NativeToMe

Final Report

Date: May 6, 2020

Prepared by: David Miller, Ralph Lira, Kevyn Higbee

Introduction

In today's social media landscape, there is a wide variety of content ranging from politics to funny cat videos. The common theme through it all is that all social media platforms today are orientated towards the individual. While this orientation is useful, NativeToMe aims to provide a social media platform that focuses on community rather than individuals. NativeToMe will address the lack of community by focusing on the qualities and attributes of a community. Consequently, NativeToMe will form a community of communities rather than a community of individuals.

System Functions

The list below is the list of functional requirements that we started with while writing the requirements documentation. As a part of the learning process for this project we learned to prioritize certain requirements over others. As a result, not all of our requirements were fully implemented. The requirements that were implemented will be marked with green and the requirements that are not yet implemented are marked in red.

Req#	Requirement	Comments	Priority
FR-01	The system shall allow users to login and logout securely.		1
FR-02	The system shall grant users different permissions based on their tier in a tribe.		3
FR-03	The system shall allow users to post content on their tribe's page.		1
FR-04	The system shall be able to appoint a new tribe leader based on the next longest membership within the tribe if the current tribe leader(s) leaves.		2
FR-05	The system shall allow users to request entry into an existing tribe		2

FR-06	The system shall allow tribe leaders to set privacy mode for the tribe	Open Door Policy, or Requested Entry	2
FR-07	The system shall allow each tribe to create tags based on its values and qualities.		3
FR-08	The system shall allow leaders to moderate their tribe.		2
FR-09	The system shall allow each tribe to create houses.		2
FR-10	The system shall allow tribes to merge with the consent of leaders.	Consent of tribe members is not necessary	2
FR-11	The system shall allow users to post content on their tent (profile page)		1
FR-12	The system shall allow users to like posts.		1
FR-13	The system shall allow up to 10 houses per tribe.	For better resource management server side	2
FR-14	The system shall dynamically limit features available to a tribe based on how many members it has.	A tribe cannot establish more houses until requirements are met.	2
FR-15	The system shall allow users to search for and filter tribes based on qualities.		1

FR-16	The system shall allow users to share content with their tribe.		2

Architecture

NativeToMe was developed using the Model-View-Controller(MVC) Architectural pattern coupled with the python based web framework, Django.

The high-level architecture for the system is shown in Figure 1-1.

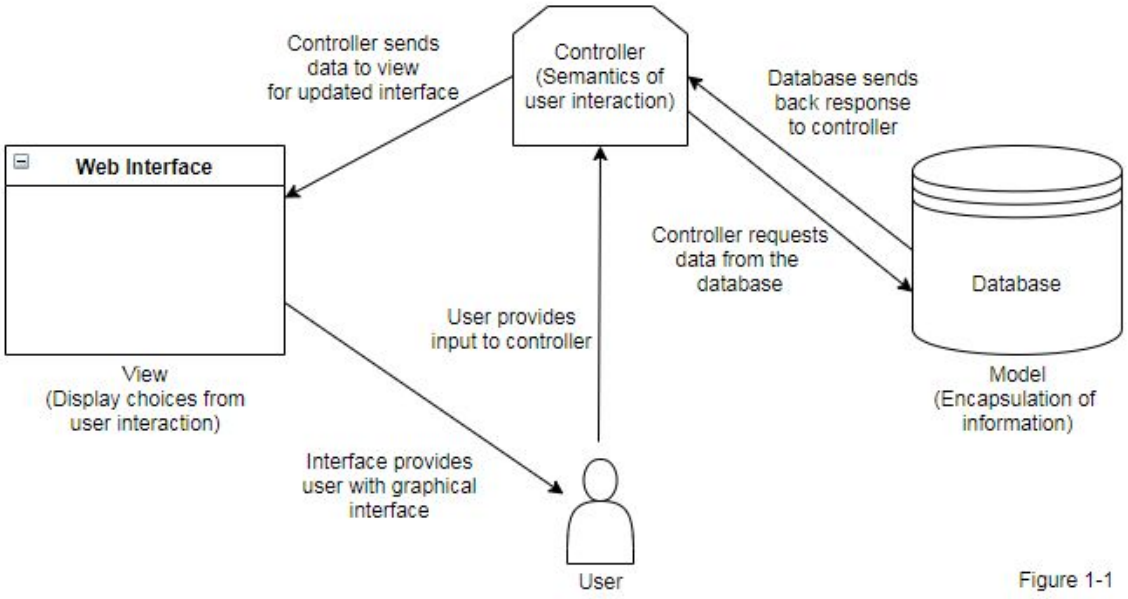

Figure 1-1

Model-Within the Django framework, the model.py files comprise the model in the image above. These files contain all the information as it relates to the SQL database and how the tables are to be set up. We are able to set primary keys, foreign keys, and the types of relationships each table will have if any. For example, NativeToMe maintains a heavy focus on the community and as such provides a gathering place known as a tribe. A tribe must contain many different columns to hold all the data required. A field such as tribeMembers might need a ManyToOne relationship with a User table. These types of relationships are defined in the model.py files. Finally, we are able to set default attributes for certain model fields. This takes some of the coding away from the views.py that will be covered later.

Controller-The control module within the MVC architectural pattern is handled with the views.py file in the Django framework. The views.py is the centerpiece of the Django framework as well as the MVC architecture. It connects the database to the HTML renderings. We are able to import the models, manipulate the data from the model, and pass the data to the HTML via a

context variable. The context variable is essentially providing access to a predefined variable from the views.py, in the HTML. We can access the variable with a “{{ variableName }}.” Setting up the context variable was a major aspect of developing NativeToMe because we had to ensure the proper data was being collected and passed to the correct rendering. Difficult bugs developed because of beginner errors. For example, the tribe home page has many different context variables that must be rendered at the appropriate time with the appropriate data being updated. As beginner’s with Django issues such as this could have been avoided with better coding practices and a deeper understanding of the framework.

View-The view portion of the MVC architectural pattern is accomplished within the HTML itself. As mentioned above, we are able to access predefined variables from the views.py in HTML using the double curly bracket “{{ variableName }}.” This allows for real time data to be displayed to the user, providing a fluid and responsive experience. The user is able to send data back to the controller via Django Forms. For example, if the User wants to update their profile they are able to access model variables and save their updates within a Form. The Controller takes that HTTP “POST” and saves the fields into the model. Additionally, the HTML is able to loop through the context variables that are passed to it from the views.py. We are able to do this with {% for <expression> %} and other typical control structures. This was especially convenient in producing a dynamic and fluid user experience because we were able to dynamically populate data as the user interacts with the website.

Implementation

NativeToMe was developed using a python based open source web framework called Django. The IDE used during development was Pycharm primarily. We were able to utilize a SQLite database to run the web application on a local machine. This allowed for much quicker debugging and testing of new functionality. Had the project gone to production we would have migrated to a standard database with an online host.

Implementation Tasks

NativeToMe uses Django to implement the database module that is able to interact with our communications module. Using Django, we were able to establish the following custom data structures and modules outlined in our architecture design: The Login/Create Account Module, UserActions Module, Profile Module, Tribe Module, Post Module. Using Python, we established these modules as models, views, and forms that conform to Django’s framework. From here we were able to easily link our database with the communications module and GUI. Finally we were able to run the database using SQLite.

The User Input Modules were also handled by Django’s form system. By creating Django forms in Python for our defined data structures, we were able to easily receive user input from the html GUI. Using these forms, we were able to accomplish the goal of our UserInput Module and Update Module.

To implement our views, we used html files. For each different webpage, we used a new html file that is accessible via the NativeToMe home.html or through other pages. We implemented the user home page, tribe home page, house page, edit pages for users,tribes,and houses, as well as log in/out and register account pages. Any input that user's placed in the page were retrieved using the forms discussed in the previous paragraph.

Although in our final demonstration we decided to showcase using an offline database, Django easily allows for that database to become online with the addition of connection handling (already implemented in Django) and using the parameter `python manage.py runserver 0.0.0.0:8000` rather than running the server with no parameters.

Consistency

Referring back to our original NativeToMe Requirement Specification document, we were able to implement all three of our user interface requirements. We created a user log-in page where a user can either sign in as an existing user or create a new profile, a main tribe homepage to explore and post content, and the individual user's homepage where they can update their profile to their preference and upload a new profile picture.

As for the functional requirements we included in our system functions section described above, we were only able to complete 9 out of the16 requirements we had listed. We completed 4 out of the 5 priority 1 requirements; the priority 1 requirement we could not finish was the allowing the user to like posts, but it was because we decided to scrap this feature near the end of our development cycle. We completed 4 out of the 9 priority 2 requirements and 1 out of the 2 priority 3 requirements. If we had more time to develop NativeToMe, we would have added these extra features defined in the system functions table above.

To further explain, we scrapped the priority 1 "like posts" functions because we did not see how it would really serve to be beneficial. Also, near the end of development, we decided to get rid of some features that we would not have time to develop. For priority 2 functions, we were not able to implement: appointment of new leaders, moderation of tribes by leaders, allow tribes to merge with each other, allow up to 10 houses per tribe, and allow for leaders to limit the features a member of said tribe has. These features were not done due to time crunch. For some, we had the bones that would make up these features but we could not finish them with the time and knowledge we had. Due to everyone included in the development process of the application was learning and using python and django for the first time to make a web-based application, there was a steep learning curve to adjust to everything. So we were determined to try and complete all of our priority 1 features and if time was left to finish the priority 2 and 3. Finally, as for the priority 3 requirement "allow tribes to create tags based on their values and qualities", this is a feature that is not critical to the finished product in terms of its functionality. Again, if we had more time, we would have finished this feature.

We followed our original design of using an MVC architectural pattern to build our web application. Luckily, we did not have to break away from the original proposal of NativeToMe but we did need to make functional changes along the way to finish and make a version of the product that was presentable.

System Availability

During this project, we used a private github repository, the following link is a public clone of that private repository: https://github.com/dav1dmiller/NativeToMe

In presentation 3 we used the following video demo: https://drive.google.com/file/d/1NShgqcRIuc_vC_j6__aJ-wC15fdrg9cC/view?usp=sharing_eil&ts=5eb22671

Project Management

Name	**Framework Research**	**Frontend**	**Backend**	**Final Report**
David Miller	*	*	*	*
Ralph Lira	*	*	* Minor role in assisting in backend development	*
Kevyn Higbee	*	*		*

During the development of this project, we faced issues due to unfamiliarity with developing an app and the tools used to make it, transferring files to one another to make sure we are all working on the same version, unexpected bugs during testing, and time management.

Python and django were difficult to use due to it being foreign to us. In order to understand how to use it, we relied on watching videos based on using django for development purposes. Initially it was difficult seeing how everything related to each other but over time it became easier, but it could not be said that we became experts at using it, only familiar. Now, we used GitHub to transfer our files to one another, but more specifically the GitHub Desktop application. We did not have experience using GitHub so when we transferred files, we came into issues with conflicts. We lost track of which exact files were updated and which were not, so when we came into conflicts when pushing to the master branch and would need to decide whether we are still using the files from the master or our local when committing, we could overwrite something important that did not need to be changed, so this was a user error, but also sometimes the desktop app was buggy and would revert itself back to previous versions. We

would push through the issues and continue and re-do what needed to be done. While testing, unexpected bugs would occur and we would not be sure how to fix these issues. Due to django having an ease of use in terms of having a lot of operations done for you, it would be difficult debugging and locating where an underlying issue was based from the compile error message it would display. As a result, this would leave us with having to leave in certain bugs that we could not fix or leave out certain details we had before, but needed to take out because it was no longer working as intended. Lastly, we were on a time frame. There were a lot of times where we could have used out time more wisely but with other classes and projects and assignments, we could not have all our focus shifted towards the project. However, that does not mean that everything we intended to have in the final product could not have been completed, it certainly could have been met. Even with all the time and effort and learning we put into the development of NativeToMe, we were satisfied with the final version we completed. Overall, these were the issues we faced during the development of NativeToMe.

Conclusion

Developing a social media web application such as NativeToMe definitely was a difficult challenge. There were many moments of frustration and disagreements on certain aspects of the development but we did our best to create a product we were satisfied with. This project was an eye-opener because it showed us a glimpse as to what goes into the development cycle and consideration of an application and it was good practice for potential projects that may be similar in the future. We now know that we need to put a lot of time and effort into the development of a product in order for it to be met at a certain standard, but also, know our limitations. With limited experience at using the tools necessary for the creation of NativeToMe, we need to acknowledge that we cannot complete everything we set out to have but still have something to be proud of. This project is an experience that will stay with us and inspire us to continue our future into computer science and development.

References

IEEE Computer Society. (1998). *IEEE Recommended Practice for Software Requirements Specifications*. Retrieved from https://standards.ieee.org/standard/830-1998.html

“Open Door Policy (Business).” *Wikipedia*, Wikimedia Foundation, 20 Nov. 2018, en.wikipedia.org/wiki/Open_door_policy_(business).

Zheng, Yongjie. (2020). SE04-CS441-Software Requirements Specification[PPT]