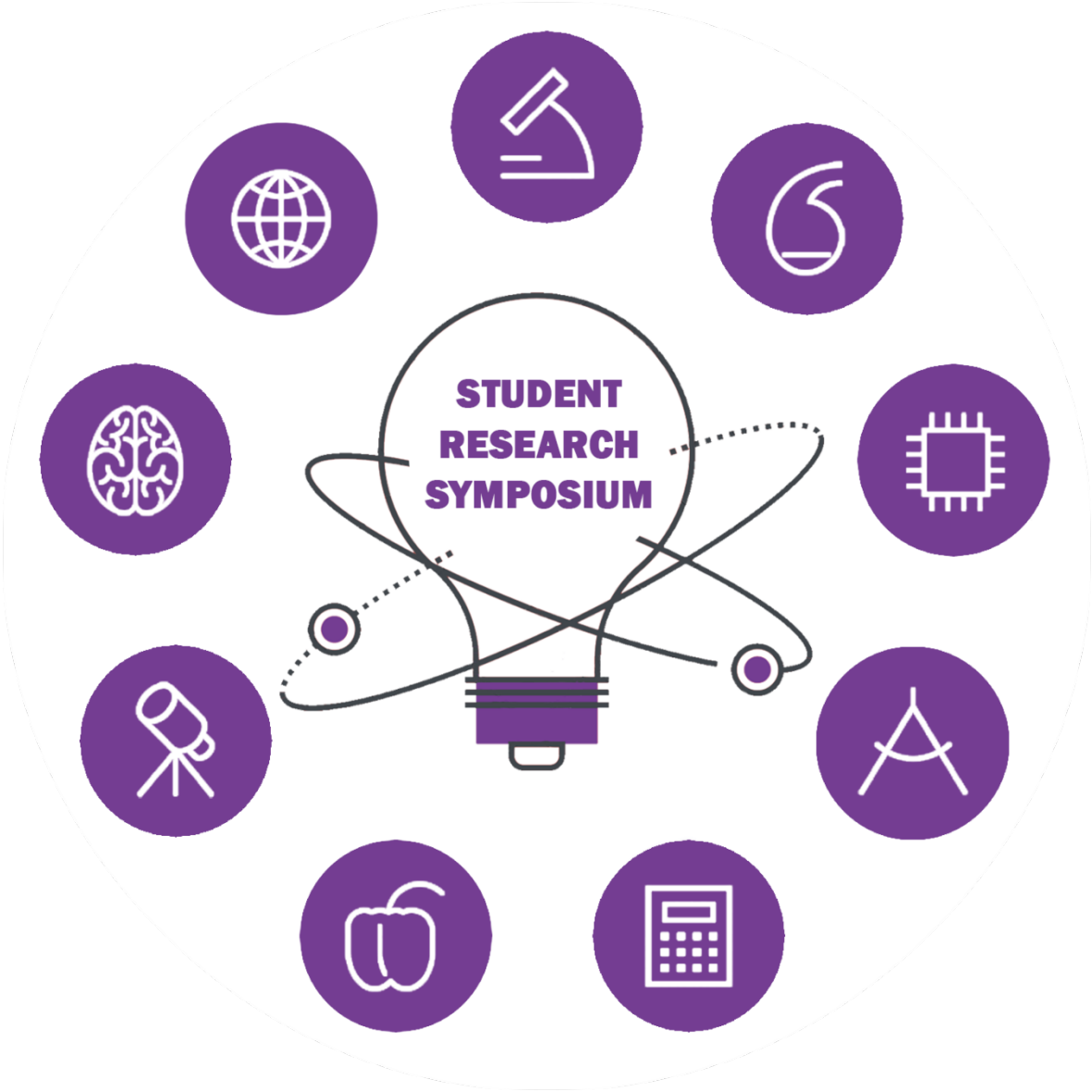

Developer's Manual v1.2

May 5, 2017

I. Revision Signatures

The following asserts that all team members have read the document and assert that the information contained within this document is complete and correct.

Name	Signature	Date
James Stewart		
Quang Nguyen		
Michael Giba		
Thaddeus Rix		
Son Nguyen		

II. Revision History

All revision history of this document is listed below.

Version	Changes	Edited Date
Version 1.0	• Initial draft	April 17th, 2017
Version 1.1	• Reformatted document structure • Added server setup	April 24th 2017
Version 1.2	• Revised wording of content	May 5th, 2017

Table of Contents

I. Revision Signatures i

II. Revision History ii

1. Introduction 1
   1.1 Purpose 1
   1.2 Project Overview 1
   1.3 Document Overview 1

2. Server Setup 2
   2.1 Apache 2
      2.1.1 Installation of Apache 2
      2.1.2 Configuration of Apache 2
   2.2 Web Application Setup (Django) 2
      2.2.1 Installing Python 2
      2.2.2 Installing Pip Package Manager 3
      2.2.3 Installing Project Python Dependencies 3
      2.2.4 Procuring Source from GitHub 3
      2.2.5 Verify Installation 3
   2.3 Database Setup 4
      2.3.1 Installing PostgreSQL 4
      2.3.2 Configuring PostgreSQL 4
      2.3.3 Starting the PostgreSQL Server 5
      2.3.4 Creating Database User 5
      2.3.5 Database Setup for Local Development 6
      2.3.6 Creating a Fresh Database for Local Development 6

3. Development Setup 6
   3.1 Getting an IDE 6
   3.2 Installing Dependencies 6
   3.3 Getting the Code 6
   3.4 Setting Environment Variables 7
   3.5 Debug Mail Server 7
      3.5.1 Why Is This Necessary? 7
      3.5.2 Launching the Mail Server 7
   3.6 Running and Changing Code 7
      3.6.1 Running the Project Locally 7
      3.6.2 Changing the Application Port 8
      3.6.3 Exposing the Application Through Public IP 8
      3.6.4 Making Changes and Testing 8
      3.6.5 Registering Changes with Source Control 8
   3.7 Updating the Production Server 9
      3.7.1 Pulling Changes from Source Control 9
      3.7.2 Updating Models if Necessary 9

3.7.3 Restarting Apache 10
4. Project Overview 11
4.1 Basic Django App Structure 11
4.1.1 Views.py 11
4.1.2 Models.py 11
4.1.3 Admin.py 11
4.2 Submitter Portal 11
4.3 Chair and Reviewer Portal 12
4.4 Admin Panel 12
5. Glossary of terms 13

1. Introduction

1.1 Purpose

This document serves to provide information and instruction to the future SRS developers on how the SRS website was made as well as procedures to maintain the website. Each technology requirements will be in a different part so that the reader can follow and know how to setup as well as update the system of the SRS website.

1.2 Project Overview

The Michael and Sally McCracken Student Research Symposium (SRS) is an event at TCU that invites undergraduate and graduate students to showcase their research projects to their colleagues and professors. The projects can be involved in any of the Science and Engineering disciplines at TCU, including interdisciplinary. The students participating will present their research projects inside of the Tucker Technology Center on a predetermined date.

TCU's old SRS site provided an outdated submission-review system for SRS. The previous system was mostly a front-end to a collection of manual procedures to receive, review, and present research projects. There was a growing need to make a more robust system that could provide smarter interfaces for various users that would allow for secure submitting, voting, and administrating.

The new system provides a host of automated processes that facilitate in the managing of the SRS event from year to year. This is possible due to a myriad of free technologies such as Django that allow for a more seamless experience participating in and managing SRS.

1.3 Document Overview

- **Section 2:** Server Setup
- **Section 3:** Development Setup
- **Section 4:** Project Overview
- **Section 5:** Glossary of Terms

2. Server Setup

2.1 Apache

Apache is a web server that sits in front of a web application to handle raw HTTP requests and performs analysis upon them before passing a response or delegating them to another process.

2.1.1 Installation of Apache

The installation of Apache is necessary so that a web server can focus on delegating application requests to Django and serve static assets and images to users.

```
> yum install httpd
```

2.1.2 Configuration of Apache

Once we have installed Apache we need to customize its configuration to make sure that it aligns with the structure dictated by the Django project. Edit the file in `/etc/httpd/conf/httpd.conf` or run the following command to properly configure Apache:

```
> wget https://goo.gl/SxkbWK >/etc/httpd/conf/httpd.conf
```

2.1.3. Starting Apache

Upon successful completion of these commands, making sure that all the assets and media directories containing content will be served by Apache, and have the permissions to be accessed by the web server. Finally start the web server service with:

```
> sudo service start httpd
```

2.2 Web Application Setup (Django)

2.2.1 Installing Python

Now we can begin to install the actual application code which will generate the SRS pages. The first program that you should get is Python, since Django requires Python to run. This project requires Python version 2.7, which you can get from https://www.python.org/downloads/. After the file is finished downloading, open it and follow the instruction to finish the installation.

2.2.2 Installing Pip Package Manager

We next require the installation of the package manager, pip, to facilitate dependency procurement. Open the command line on the target server, locate the directory you just downloaded the file to, and then run the following code:

```
> curl https://bootstrap.pypa.io/get-pip.py > get-pip.py
> chmod +x get-pip.py
> python get-pip.py
```

2.2.3 Installing Project Python Dependencies

Once pip has been installed, we can then install all other needed dependencies.

```
> pip install django==1.9.1
> pip install Pillow
> pip install psycopg2
> pip install reportlab
> pip install requests
```

Upon completion of dependency installation, change directories to `/opt/` on the server, or create the directory if it does not already exist. Then clone into the code repository hosted by GitHub with the commands in the following section.

2.2.4 Procuring Source from GitHub

To download the project source you will need to have access to the internet and you will need to be given repository access by the project's owner on GitHub. Once these requirements are met simply run:

```
> cd /opt/
> git clone https://github.com/liranmatcu/SRS
> mv SRS srs
```

2.2.5 Verify Installation

Verify the Django installation completed successfully by running a development server with the following commands.

```
> cd /opt/srs/
> ./manage.py runserver
```

If there are no errors because of these commands move on to installing the PostgreSQL database, otherwise please ensure you followed all the instructions above.

2.3 Database Setup

Django supports many different database servers, for example PostgreSQL, MySQL, Oracle and SQLite. In this project, we are going to use PostgreSQL version 9.6 for the database.

2.3.1 Installing PostgreSQL

The first thing that is necessary to install PostgreSQL is to pull the package down and install it using the built-in package manager yum. Run the following command:

```
> yum install postgresql-server
```

2.3.2 Configuring PostgreSQL

PostgreSQL needs a directory to store all its necessary data and metadata. If this directory doesn't already exist, we will need to create it and make sure that the permissions for the directory are correct. First, we will need to verify that the pgsql directory can be edited by the postgres user. To do this run the following command as an authenticated user.

```
> sudo chown -R postgres /usr/local/pgsql/
```

This will change the owner of the pgsql directory to be owned by the user, postgres. Now we will need to change to the user postgres to start the database instance. Run the following commands to change users:

```
> sudo su postgres
```

Now we will need to initialize the database data directory by running the following commands:

```
> export PGDATA=/usr/local/pgsql/data
> ./pg_ctl initdb
```

The screen should show some progress information and after a few seconds you should see a success message. To confirm that the initialization ran successfully, print the contents of the newly created directory.

```
> ls -l $PGDATA
```

2.3.3 Starting the PostgreSQL Server

Once you have successfully configured and installed PostgreSQL you can start the server. While you are still the user postgres, enter the following command:

```
> /usr/bin/postgres -D /usr/local/pgsql/data &
```

This will start the database server and assign the correct data directory. The trailing ampersand is used to indicate that the process should be ran in the background. Optionally, if you wish to use a more explicit command you can simply enter:

```
> /usr/bin/postgres -D $PGDATA &
```

2.3.4 Creating Database User

PostgreSQL relies on users to determine levels of access management. Currently, the Django server only expects one role which is a user named srs. You need to create a database user that matches the credentials listed in the settings.py file of the project. First connect to the PostgreSQL server using the following command

```
> psql
```

Then create a user:

```
$ CREATE USER srs WITH PASSWORD 'yourgreatpassword';
```

Then create the database:

```
$ CREATE DATABASE srs;
```

Finally, you need to grant the new user permission to access the database, srs's public schema:

```
$ GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO srs;
```

After entering these commands, you can close out of the psql shell, and switch back to the user srs. Finally, navigate to the Django project directory and run the following commands to sync the project schema with the PostgreSQL database.

```
> ./manage.py makemigrations
> ./manage.py migrate
```

2.3.5 Database Setup for Local Development

For development on local machines we recommend simply using a SQLite instance populated with test data. The current project git repository already contains a copy of a suitable SQLite database which will be connected to automatically if the project is run in development mode.

2.3.6 Creating a Fresh Database for Local Development

If no pre-existing database is available a new one can be created locally after successfully cloning into the SRS project by navigating to the project directory and by running the following command:

```
> ./manage.py makemigrations
> ./manage.py migrate
```

3. Development Setup

3.1 Getting an IDE

While it is possible to edit the code from the project using any editor such as vim, nano, emacs or Textedit, it is highly recommended to download a more sophisticated IDE. Some of the IDEs which we used for initial development are listed below.

- Sublime Text 3: https://www.sublimetext.com/3
- Atom Editor: https://atom.io/
- Visual Studio Code: https://code.visualstudio.com/

3.2 Installing Dependencies

The dependencies that are needed for local development are the same as what are needed for the server. Please install pip, then install the necessary dependencies as defined in section 2.2.2-3. If you do not wish to have the dependencies installed directly on your computer create a virtual environment as described here: https://virtualenv.pypa.io/en/stable/

3.3 Getting the Code

If you have not already cloned into the repository for SRS you will want to do that now. Simply follow the same steps outlined for the server in section 2.2.4 on your local machine. Once the code is properly pulled down, you can begin to run and make changes to the project.

3.4 Setting Environment Variables

To run the SRS project locally for testing and development, an environment variable called SERVER_DEBUG must be set equal to one. This can be done with the following command:

```
> export SERVER_DEBUG=1
```

This environment variable will cause the Django application to connect to a local SQLite database instead of trying to connect to a PostgreSQL instance. This means that you will not have to install and run a local PostgreSQL database in order to run the website locally on your machine. Furthermore, when the SRS project is in development mode it will serve static assets and media files itself, instead of requiring a web server like Apache.

3.5 Debug Mail Server

3.5.1 Why Is This Necessary?

While local development does not require Apache or PostgreSQL, one service is needed that is not required by the production implementation. This service is a local mail server. The reason a local mail server is needed is because in production, emails are dispatched through the TCU official SMTP server. This official SMTP server is only accessible by specific machines within the TCU network, and accordingly an average box will not have this permission. To allow development to continue in a distributed manner, when the project is running in development mode it will attempt to send its emails through a test mail server listening on port 1025.

3.5.2 Launching the Mail Server

To launch this new mail server, launch a separate terminal window and run the following command:

```
> python -m smtpd -n -c DebuggingServer localhost:1025
```

After running this command, leave this window running. As a bonus, any emails that are sent in development mode never get delivered to their recipient, but are visible with all metadata in the terminal window.

3.6 Running and Changing Code

3.6.1 Running the Project Locally

Open a fresh terminal window and navigate to the SRS project directory. If all the previous steps have been properly followed, you should be able to start up a simple development server by running the following command:

```
>  python manage.py runserver
> ./manage.py runserver #shorthand
```

3.6.2 Changing the Application Port

If you would like to run the development server on a different port than the default 8000, you can run the same command as the previous subsection (3.5.1), but with a trailing available port for example 8080:

```
> ./manage.py runserver 8080
```

3.6.3 Exposing the Application Through Public IP

If you are performing development on your local machine but would like to debug your application through a mobile phone or tablet, it may be helpful to expose the application via your public IP address. To do this simply run the command:

```
> python manage.py runserver 0.0.0.0:8000
```

If your internet router and firewall permit it, you should be able to open a browser on your device and access the development instance at:

```
> {{PUBLIC_IP}}:8000
```

3.6.4 Making Changes and Testing

Once the development server is running, you can make as many changes to code as you would like. If you change a page's HTML, you do not even have to restart the development server for the changes to take effect. If you make changes to portions of python code, you will either have to restart the development server manually, or simply wait a few seconds for Django to notice changes have occurred and wait for it to restart itself manually.

3.6.5 Registering Changes with Source Control

Once you have made various changes to the content of the python code or any static assets, you will want to save these changes and push them up to our centralized git repository. First, add any new files using the following commands:

```
> git add file_a.py file_b.py # adds specific files
> git add -A # adds the entire working directory
```

Once you have added the changed files to the repository, commit the changes with the command:

```
> git commit -am "Made some changes to the project"
```

At this point, all your changes to the project have been saved to the local git repository but still have not been reflected in the remote (or centralized) repository. First make sure to pull down any changes other users may have made to the project with the command:

```
> git pull
```

Then, assuming there are no errors, simply push up your changes with:

```
> git push
```

3.7 Updating the Production Server

3.7.1 Pulling Changes from Source Control

Now we will update the production server with the changes made during local development.
First connect to the server with either the command below or the correct ssh command for your particular distribution:

```
> ssh srs@srs.tcu.edu
```

After successful connection, verify the project has been previously downloaded to the server and that the production environment has been properly set up. Once this has been verified, pull the updated code down from the centralized git repository with the command

```
> git pull origin master
```

3.7.2 Updating Models if Necessary

If any changes were made to the database schema, definitions defined in any of the Django apps models.py files, then migrations, should be created and ran to update the server's PostgreSQL database. Enter the following commands from the project base directory:

```
> ./manage.py makemigrations
> ./manage.py migrate
```

3.7.3 Restarting Apache

Once all changes from local development have been properly migrated to the server at both the code and database levels, then we are ready to deploy the changes. Changes will be visible on the published site upon execution of the subsequent command:

```
>  sudo systemctl restart httpd
```

4. Project Overview

4.1 Basic Django App Structure

A standard Django application is comprised of various apps that make up subcomponents of the application as a whole. In the SRS project, there are three main apps, submission, review, and srs_admin. Each of these individual apps will be further analyzed later in the section. In order to form a better basis for understanding these apps, we will first need to understand the basic files that make up any general Django app.

4.1.1 Views.py

The views file in a Django app typically consists of functions which expect an HttpRequest object as an argument followed by URL keyword arguments if applicable. These functions are then expected to render, construct a view (usually through Django's templating engine), and return it via a HttpResponse object to the user.

4.1.2 Models.py

Django is built to utilize an amazing python library called SQLAlchemy. This library allows for you to define and access a relational SQL database schema using an Object-relational Mapper (ORM). The models.py file is meant to be where the relational database schema is programmatically defined. Subclasses of the Django built-in class Model are defined here which are automatically interpreted by the Django manage.py program as relational database schema definitions when the makemigrations and migrate commands are run.

4.1.3 Admin.py

The admin file in a Django app is meant to be the location in which connections between a specific model class and the admin panel are defined. Models can either be automatically "registered" with the admin panel so that they pop up for CRUD operations, or an explicit ModelAdmin subclass can be defined and attached to that model definition. A ModelAdmin subclass allows for detailed configuration of the appearance of a given model's appearance during CRUD operations in the admin panel.

4.2 Submitter Portal

The submitter will have a web interface with which they will register and sign in to their account, wherein they can submit their abstract, poster, and personal information. Here they will be able to update and monitor their submitted information. All the code for this section of the website resides in the Django app (subdirectory) submission which contains all logic pertaining to the following:

- Rendering submission related views - views.py
- Rendering the registration page - views.py
- Creating database entities for new submission accounts - various locations
- Defining the schema of submission related tables - models.py

4.3 Chair and Reviewer Portal

The chair will be provided an authorized account and credentials with which they can sign in to the web interface, get ballots for their department posters, and select the undergraduate and graduate winners of their department. All the code pertaining to these functions of the site reside in the review app of the Django application. This application contains logic pertaining to the following:

- Creating the ballots for the chairs and issuing them to the chair portal - ballots.py
- Rendering and filtering of submissions based on criteria - views.py
- Selecting winners, ballot downloading, chair sign-ins and sign-outs - views.py

4.4 Admin Panel

The code that generates the admin panel is located within Django's internal libraries, which are generally inaccessible. However, the models and admin actions which we have defined are located in various places throughout the project determined by their pertinence to the app they control. For example, some of the logic defined within this scope includes:

- Configuration of submission related admin actions and admin model views resides in: $PROJECT_ROOT/submission/admin.py
- Configuration of reviewer related admin actions and admin model views resides in: $PROJECT_ROOT/review/admin.py
- Configuration of beginning of year set up related admin actions and admin model views resides in: $PROJECT_ROOT/srs_admin/admin.py

5. Glossary of terms

Apache: Apache is an HTTP Server that is robust, commercial-grade, feature-rich, and freely available.

CRUD: Create, read, update and delete.

Django: Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

IDE: A software application that provides comprehensive facilities to computer programmers for software development.

ORM: ORM or Object-relational mapping and it refers to a technique in computer programming wherein the data is converted from differing incompatible type systems in languages that are object-oriented. Object-relational mapping and it refers to a technique in computer programming wherein the data is converted from differing incompatible type systems in languages that are object-oriented.

QA Testing: Quality Assurance Testing.

Virtual Environment: In the used context a virtual environment refers to a practice used in software development in which python dependencies are bundled and containerized to prevent unnecessary dependencies polluting a host machine.

Virtual Machine (VM): Operating system that is installed on software and imitates dedicated hardware.