship without one.

**M\_Majors Table**

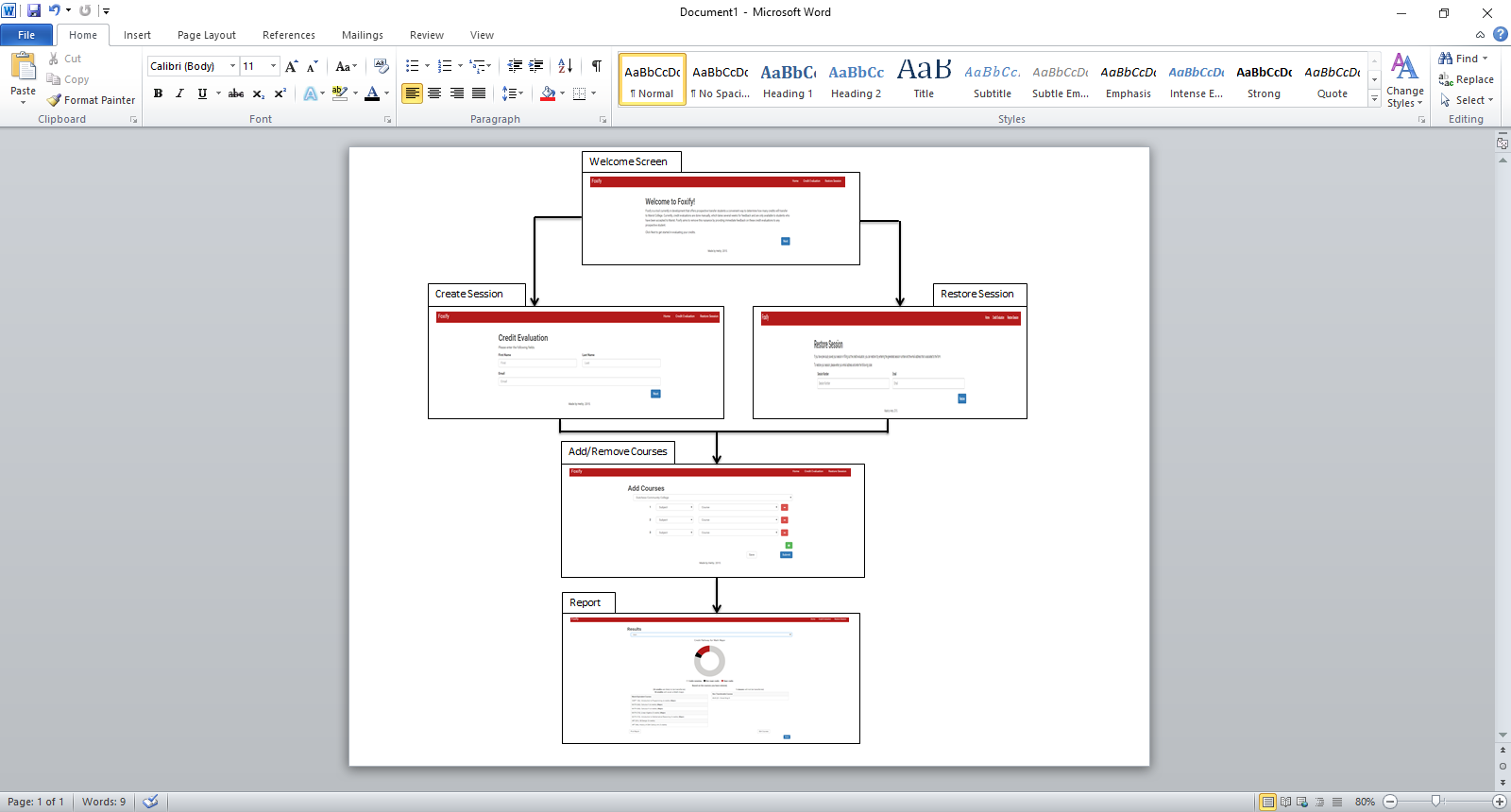
The ***M\_Majors*** table will hold information about Majors at Marist. It will store the Major name and the school the Major belongs to. It will use a MJID to identify Majors. This table will have a many to one relationship with ***Major\_Classes*** because a class can be part of several Majors.

**Major\_Classes Table**

The ***Major\_Classes*** tablewill hold information about classes that count towards specific Majors at Marist. It uses a composite key of MarID and MJID from the ***M\_Majors*** table to the ***M\_Classes*** table. The ***M\_Classes*** table has a one to many relationship with the ***M\_Majors*** table and the ***M\_Classes*** table this is because many classes can have many majors and many majors can have many classes. We need the ***Major\_Classes*** composite table to denote this since you cannot have a many to many relationship without one.

## Application Design: Activity Diagram

Here is our Activity Diagram that visually shows the steps that prospective student will take to use Foxify by either creating or restoring a session, then adding or removing courses, and then they select a major that they are interested in, and finally they will receive a report that shows what credits will count towards the Major, or be an elective, or credits that will not be accepted.



## User Interface Design:

As a group we decided that the Graphical user interface (GUI) should be very simplistic, so that someone who has never used this product before can pick it up in second and not get confused or frustrated. We really spent a lot of time on the Aesthetics and ergonomics to make it simple, easy on the eyes, and easy to use. We kept with the Marist colors and themes, but we didn’t make it look exactly like the Marist site because the prospective student wouldn’t be used to the Marist sites and functionality. We also did not want to bombard them and overwhelm them so that is why we kept it very simple, but yet familiar by using techniques, designs, and functionality that many popular sites use today that people from this generation are used to.

Below we have listed documentation and explanations of what is on each page and what the user can do from page to page. There is also an activity diagram that will show each page and the actions that you can perform and where it will take you. This may be a simpler way to visualize and comprehend actions on this site.

**Prospective Student Login UI**

● This will be the first page Students are directed to after clicking “How will my credits transfer?” on the transfer home page.

● Students will be asked for basic information. First and Last name and their Email.

● They will hit the “Next” button and proceed to the ***Add Classes UI*** page.

**Add Classes UI**

● Students will be taken to this page from the next button on the ***Prospective Student Login UI*** page.

● Students will be asked to provide the school they are transferring credits from and the Major and possibly Minor they plan on looking up.

● Students will add classes by selecting the class subject and the class number for the class at their school.

● The “+” button will add another line for students to add classes.

● Clicking submit will bring the student to the ***Feedback UI*** page

**Feedback UI**

● After a student hits submit on the ***Add Classes UI*** page this is what their report will look like.

● Each class they entered will show the Marist equivalent or the fact that it does not transfer.

● The progress bar shows how much of the major/minor they have selected has already been completed.

**Admin Login UI**

● Admins login using a username and password.

● They will then be brought the the ***Admin Action UI***.

**Admin Action UI**

● Admins will choose either to lookup a course or edit a course.

**Admin Lookup UI**

● Admins will choose from drop down menus of subject and course number.

● Then the classes that map to that course will be listed.

**Admin Edit UI**

● The Admin can lookup a course and make changes to it such as course name, course number, credit hours, and the particular academic school of said course.

## Infrastructure Design:

When it comes down to the infrastructure design we have included explanations of why we chose what we chose and how we set everything up. This portion of the paper also includes how IT configured all of the servers, and the network. If you would like to view all of the code it is posted on our team GitHub: <https://github.com/weonyuan/Herby>

**Why PostgresSQL?**

The team chose to use PostgreSQL for the database for several reasons. PostgreSQL is used industry-wide and is a well-known and documented. It also offers a wide variety of different data types that other databases do not for data such as IP addresses and json. Several members on the teams have worked with PostgreSQL before and were already familiar with its syntax which made for an easier integration with the application.

**Why Angular?**

Angular serves as the front-end framework for Foxify. Angular excels in creating the application more dynamic with its use of two-way data binding, which basically means whatever action the user initiates (i.e., adding a course to the list) the application updates itself immediately without necessarily refreshing the browser.

**Why Bootstrap?**

Bootstrap offers the convenience for us to develop a user interface that is aesthetic, intuitive, and scalable across various devices (e.g., phones, tablets, and computers) under a short development timespan.

**Why Express?**

Express acts as the backend component of Foxify and it provides the flexibility of configuring routes and defining logistics for the application. Additionally, Express serves as a hosting server for our application and our API.

**IT Server Documentation**

**Step 1: Setting up Servers on VCENTER**

Download vSphere Client & Install from a web browser

Use the IP address: 10.10.7.30

Chris Ahmed has the username and password for the Client. He won’t give the credential out to students.

Once we are in, we are able to view/configure servers that have been created.

As of yet nothing has been created.

Following a series of steps we were able to create and deployed a Virtual Linux Machine (Ubuntu 14.04 LTS) as the 1st Server.

Ram: 4GB

**Step 2: Install Prerequisite Packages**

Once the installation of the server is completed, we navigated over and opened up a terminal on the Ubuntu Server.

**The following Commands were issued:**

sudo apt-get update – updates the repositories of the system

sudo apt-get upgrade – installs newer versions packages for the system

sudo apt-get install openssh - install ssh on the Server. This will help putty get access to the server. This will be covered later in the documentation.

sudo apt-get install x11vnc – installs a vnc if we want to ssh to our server with a GUI.

**Step 3: Network Requirements**

Once the system repositories are up to date, with the help of the Update and Upgrade commands, we can go ahead to configure and setup the network requirements for the servers. We assigned an IP address, Subnet mask, gateway, and DNS 1 & 2 to the 1st server that I created.

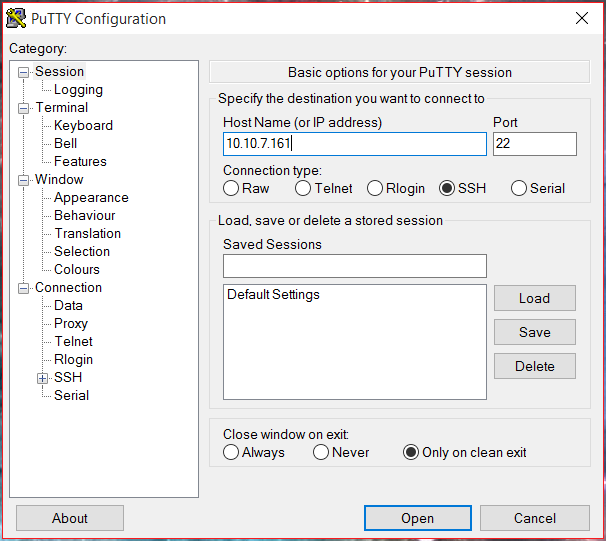
Chris cloned the server each student created into the desired number of server that each group wanted. He also gave each IT student a range of IP address they can use for their servers. Range of IP addresses that were assigned to us is 10.10.7.160 – 165. We only have Two Servers, one for the Database and application.  We chose IP address 10.10.7.161 and 162 and assigned it to the 2 Servers.

After getting the server configured and running, Chris cloned 2 more Servers, an Application server and a backup server. The IP address changes for the 2 new servers, but the rest of the network requirements information stays the same.

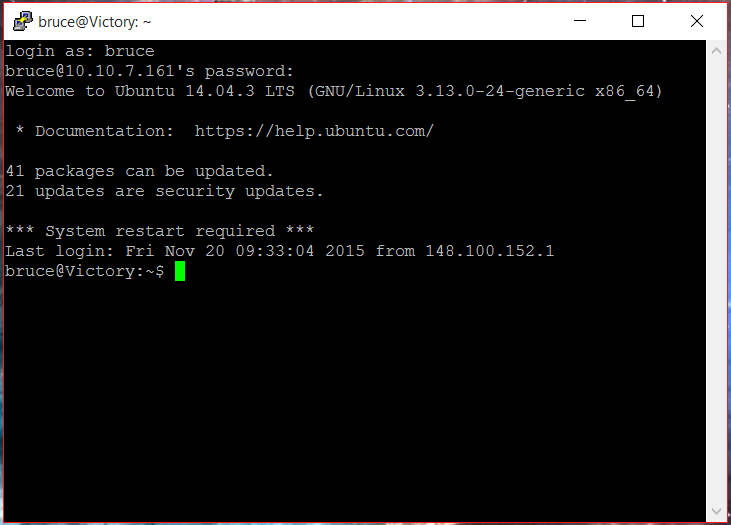
Server 1 (**Database**) (**Application**) has an IP address: 10.10.7.161, Subnet Mask: 255.255.255.0, Gateway: 10.10.7.1, DNS 10.12.1.11, DNS 10.12.1.12.

Server 2 (**Backup**) has an IP address: 10.10.7.162, Subnet Mask: 255.255.255.0, Gateway: 10.10.7.1, DNS 10.12.1.11, DNS 10.12.1.12.

**Step 6: Install Putty on PC**

We had to install Putty on our personal machines. Putty is a program that helps a user get access to an operating system (OS) they created through a command-line interface terminal. They can do so with the help from the IP address assigned to the OS, and default port of 22.  

Putty will then open up a command-line interface terminal that will ask for a username and password that was setup at the beginning of the creation of the server. Once the user inputs this information they will be granted access to the server’s terminal.



(10.10.7.161)

**Database Server**

**Step 1: Install PostgreSQL**

(PostgreSQl is an open source relational database management system.)

The following commands will install PostgreSQL on the server. We did this to each server.

Ensuring the list of available packages is up to date before installing anything new.

apt-get -y update

Then it’s a matter of just running one command for installation via apt-get:

apt-get -y install postgresql postgresql-contrib

**Let’s switch into that system user:**

su – postgres

**And then connect to the PostgreSQL terminal (in the postgres role):**

psql

**To get out of the PostgreSQl terminal:**

\q

This site helped us a lot through these steps: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-14-04

**Step 2: Install Git**

**(Git is an open source controlled system designed to handle small to larger projects.)**

The following commands will install Git on the server. I did this to each server.

Ensuring the list of available packages is up to date before installing anything new.

sudo apt-get update

sudo apt-get install build-essential libssl-dev libcurl4-gnutls-dev libexpat1-dev gettext unzip

Back on your Ubuntu 14.04 server, you can type wget and follow it by pasting the address you copied.

The URL that you copied may be different from mine:

wget https://github.com/weonyuan/Herby/archive/master.zip -o git.zip

Unzip the file that you downloaded and move into the resulting directory by typing:

unzip git .zip

cd git-\*

Now, you can make the package and install it by typing these two commands:

make prefix=/user/local all

sudo make prefix=/usr/local install

Now that you have git installed, if you want to upgrade to a later version, you can simply clone the repository and then build and install:

git clone github-windows://openRepo/https://github.com/weonyuan/Herby

This will create a new directory within your current directory where you can rebuild the package and reinstall the newer version, just like you did above. This will overwrite your older version with the new version:

make prefix=/user/local all

sudo make prefix=/usr/local install

How To SetUp Git

git config –-global user.name “Your Name”

git config –-global user.name “youremail@domain.com”

We can see all of the configuration items that have been set by typing:

git config –-list

**git configuration**

**user.name=Your Name**

**user.email=youremail@domain.com**

As you can see, this has a slightly different format. The information is stored in the configuration file, which you can optionally edit by hand with your text editor like this:

Nano ~/.gitconfig

~/.gitconfig contents

[user]

name = Your Name

email = youremail@domain.com

This site helped us a lot through these steps: <https://www.digitalocean.com/community/tutorials/how-to-install-git-on-ubuntu-14-04>

**Application Server**

**Step 1: Install Node**

**(Node.js is a Javascript platform for server-side programming that allows users to build network applications quickly.)**

sudo apt-get update  
sudo apt-get install nodejs

sudo apt-get install npm

Step 2: Install Nginx

**(Nginx is a Web Server that is great with performance and low memory usage.)**

Since this is our first interaction with the apt packaging system in this session, we should update our local package index before we begin so that we are using the most up-to-date information. Afterwards, we will install nginx:

sudo apt-get update  
sudo apt-get install nginx

Start Nginx Service

sudo service nginx start

Step 3: Install Curl

**(Curl is a command line tool for transferring data with URL syntax.)**

You can access the default Nginx landing page to confirm that the software is running properly by visiting your server's domain name or public IP address in your web browser, but first we have to install curl.

sudo apt-get install curl

This site helped us a lot through these steps: <https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-14-04-lts>

**Step 3: Check / Connect to Nginx on the Web**

**Enter the following commands**

ip addr show eth0 | grep inet | awk '{ print $2; }' | sed 's/\/.\*$//'

**curl http://icanhazip.com**

**When you have your Server's IP address or domain, enter it into your browser's address bar:**

**http://10.10.7.161**

**Stop Nginx Service:**

sudo service nginx stop

**We can make sure that our web server will restart automatically when the server is rebooted by typing:**

sudo update-rc.d nginx defaults

**This should already be enabled by default, so you may see a message like this:**

System start/stop links for /etc/init.d/nginx already exist.

https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-14-04-lts

(10.10.7.162)

**Backup Server**

Step 1: Install RSnapShots

sudo apt-get update  
sudo apt-get install rsnapshots

In order to backup another server, your backup server will need to be able to connect through SSH to the server you wish to back up

To generate a public and private key run this command on your backup server:

sudo ssh-keygen -t rsa

We need to make sure we can SSH to another server from the Backup Server.

**sudo apt-get install sshpass**

**sshpass -p password ssh bruce@10.10.7.161**

**exit**

Step 2: Configuring RSnapShots

**After you have installed rsnapshot, you will need to edit the configuration file.**

sudo nano /etc/rsnapshot.conf

Once in the configuration file you can move up with F7 and down with F8. Find the variable for each directory and make sure it corresponds to what’s below, if not edit it to match the variables.

**snapshot\_root /backup/**

**cmd\_ssh /usr/bin/ssh**

**cmd\_du /usr/bin/du**

We set these up under the “BACKUP INTERVALS” section of the configuration.

**#########################################  
#           BACKUP INTERVALS            #  
# Must be unique and in ascending order #  
# i.e. hourly, daily, weekly, etc.      #  
#########################################  
  
retain hourly  6  
retain daily   7  
retain weekly  4  
retain monthly 3**

Another item that may need to be edited is the ssh\_args variable and backup home directory.

**ssh\_args -p 25000**

**backup /home/ localhost/  
backup /etc/ localhost/**

**backup bruce@10.10.7.161:/home/ remote-droplet/**

Step 3: Testing the Configuration

sudo rsnapshot configtest

Enter the commands below to set when you want to backup snapshot to be taken.

sudo rsnapshot -t daily

sudo rsnapshot daily

Step 4: Automating the Process

sudo nano /etc/cron.d/rsnapshot

We’re going to remove the “#” character from the beginning of the scheduling section to activate these values.

**# This is a sample cron file for rsnapshot.  
# The values used correspond to the examples in /etc/rsnapshot.conf.  
# There you can also set the backup points and many other things.  
#  
# To activate this cron file you have to uncomment the lines below.  
# Feel free to adapt it to your needs.  
  
0 \*/4            \* \* \*           root    /usr/bin/rsnapshot hourly  
30 3          \* \* \*           root    /usr/bin/rsnapshot daily  
0  3            \* \* 1           root    /usr/bin/rsnapshot weekly  
30 2            1 \* \*           root    /usr/bin/rsnapshot monthly**

These settings will run add a snapshot to the “hourly” directory within our “/backup/” directory every four hours, add a daily snapshot everyday at 3:30 am, add a “weekly” snapshot every Monday at 3:00 am, and add a “monthly” snapshot on the first of every month at 2:30 am.

Step 5: Restoring the files from the Backup Server

Login to Backup.nixcraft.net.in, enter:

# ssh user@backup.nixcraft.net.in

Cd to your rsnapshot directory:

# cd /.raid/rsnapshot

To list current snapshots, enter:

# ls -l

The daily backups for server1 is stored at /.raid/rsnapshot/daily.\*/server1/ directory. To see latest daily backup (daily.0), enter:

You can restore backups to server1 using the rsync as follows to restore /var/www/html/:

# cd /.raid/rsnapshot/daily.0/server1/var/www/html/

# rsync -avr \* [user@server1.nixcraft.net.in](mailto:user@server1.nixcraft.net.in):/var/www/html/

OR use the scp command to copy selected file such as /var/www/html/db.conf.php:

## Our Credit Transfer Project: Foxify

In this section of documentation we are going to talk about the final project itself and give you a link to out GitHub repository again, give you a link to directly access our final version of our Capstone project so that you can use it and demo it in real-time, and we are going to also discuss some test cases that we have gone over to demonstrate extreme/boundary conditions, discuss different types of users for example malicious users, accidentally malicious users, and regular users.

As we have discussed earlier we have a GitHub repository where our code is hosted for your viewing pleasure. This is a great way to peak into the functionality to do further research and to better comprehend how exactly everything works. Link to the GitHub repository: <https://github.com/weonyuan/Herby>

Now I am sure that you are excited to go to and play with Foxify! Below we have put a link for you to follow to our site that is hosted on Marist servers. We have test data in our database that maps Dutchess Community College courses to Marist College courses and gives you a report based off of what potential majors you are interested in. This report tells you what credits will be accepted, what credits fulfil the Marist major requirement, and what credits are electives. You can pretend to be a prospective student and play around with it and see how it works out! Follow this link to Foxify: <http://10.10.7.161:8888/>

We wanted to Test out Application Foxify by seeing what type of traffic it could take, how it defends against malicious or accidentally malicious users, how secure it was, and how it handles when people do not fill out the forms correctly or if users put in non-valid information.

When we met with Chris Doyle in Admissions he said that during peak traffic times there are roughly 500 look-ups processed manually a year. These look-ups do not happen all at once, but are spread out throughout the two semesters and the summer and winter. Although the server is relatively small and the database is fairly small and simple we concluded that Foxify could handle 30-40 look-ups at a time without crashing or losing much performance. We found that when there were more look-ups being performed than that then we lost some performance speed. To fix this issue to be able to handle a higher workload we suggest that it should be hosted on a larger server and that some tweaks to the Database should be made. For the most part PostgreSQL is a very reliable source for hosting a database.

As a group we know that if there is website or an application there is someone who wants to break it of ‘hack’ it. In a way the saying from the movie “Field of Dreams” starring Kevin Costner hits the nail on the head: “If you build it they will come”. Malicious user are always looking on trying to break something so yes, they will come. As a group we recognize the threat and how important security is because this application is handling private student information. We made sure that it was not breach-able. We tried inputting SQL injection and we concluded that it had no effect on Foxify. SQL Injection do not have any effect on Foxify because we designed it to not get SQL requests on this application the API defends against this. Another case that we have thought about is someone trying to break in and restore someone else’s session. This is not an issue because when they submit their email they will get a randomly generated 6-digit number that is their key to restore their session number. No-one will be able to guess user e-mails with an encrypted randomly generated number.

When it comes to users that make errors while inputting their e-mail and or class information, it is really not a big deal at all. If a user accidentally puts in the wrong e-mail information it will go to the wrong e-mail that probably does not exist and there is no personal information about grades, names, schedule, address, or billing information. This will just be a link to a list of classes and no personal information is lost, but if the user reports their mistake we can delete the user and session ID in the database and the incorrect e-mail will not have access to the information when deleted. If the user just input a wrong class, that is an easy fix. They can just restore their session and correct their mistake.

## Key Features:

We are very proud of Foxify because we believe that it is very comprehensive and can make Admissions much more efficient. We have a few key feature that we would like to discuss that we believe are unique to out transfer credit program. Foxify has great scalability to fit and be functional on almost any device making it very easy to use over a plethora of devices and screen resolutions. We also believe that our comprehensive final report is very unique and fresh and that it give a an easy to read and understand report to the prospective student. Foxify also allows prospective student to save their session so that they can go back and change their courses and even select a different major. This is a very unique system that we developed that doesn’t make you have to remember a password. You sign up with your email and then our system generates a unique randomized session key that you type in with your e-mail and you are easily back to your saved session.

## Future Enhancements:

In the future when we have more time to improve on Foxify we already have some ideas from future enhancements. Some of the ideas that we have for the future of Foxify include the addition of minor checks and the option to include pathways. We would also like to work on the marketing of this product by reaching out to prospective student and maybe even create partnerships with other 2 years schools and Community colleges. Also, we would like to add an Admin Dashboard that has easy access to statistics about the usage and interests of people who have been using Foxify.

## User Help and Documentation:

For this application to be set up and work there are a lot of moving parts. In this section we will discuss some help features for users and administrators. Also we will discuss some things that we ran into so that the next team that works on this in the future has some sort of reference guide.

**Server set up and IT:**

For server set up, you should refer to the Infrastructure design part of this document. We have put in a great step by step process of how IT set everything up.

**Back-up and restore:**

This is also mentioned into the ‘Infrastructure design’ part of the paper, but here we will elaborate more on how to set up the backup server and how to restore or revert from the last back-up so that no data is lost.

Step 1: Install RSnapShots

sudo apt-get update  
sudo apt-get install rsnapshots

In order to backup another server, your backup server will need to be able to connect through SSH to the server you wish to back up

To generate a public and private key run this command on your backup server:

sudo ssh-keygen -t rsa

We need to make sure we can SSH to another server from the Backup Server.

**sudo apt-get install sshpass**

**sshpass -p password ssh bruce@10.10.7.161**

**exit**

Step 2: Configuring RSnapShots

**After you have installed rsnapshot, you will need to edit the configuration file.**

sudo nano /etc/rsnapshot.conf

Once in the configuration file you can move up with F7 and down with F8. Find the variable for each directory and make sure it corresponds to what’s below, if not edit it to match the variables.

**snapshot\_root /backup/**

**cmd\_ssh /usr/bin/ssh**

**cmd\_du /usr/bin/du**

We set these up under the “BACKUP INTERVALS” section of the configuration.

**#########################################  
#           BACKUP INTERVALS            #  
# Must be unique and in ascending order #  
# i.e. hourly, daily, weekly, etc.      #  
#########################################  
  
retain hourly  6  
retain daily   7  
retain weekly  4  
retain monthly 3**

Another item that may need to be edited is the ssh\_args variable and backup home directory.

**ssh\_args -p 25000**

**backup /home/ localhost/  
backup /etc/ localhost/**

**backup bruce@10.10.7.161:/home/ remote-droplet/**

Step 3: Testing the Configuration

sudo rsnapshot configtest

Enter the commands below to set when you want to backup snapshot to be taken.

sudo rsnapshot -t daily

sudo rsnapshot daily

Step 4: Automating the Process

sudo nano /etc/cron.d/rsnapshot

We’re going to remove the “#” character from the beginning of the scheduling section to activate these values.

**# This is a sample cron file for rsnapshot.  
# The values used correspond to the examples in /etc/rsnapshot.conf.  
# There you can also set the backup points and many other things.  
#  
# To activate this cron file you have to uncomment the lines below.  
# Feel free to adapt it to your needs.  
  
0 \*/4            \* \* \*           root    /usr/bin/rsnapshot hourly  
30 3          \* \* \*           root    /usr/bin/rsnapshot daily  
0  3            \* \* 1           root    /usr/bin/rsnapshot weekly  
30 2            1 \* \*           root    /usr/bin/rsnapshot monthly**

These settings will run add a snapshot to the “hourly” directory within our “/backup/” directory every four hours, add a daily snapshot everyday at 3:30 am, add a “weekly” snapshot every Monday at 3:00 am, and add a “monthly” snapshot on the first of every month at 2:30 am.

Step 5: Restoring the files from the Backup Server

Login to Backup.nixcraft.net.in, enter:

# ssh user@backup.nixcraft.net.in

Cd to your rsnapshot directory:

# cd /.raid/rsnapshot

To list current snapshots, enter:

# ls -l

The daily backups for server1 is stored at /.raid/rsnapshot/daily.\*/server1/ directory. To see latest daily backup (daily.0), enter:

You can restore backups to server1 using the rsync as follows to restore /var/www/html/:

# cd /.raid/rsnapshot/daily.0/server1/var/www/html/

# rsync -avr \* [user@server1.nixcraft.net.in](mailto:user@server1.nixcraft.net.in):/var/www/html/

OR use the scp command to copy selected file such as /var/www/html/db.conf.php:

**Node.JS API Documentations:**

If you are not familiar with Node.JS or API we have provided this documentation to help with comprehending how it work and how to set it up to get it up and running. This will be very important and helpful to solve many of the problems that we ran into.

Connection string: "postgres://postgres:alpaca@localhost:5432/transfercredits"

**ADD THIS HEADER TO “server.js”. THIS AUTHORIZES USERS TO ACCESS THE API.**

app.use(function(req, res, next) {

 res.header("Access-Control-Allow-Origin", "\*");

 res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

 next();

});

App.get requests

|  |  |  |
| --- | --- | --- |
| URL | Parameters | Result |
| /Subjects | None | Distinct list of Dutchess Subjects |
| /Majors | None | List of Marist Majors |
| /Minors | None | List of Marist Majors |
| /DCCCourseInfo/:subject | subject | Course name, course number, and credits of Dutchess classes where the subject equals the parameter provided. |
| /MajorCourseInfo/:major | major | Course name, course number, and credits of Marist classes that belong to the specified Major parameter provided. |
| /MinorCourseInfo/:minor | minor | Course name, course number, and credits of Marist classes that belong to the specified Minor parameter provided. |
| /StudentCourseInfo/:sesID | sesID | Course name, Subject, course number, and credits of Dutchess classes that belong to the specified sesID parameter provided. |
| /MaristTransferClasses/:major/:emailAdd' | major, emailAdd | Will return a JsonObj with a MajorClasses Array of classes a student has taken that count towards the specified Major. The JsonObj will also contain a nonMajorClasses Array of classes that transfer but do not count towards the selected major. It will also provide an array with classes that don’t transfer (See below for an example) |
| /ClassMappings | None | Will return a JsonObj with a mClasses and dClasses array. each one has an array of the classes that map to each other. They are sorted by the MarID of the Marist Class. Each object in the array contains MarID, CourseName,Subject, CourseNum, Credits, and DID |

App.post requests

|  |  |  |
| --- | --- | --- |
| URL | Parameters | Result |
| /InsertStudent/:fname/:lname/:emailAdd | fname, lname, emailAdd | Adds the Student to the People and Students table. |
| /InsertAdmin/:fname/:lname/:emailAdd/:password | fname, lname, emailAdd, password | Adds an Admin to the people and admin tables |
| /InsertStudentClasses | JsonObj (See below for example) | Will save the sesID and the classes a student has inputted into the Student\_Classes Table. |
| /InsertMaristClass | JsonObj (See below for example) | Will Insert a class into the M\_Classes Table |
| /InsertDutchessClass | JsonObj (See below for example) | Will Insert a class into the D\_Classes Table |
| /InsertClassMapping | JsonObj (See below for example) | Will Insert a class mapping into the Cred\_Transfers Table |
| /DeleteMaristClass | subject, courseNum | Will Delete the Marist class and any mapping it has |
| /DeletDutchessClass | subject, courseNum | Will Delete the Dutchess class and any mapping it has |

JsonObj for /InsertClasses

{"student\_classes": {

   "sesID": "111111",

   "emailAdd": "charles.ropes1@marist.edu",

   "session": [

     {

       "subject": "MATH",

       "courseNum": "222"

     },

     {

       "subject": "MATH",

       "courseNum": "223"

     },

     {

       "subject": "MATH",

       "courseNum": "215"

     }]

}

JsonObj for /MaristCourseName/:major/:emailAdd'

{

 "creditsNeeded": 120,

 "majorCreditsNeeded": 43,

 "creditTotal": 24,

 "majorCreditTotal": 18,

 "majorClasses": [

   {

     "marid": 48,

     "coursename": "Introduction to Programming",

     "coursenum": "120L",

     "subject": "CMPT",

     "credits": 4

   },

   {

     "marid": 26,

     "coursename": "Introduction to Mathematical Reasoning",

     "coursenum": "310L",

     "subject": "MATH",

     "credits": 3

   },

   {

     "marid": 25,

     "coursename": "Linear Algebra",

     "coursenum": "210L",

     "subject": "MATH",

     "credits": 3

   },

   {

     "marid": 24,

     "coursename": "Calculus III",

     "coursenum": "343L",

     "subject": "MATH",

     "credits": 4

   },

   {

     "marid": 23,

     "coursename": "Calculus II",

     "coursenum": "242L",

     "subject": "MATH",

     "credits": 4

   }

 ],

 "nonMajorClasses": [

   {

     "marid": 58,

     "coursename": "3D Design",

     "coursenum": "201L",

     "subject": "ART",

     "credits": 3

   },

   {

     "marid": 52,

     "coursename": "History of 20th Century Art",

     "coursenum": "366L",

     "subject": "ART",

     "credits": 3

   }

 ],

 "nonTransferClasses": [

   {

     "did": 25,

     "coursename": "Choral Sing III",

     "coursenum": "221",

     "subject": "MUS"

   }

 ]

}

/InsertMaristClass

{

   "subject": "FASH",

   "courseNum": "101L",

   "courseName": "Intro to Fashion",

   "credits": "3"

}

/InsertDutchessClass

{

   "subject": "FASH",

   "courseNum": "110",

   "courseName": "Intro to Fashion",

   "credits": "3"

}

/InsertClassMapping

{

   "maristSubject": "FASH",

   "maristCourseNum": "101L",

   "dutchessSubject": "FASH",

   "dutchessCourseNum": "110"

}

**Prospective student help:**

We really pride ourselves on how simplistic and functionally easy our product is. If for some reason a student needs help we have added the phone number and admissions e-mail to our site and if any of the prospective students need help, then they can just reach admissions by phone or e-mail.