Lafayette College Image Labeling Game Development Manual

Purpose of This Manual:

This manual is intended to be a guide for future developers of the Lafayette College Image Labeling Game. This manual outlines the different components of the program and how they interact. Also included are instructions for setting up a development environment and a description of all the tools needed to make changes to the program. This includes frontend, backend and database tools and the languages they require. The steps taken to startup the program are described so all changes can be implemented on the web server. Difficulties we encountered are listed as well, providing future developers hints and specific cautions about problematic areas that the original developers dealt with. Finally, this manual makes suggestions for the direction of future developments and additions to be made to the game.

A. Introduction to Structure

This Introduction to Structure section will familiarize the developer with the technologies, frameworks, languages and systems utilized during the development of this project. This section will explore the software the system was run on, the version control mechanism used, the framework utilized for development, the high level architecture of the system, the database language used and various other languages that were used for implementation.

1. Virtual Machine

During the development process, our team was provided a virtual machine found on a unique IP address that is hosted by the Lafayette College servers. A virtual machine is a computer file that behaves as an actual computer. Essentially, it is a computer within a computer that provides an ideal environment for software development. The IP address provided for our team was 139.147.9.239 which we could access through separate accounts. The admin of the virtual machine (Professor Ordille) utilized our Lafayette username to provide us with separate accounts that we could use to access the virtual machine through the secure shell protocol (SSH) Linux command.

2. Git

As a team, we needed a version control system to help us manage changes to source code over time. The chosen version control software, Git, keeps track of every modification to the code in a special type of database so that if a mistake is made, developers are able to compare earlier versions of the code to help fix the mistake without disruption to the rest of the team. Git is a Distributed Version Control System which allows every team member to have a working copy of the code in a repository that can contain the full history of changes. This is beneficial because one does not have to have a network connection to do most of the git commands. The git repository was initialized within the virtual machine and each team member used their generated public key. This way each team member was able to work with their own local copy of the code repository on their personal computing device. This was beneficial to us throughout the development process as we were able to remain in sync and be up to date with each others changes.

3. Django: MVC Framework

The main platform we used for development is the Django application, which is a high-level Python Web framework that supports rapid production with clean and pragmatic design. A web framework like Django handles much of the hassle of web development by providing easy to use methods for the HTTP protocol and taking care of security. Django follows the Model-Template-View (MTV) architectural pattern, which is similar to the Model-View Controller (MVC) development pattern that most frameworks use. The model is the data access layer that contains all details pertaining to the data, such as how to access it, how it behaves, how to validate it and the relationships between the data. The template is the presentation layer that contains front end related decisions, such as how something should be displayed on a Web page. The view is the business logic layer that contains the logic that accesses the models and defers the information to the appropriate templates. It acts as a bridge between the model data layer and the template presentation layer.

Figure 1 shows the correlation between each portion of web development (back-end, middleware, front-end and user display) with each portion of the MTV framework. Our team choose this framework in particular for its ability to aid us in productively building a scalable, secure and fast growing web application. In addition, our desire to learn Python encouraged us to select this framework. With Django we were able to focus on the design and development process, rather than dive into the complexities of web development. As a fully loaded framework, Django allowed us to utilize open source libraries with aided us with user authentication, security issues and content administration. The latest development version of Django (Django 2.0) was installed directly onto our virtual machine.

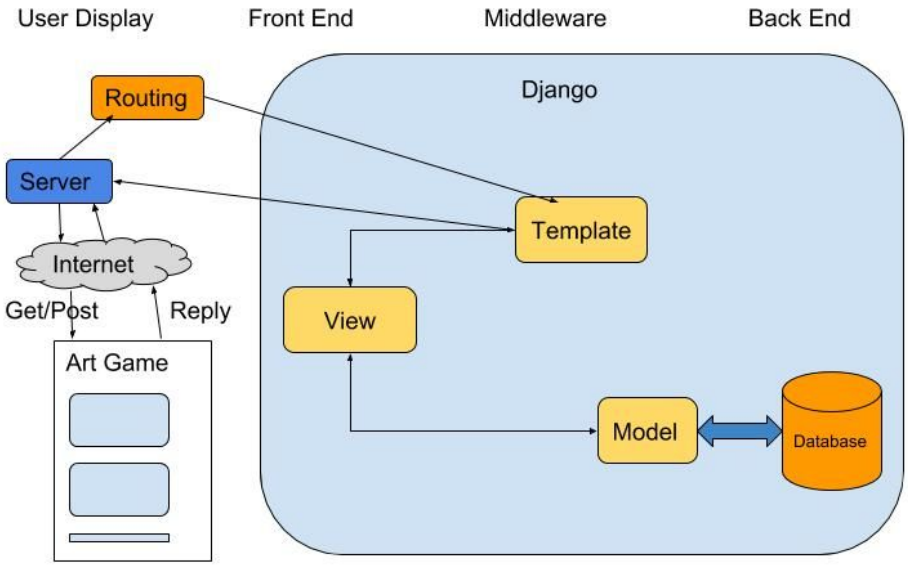

Figure 1: Visual of Django MVC framework

4. Django: Project and Applications

Once Django was correctly installed onto our virtual machine, we created a Django project which is a collection of settings for an instance of Django. This project includes database configuration, Django-specific options, application-specific settings and auto-generates source code. This was done using the command:

```
django-admin startproject artgame
```

This project creation command established an outer root directory called artgame which is a container for our project, a manage.py file which is a command-line utility file that allowed us to interact with our Django project and an inner directory called artgame which is the actual Python package for our project. Within this inner artame directory, the settings.py file allowed us to control the project fongurations, the urls.py file allowed us to configure the URL declarations for the project and the wsgi.py file served as an entry point for WSGI-compatible web servers to serve our project.

Once the Django project was set up, we were able to create Django apps within it. Each application you write in Django consists of a Python package that follows a certain convention that is already generated. When the artgame project was created, it auto generated an art game app directory. To store all of the functionality for the web application, a game app was created. This was done using the command (must be in the same directory as the manage.py file within the Django project) :

```
python manage.py startapp game
```

This app creation command established many different Python files which are used to create functionality. The admin.py file is used to display the created models into the admin portal which can also be customized in this file. The apps.py file allows the developer to include any application configuration for the app. The models.py file contains all the models that was implemented within the project. Models are a special type of object which is saved inside the database. The tests.py file is used to complete any sort of testing for the web application. The views.py file contains view functions that takes a Web request and returns the corresponding Web response. Finally, the migrations folder is where the developer can propagate changes one makes to the models into the database schema. A urls.py file was created to establish and configure URL declarations for the app.

The Django project was established within the account of the virtual machine of one of the team members than pushed to the repository using GIt so that all members had access. The Django applications were established within the Django project directory and also pushed to the repository using Git. To compare, a project is a collection of configuration and apps for a particular website while an app is a Web application that contains functionality. A project can contain multiple apps and an app can be in multiple projects.

Figure 2 shows the overview of our project. The virtual machine houses our entire web application. Within the virtual machine, the 'ArtGame' folder houses our Git repository. Within the Git repository, the argame folder is the project root directory which contains the artgame app (project configuration directory) and the game app (code for the final product). *Figure 3* shows what this looks like directly in the code.

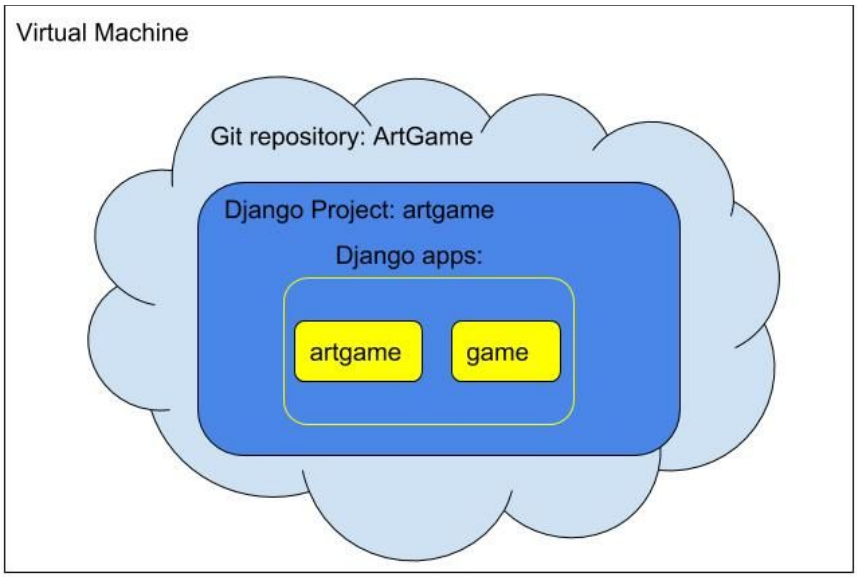

Figure 2: Visual Architecture

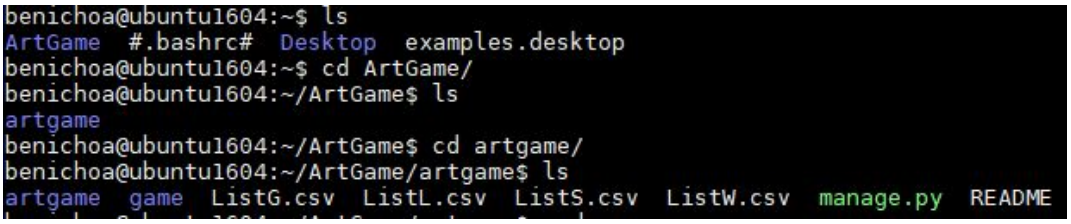

Figure 3: Visual Code Access

5. Python

Python is an interpreted high-level programing language that Django is written in. Django 2.0 supports the latest version of Python, which is Python 3.4 so both of these versions are compatible with one another. Initially, Python 3.0 was downloaded which caused all Django commands have to start with 'python3', rather than simply 'python.' To fix this, we installed Python 3.4 onto a virtual environment within the virtual machine. In order to access this latest version, we needed to type in the command `workon dj` immediately after SSHing into the virtual machine. This command allows us to develop using the latest version of Python and its corresponding packages which is the one compatible with the version of Django we have installed.

6. PostgreSQL

PostgreSQL is an open-source relational database management system that uses and extends the SQL languages. PSQL combines SQL with many features that safely store and scale the most complicated data workloads. PSQL is the chosen database language to be used with Django. Our database is called cardgamedb and it is configured to work with Django with the settings outlined in the settings.py file within the Django project. To create data to be added to the database, Django models are created within the apps.py file in the app folder. As previously discussed, a model is the single and definitive source of information about our data. Eahc model contains the essential fields and behaviors of the data to be stored, such that each model maps to a single database table with PSQL. Every Django model is a Python class that subclasses django.dj.models.Model and each attribute of a model represents a database field.

The database models that we created include Players, Artwork, Keywords, Categories and Playerword. Each of these models are prefixed with 'game_' in PSQL to dictate their source. Images for the Artwork model are stored in the file system at /home/cardgamedb/images/artwork. Any remaining models within the database were automatically generated by Django to handle the build in features we are enabling, such as user authentication. Changes to a model si automatically created by Django using the command:

```
python manage.py makemigrations
```

Note that this command must be done in the same directory as the manage.py file resides. These changes can be migrated into the actual PSQL database using the command:

```
Python manage.py migrate
```

7. Apache: Updating and starting the server

The Apache HTTP Server is a free and open-source cross-platform Web server which provides a secure, efficient and extensible serves that provides HTTP in sync with the current HTTP standards. We utilized an Ubuntu (Linux) Apache server to launch the production version of our product, which is an environment running on the virtual machine. The code served by Apache exists in /user/ArtGame and the configuration files /etc/apache2/sites-available/000-default.conf and ../confs-available/wsgi.conf were modified to deliver our product through Apache. A module called mod_wsgi was installed and enabled in Apache. The file Artgame/artgame/artgame/wsgi.py is what mod_wsgi access to serve our product. It was modified from the original state to incorporate necessary features. The wsgi.py file is the only file Apache needs access to, which then serves the remainder of the code. The Apache server can be started and stopped using the command: `sudo apache2ctl start` and `sudo apache2ctl stop`, respectively. The modules, configurations and sites can be enabled using the following parameters: `a2enmod`, `a2enconf` and `a2ensite`, respectively.

8. Front end: HTML, CSS and Javascript

Front-end development is the practice of converting data into a graphical interface for the user to view and digitally interact with through HTML, CSS and Javascript. HTML (Hypertext Markup Language) is the standard markup language for creating web pages and web applications. HTML describes the structure of the web page in terms of elements that are the building blocks of the page and are represented by HTML tags. The browser does not display the HTML tags but it uses them to render the content to the page. CSS (Cascading Style Sheets) is a

style sheet language used for describing the presentation of a page written in HTML. Specifically, CSS describes how the HTML elements on the page are to be displayed on the screen. JAvaScript is a high-level, interpreted language that is used to create interactive effects within a web browser. To bring it all back together: HTML defines the content of the web pages, CSS specifies the layout of the web pages and JavaScript programs the behavior of web pages.

Within our project, the view files are all found within the game app. *Figure 4* shows a screenshot of the top level game app directory, which contains many different files used for development. The ones important for front end functionality is the 'static' folder and the 'templates' folder. The 'static' folder is where Django looks for static files and the 'template' folder is where Django looks for template files.

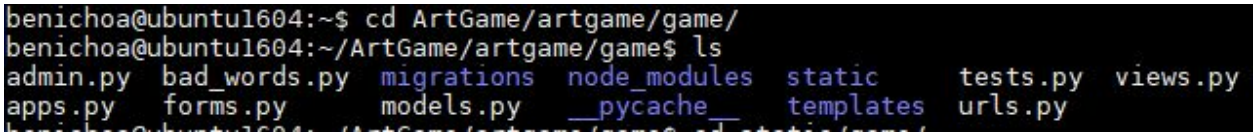

Figure 4: Top level view of game app

Within the 'static' folder, there is a 'game' folder as shown in *Figure 5*. Static files include CSS, JavaScript and images. The internal 'game' folder is needed so that Django knows what app this static folder is for. This directory contains the images we use for our home made and a CSS file which contains all the code for describing the layout of webpages.

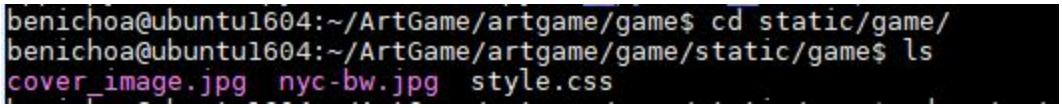

Figure 5: Static directory

Within 'templates' folder, there is a 'game' directory and a 'registration' directory as shown in *Figure 6*. This 'templates' directory is where Django will looks for templates that are referenced to in the views.py file. These templates describe how Django loads and renders templates. The 'game' subdirectory is where Django will look for all templates referenced to in

the views.py file and the ‘registration’ subdirectory is where Django looks for templates for the built in system, such as user authentication.

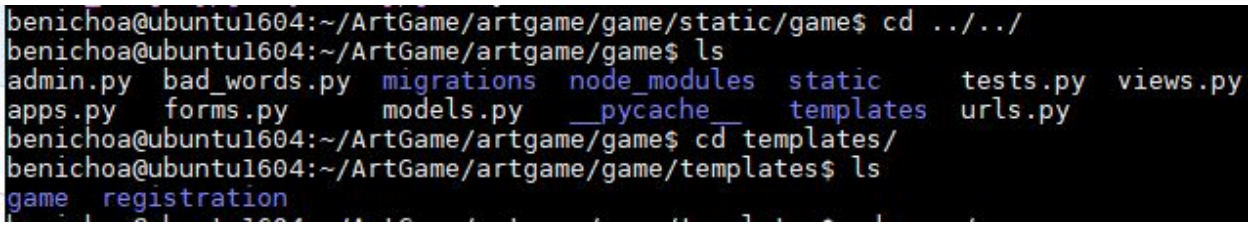

Figure 6: Template directory

The contents of the ‘game’ and ‘registration’ subdirectory can be seen in *Figure 7*. The ‘game’ subdirectory contains the HTML files for the static and dynamic web pages in our web application. The base.html file contains code for the navigation bar, the footer and the overarching theme. Every other template file imports the base.html file for a common theme. The about.html template contains the view for the about page, the home.html template contains the view for the home page, the profile.html template contains the view for the profile, the register.html template contains the view for the registration form, the leaderboard.html template contains the view for the leaderboard and the game.html template contains the view for the game portal. The ‘registration’ subdirectory contains the HTML files that Django will look for when using its built in features. With this, login, logout, sign up and password reset can be implemented. The login.html template contains code for the login page and all templates that start with password_reset_ are used for the password reset functionality.

Figure 7: Game and Registration subdirectory

B. Production Cycle

This Production Cycle section will introduce the concept of Agile Web Development and we utilized this software development methodology, in conjunction with the scrum process, during the development of this project. Additionally, this section will explore each sprints and the items completed during the iteration of the sprint.

1. Agile web development

Throughout the completion of this project, our group followed the Agile Software Development Methodology to efficiently develop the web application. By utilizing agile, our team was able to collaboratively provide rapid and continuous development of our software solution. Specifically, we applied the scrum process of agile methodology to collectively determine a product backlog, which consisted of the tasks that need to be completed and features that need to be implemented for the project at hand. The items in the product backlog were split up into three consecutive sprints. A sprint is an scrum tool used as a time box which, in our case, spanned 2 to 3 weeks. During this time, the items in the product backlog are divided among the team members who are tasked with their complete implementation.

2. Sprint 1: Build a Minimally Viable System

Sprint 1 was the first of three sprints completed during the development of this project. The Sprint 1 backlog consisted of setting up the necessary software development tools, such as activating the virtual machine for each member, initializing a Git repository in the virtual machines using SSH keys and installing virtual environments and python packages into the virtual machine. Once the foundation was set up, we created a Django project which is a collection of configuration and applications for the web application and created a PostgreSQL account for the project by establishing the artgame db PSQL user. Within the Django project, we created two Django applications which are the web applications that implement functionality. The artgame app is the root directory that acts as a container for the project and the game app is used to handle all items related to the game interface. Within the game app, a navigation bar was

implemented which contained all the desired tabs for our game: homepage, game page, about page, profile page, login page, register page and leaderboard page.

3. Sprint 2: System with Most Features

Sprint 2 was the second of three sprints completed during the development of this project. The Sprint 2 backlog consisted of completing user authentication so a user could now create an account, login and logout, implementing leaderboard functionality to display the top 10 current players on a chart and fixing a navigation bar error to allow dynamic navigation throughout the game portal. Additionally, we completed a portion of the game functionality which is the main component of our web application This included retrieving photographs currently saved in the database and displaying them to the screen, implementing a timer that counts down from 60 seconds, adding a input box for users to place their generated labels for the photograph and allowing keywords to be inputted to the box for them to be saved into the database with the photograph they were associated with.

4. Sprint 3: Prepare for Delivery

Sprint 3 was the final of three sprints completed during the development of this project. The Sprint 3 backlog consisted of editing the digitized photographs provided by our customer to ensure they were all the same rotation, uploading the digitized photographs to the game database to allow them to be displayed in the game interface and implementing the categorization feature so that players have the option of selecting a certain photographer whose photographs they want to label. The major portion of this sprint was completing the game functionality so that it worked properly. This included changing the photograph displayed on the screen when the 60 second timer completes, creating an algorithm to generate points for a given label based on its relevance, adding a text display box for the users generated labels to be pinned to while they are labeling a given photograph and keeping track of the players current score for the given photograph which gets added to the players current score and then refreshes once the photograph is changed. Additionally, we implemented a mechanism to filter out words that are considered unsavory, repetitive or that contain symbols or numbers. When this occurred, an error would be displayed

at the top of the game portal to inform the user that some sort of invalid keyword was inputted. We altered the game access so that a user that does not have an account cannot view the game portal, thus cannot view the photographs. The final item that was completed is the set up of the production server so that game portal could be viewed by anyone based on an IP address. This transition from the development server within the virtual machine to the production server through the Internet marked the debut of the portal.

C. Running the application

In this section, we will discuss how to get Django and all of its components up and running on your personal computing device. Please note that this can be done by following the instructions documented here: https://docs.djangoproject.com/en/2.0/topics/install/. Additionally, muhc of this installation was done by Professor Ordille and Professor Liew, as they are the admins and the ones who have access to the virtual machines.

1. System Setup

The Django version that was installed was Django 2.0, which is only compatible with Python version 3.4. The first step is to download the latest version of Python by following this link: https://www.python.org/downloads/. To use Django on a production site, one must download Apache and mod_wsgi. Download the latest version of Apache by following the directions indicated in this link: https://httpd.apache.org/ and install the mod_wsgi package yb following the directions indicated in this link: http://modwsgi.readthedocs.io/en/develop/. For this to work, Apache must be installed with the mod_wsgi module activated. For directions on how to configure mod_wsgi once it is installed, follow the directions indicated on this link: https://docs.djangoproject.com/en/2.0/howto/deployment/wsgi/modwsgi/.

2. Database setup

To use Django's database functionality, one must get the database server up and running. Django supports many different database servers, but we chose to use PostgreSQL which can be downloaded by following this link: https://www.postgresql.org/. From using PSQL, we needed to download a psycopg2 package which is the PSQL adapter for Python. The latest psycopg2 version can be found at this link: http://initd.org/psycopg/. Once this has all been installed, one can migrate their models by following the instructions dictated in Section A, subsection 6 PostgreSQL.

3. Model Organization

There are five models defined in models.py; Player, Artwork, Keyword, Category, and PlayerWord. Unless a different primary key is created, all models have an ‘id’ field. Player contains a username, next artwork to display, total score, highest score in a game, current score for the current game, current category selected, total games played, a place to save the current timer state. The username field must match that of an authenticated user. The Artwork model represents an image to be labelled in the game. It contains an image field for the uploaded artwork, an upload date, and a category which foreign keys to the Category model.The keyword model contains an artwork foreign key which the word is describing, a text field for the actual contents of the word, and a count which is incremented every time the word is re-entered. The category model contains solely a category name represented in text. The PlayerWord model is a backend location to store words which a player has currently inputted for the current image. It contains a foreign key to a player it belongs to, the points which were awarded for that word, and the contents of the word itself. This information is used for display in the white box next to the image while playing. All models contains method definitions for functionality needed from these models. These functions are then used in the view layer to generate the game functionality. *Figure 8* shows the relation between these models in the database.

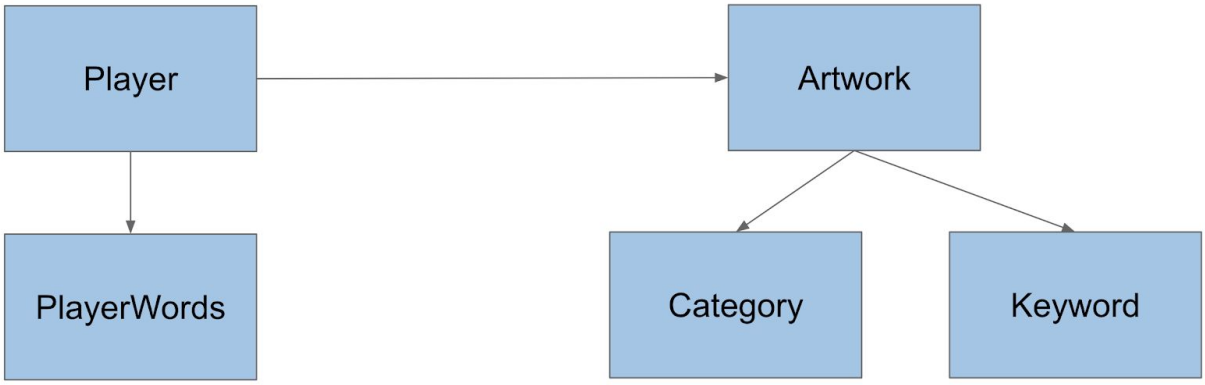

Figure 8: Relation of models

4. Running the Django development server

Within the root artgame directory is a manage.py file which allowed us to interact with the Django project. One of these interactions includes running the Django provided development server. This Django development environment is an installation of Django on our local computers that can use for development and testing Django apps prior to deploying them to a production server. To run the development server, one must be in the same directory as the manage.py file, have the Python virtual environment launched using `workon dj` and the run:

```
python manage.py runserver
```

If done correctly, the output should resemble *Figure 9* below:

```
(dj) benichoa@ubuntu1604:~/ArtGame$ cd artgame/
(dj) benichoa@ubuntu1604:~/ArtGame/artgame$ ls
artgame  game  ListG.csv  ListL.csv  ListS.csv  ListW.csv  manage.py  README
(dj) benichoa@ubuntu1604:~/ArtGame/artgame$ python manage.py runserver
/home/benichoa/.virtualenvs/dj/lib/python3.5/site-packages/psycopg2/__init__.py:144: UserWarning: The psyco
pg2 wheel package will be renamed from release 2.8; in order to keep installing from binary please use "pip
 install psycopg2-binary" instead. For details see: <http://initd.org/psycopg/docs/install.html#binary-inst
all-from-pypi>.
  """)
/home/benichoa/.virtualenvs/dj/lib/python3.5/site-packages/psycopg2/__init__.py:144: UserWarning: The psyco
pg2 wheel package will be renamed from release 2.8; in order to keep installing from binary please use "pip
 install psycopg2-binary" instead. For details see: <http://initd.org/psycopg/docs/install.html#binary-inst
all-from-pypi>.
  """)
Performing system checks...

System check identified no issues (0 silenced).
May 07, 2018 - 21:56:58
Django version 2.0.2, using settings 'artgame.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Figure 9: Running the development server

This message indicates that the Django development server has been started. By default, runserver command starts the development server on the internet IP at port 8000. If there are multiple developers using the development server at once with their local copy of the repository, they can indicate which port they would like to start the development server on by passing the desired port number to the python command as such: `python manage.py runserver 80xx`. From here, you can visit https://127.0.0.1:8000 within a Web browser. From the virtual machine, one can run firefox 127.0.0.1:8000 in another command line window. This will pull up a Firefox window of the listed IP address which looks as indicated in *Figure 10*:

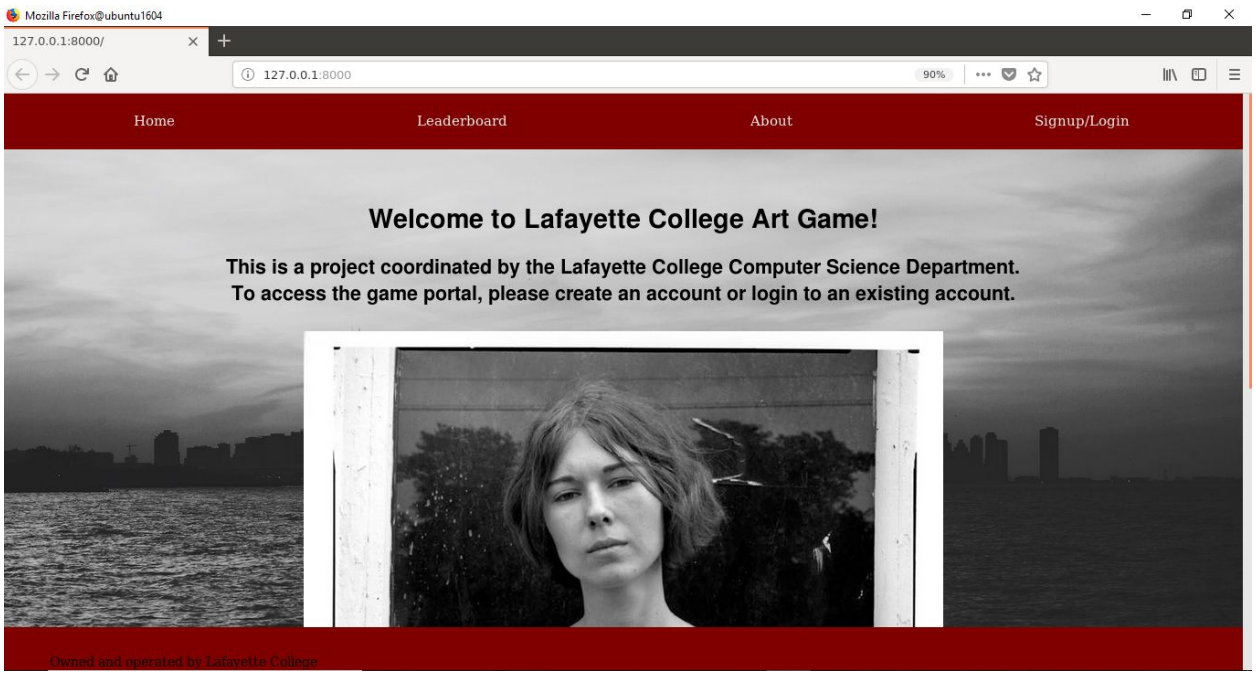

Figure 10: The development server on firefox

Once the development server has been run and the firefox window has been opened, one can make edits and changes within their local copy of the repository and it will be updated on the development server in real time. This way, one can see the immediate effects of a change. This is extremely helpful when playing around with front end functionality to see how adding a CSS line or HTML tag will update the page at hand. Additionally, one can quickly test a functionality to see if it is working as they think it is.

D. Bugs and other potential problems

Overall, there are no impeding bugs in the code. However, it is worth mentioning that an attempt was made at the password reset mechanism but there was an error in the process that we were unable to debug. All the necessary view templates are set up to import the Django password reset configuration but a URL error would not allow us to do so. Additionally, image resizing causes a big problem to the game portal. Although we did not digitize the images ourselves, we had to resize them considerable and edit them so that they were all the same rotation. The images displayed in the game portal are varying size and sometimes they are too large for all the features to be shown. Finally, the labels that a user inputs are being checked to see if they are slanderous but they are not checked for spelling. This could bring up problems in the future when an additional word is associated with an image because it has a different spelling.

E. Future Implementations

Throughout the 15 weeks of this semester, we were able to our product up and running by using our Software Engineering skills, the Django framework and with the guidance of Professor Ordille. Despite our hard work, there are many implementations which we would have liked to complete but that we were unable to due to unforeseen difficulties. One major feature we would have wanted to implement would be allowing users to identify themselves as a student, faculty or administrator upon registration. This mechanism could dictate their authorization for certain features. Administrators would have all the privileges among the web application, such as managing users, uploaded images, removing images and choosing categories. The main difference between faculty and users would be the weight that their labels play in the algorithm to determine label relevance. We would assume that faculty would have a firmer grasp on photographs, thus their labels would be considered with greater weight. Additionally, an email confirmation upon registration could ensure that a valid Lafayette email address was being used.

Currently, images are uploaded through the Django provided admin portal from a local directory. We would have wanted to provide non-technical administrators with a mechanism to

easily choose a folder full of photographs from their local directory. Once uploaded, the administrator could select if they wanted these images to be grouped into a category for the game portal and then the images would become available to the users on the game portal. Int terms of production, we would have wanted to register a domain name with Lafayette college. This domain name could be sites.lafayette.edu/imagelabeler or compsci.lafayette.edu/imagelabeler. A feature of the game that we would have wanted to enhance is the profile. We had in mind to implement a ranking system, where users earn a new rank as they label images and earn points.

The next step for this project, other than to enhance the game application as previously discussed, would be to create a search engine for these images. The results from the labels in the image game labeler could be funneled through to a search engine, so that each label is associated with each image. This would include creating a search engine which houses all the images uploaded to the game portal and funneling the generated labels from the game portal into the search engine for immediate update.

F. References

"Django 2.0 Development API." *Django Documentation*, Django Software Foundation, docs.djangoproject.com/en/2.0/.

"What Is a Virtual Machine and How Does It Work | Microsoft Azure." Microsoft Azure, Microsoft, azure.microsoft.com/en-us/overview/what-is-a-virtual-machine/.

"What Is Git: Become a pro at Git with This Guide | Atlassian Git Tutorial.", Atlassian, www.atlassian.com/git/tutorials/what-is-git.

"The Model-View-Controller Design Pattern - Python Django Tutorials." *The Django Book*, djangobook.com/model-view-controller-design-pattern/.