Developer Guide

Version 1.0

May 05, 2017

Revision Signatures

By signing the following, the team member asserts that he/she has read the entire document and has, to the best of his knowledge, found the information contained herein to be accurate, relevant, and free of typographical error.

Name	Signature	Date
Cameron Diou		
Harrison Engel		
Steven Garcia-Renteria		
Rebecca Ruch		
Will Taylor		

Revision History

The following is a history of document revisions.

Version	Changes	Date
1.0	Initial Guide	05/05/17

Table of Contents

1 Introduction

1.1 Purpose

This document provides the information and processes required for a developer to work on the Scheduling Your Horizons (SYH) system and for a System Administrator to deploy and maintain the system.

1.2 Project Overview

Expanding Your Horizons Network (EYHN) is an organization that was founded to pique girls' interest in STEM fields. Every year, chapters of the organization host conferences around the globe where young girls watch presentations and participate in workshops led by women adult role models who are working in a STEM field. There are currently more than 80 conferences in 32 U.S. states, Europe, and Asia with up to 25,000 girls attending each year.

The Texas Christian University (TCU) Computer Science Department was originally approached in 2005 by the EYHN, Texas Wesleyan University (TxWes) branch to create a software solution for its scheduling and registration system. The system was well received and has been operational for the past eleven conferences. It is now out of date and can no longer be used. Scheduling Your Horizons (SYH) will replace this system while expanding upon the original functionality to allow user registration.

1.3 Section Overviews

The System Overview (Section 2) gives a basic overview of how the system is structured and some of the design philosophies of the system.

The System Requirements (Section 3) describes the system and environment requirements for developing the system and for deploying the system.

The Development Workflow (Section 4) describes the development workflow for making changes and adding features to the system.

The Production Deployment (Section 5) provides the information and processes necessary to deploy the application in a production environment.

The Application Guide (Section 6) gives an overview of the code structure and main classes of the application.

The Glossary of Terms (Section 7) provides a glossary of terms used.

1.4 References

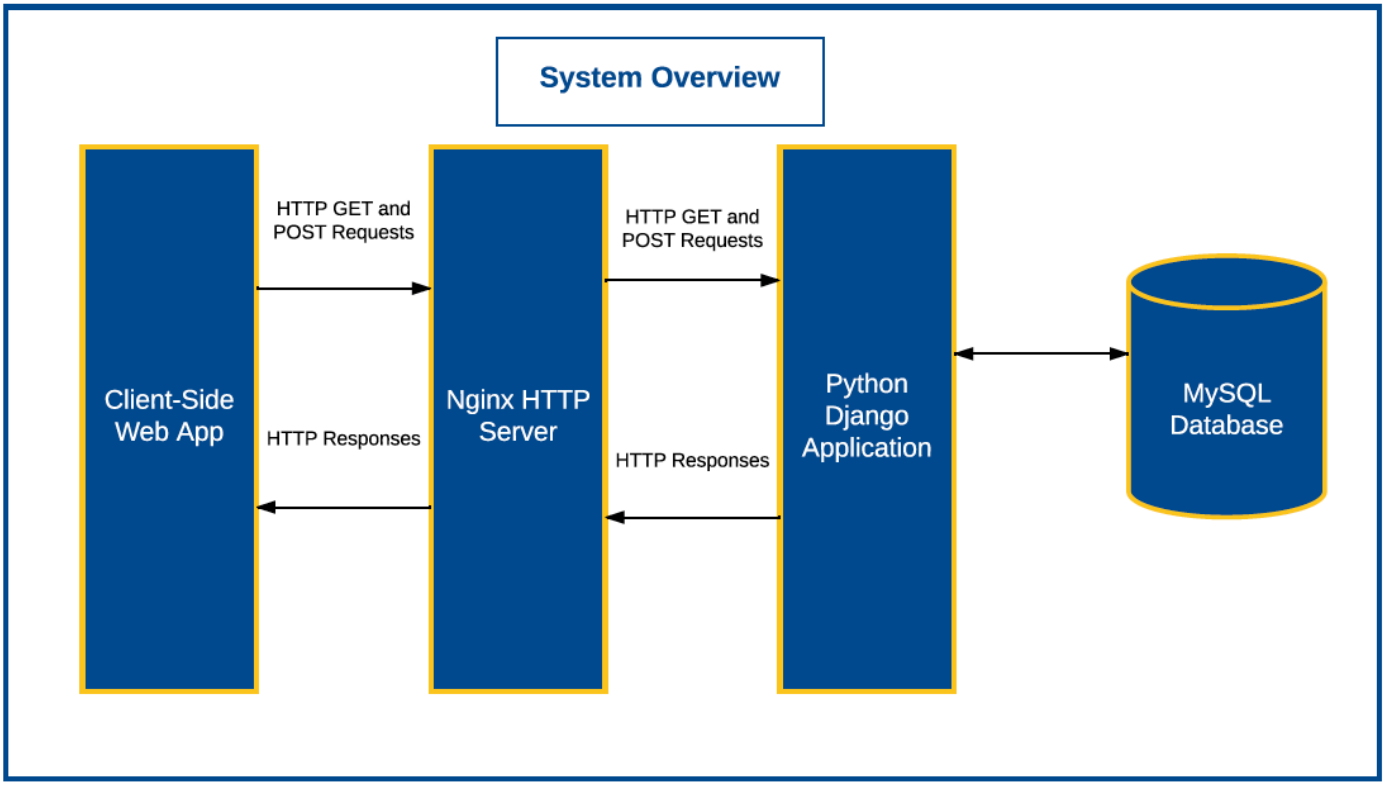
It is expected that developers who plan to make modifications understand Python 3, the Django framework, Nginx, Passenger, and Docker. Links to these technologies can be found below.

Python 3: https://www.python.org/doc/
Django: https://www.djangoproject.com/
Nginx: https://www.nginx.com/
Passenger: https://www.phusionpassenger.com/library/
Docker: https://www.docker.com/

2 System Overview

This section gives a high-level overview of the components of the system and where to look in this document to find out more information about each component.

Overview of System Architecture: The architecture of the system consists of five major pieces: the Django web application, the Scheduling library, a Passenger app server, an Nginx HTTP server, and a MariaDB database all residing on Docker images.



Django Web Application: The Django Web Application contains the logic of the application and is the core piece of the system. This system is described in depth in **Section 6**.

Scheduling Library: A Python library that provides the scheduling algorithms used by the web application for scheduling the event. This library is independent because it does not necessarily rely on the Django Web Application to function and could be used in other similar scheduling applications.

Passenger App Server: Passenger is used as an intermediary between the Nginx HTTP server and the Django Web Application.

Nginx Http Server: Nginx is a very fast HTTP server often used as a reverse proxy server.

MariaDB Database: MariaDB is the open source version of MySQL. It's meant to be a stand in replacement for MySQL, meaning that the application doesn't need to distinguish between a MySQL database and a MariaDB database. The drivers used by the Django Web Application to access the MariaDB database also work for MySQL databases. The MariaDB database schema is described using Django's Models framework. For the database schema see **Section 6.4.3.**

Docker: Docker is used to containerize the application. It makes managing the environment easier and helps standardize deployment.

3 System Requirements

This section details the system requirements for installing and running the application.

3.1 Hardware Requirements

There are no specific hardware requirements, although it is recommended to have at least 156GB of disk space and 4 GB of RAM.

3.2 OS Requirements

As a Docker container, this system can run on any operating system that supports later versions of Docker (17+). At the time of this writing it includes most Linux servers and desktop systems, macOS, and Windows 10. The instructions in this developer guide are written from the perspective of developing on macOS and deploying to a Linux server.

Platform	Docker CE x86_64	Docker CE ARM	Docker EE
Ubuntu	✓	✓	✓
Debian	✓	✓	
Red Hat Enterprise Linux			✓
CentOS	✓		✓
Fedora	✓		
Oracle Linux			✓
SUSE Linux Enterprise Server			✓
Microsoft Windows Server 2016			✓
Microsoft Windows 10	✓		
macOS	✓		
Microsoft Azure	✓		✓
Amazon Web Services	✓		✓

(Source: https://docs.docker.com/engine/installation/#supported-platforms)

4 The Development Environment

This section details how to set up the environment required to run the application. Since the application is containerized using Docker it can be installed and developed on any operating system that supports Docker (see **Section 3**).

The **Environment Overview** subsection describes the system, applications, and libraries required to run the application.

The **Development Environment Setup** subsection describes how to set up the development environment (with and without Docker) on macOS systems. This setup is good for rapid development and testing.

The **Development Workflow** section describes the basic development workflow used to make changes to the application.

4.1 Environment Overview

4.1.1 Application Image

The Docker Image of the application defines the environment, operating system, and files available to the application. It is constructed using the following layers.

4.1.1.1 Base Image: Passenger-full

- Source: https://hub.docker.com/r/phusion/passenger-full/
- Description: The base image of the main application image is passenger-full. It is an **Ubuntu 16.04** image with Passenger, Nginx, and Python already pre-installed along with some other useful tools. See https://hub.docker.com/r/phusion/passenger-full/ for more information.

4.1.1.2 Configuration Files

- Source: */nginx* folder
- Description: The files used to configure the Nginx Server, Passenger, and Django Application.

4.1.1.3 Application Code

- Source: */web* folder
- Description: The Python code for the main application. It is stored under the /app/webapp folder of the image.

4.1.1.4 MySQL Client Libraries

- Source: libmysqlclient-dev package
- Description: This package contains binaries used by the application to act as a client of the MariaDB database.

4.1.1.5 Python Packages

- Source: PyPy repository. The packages installed are specified in the **requirements.txt** file.
- Description: The application needs various Python modules. These modules are installed directly into the Docker image during build time.

4.1.1.6 Base Image: mariadb

- Source: https://hub.docker.com/_/mariadb/
  Description: The official base image for MariaDB. It comes with MariaDB pre-installed. The MariaDB database resides on a separate image from the main application.

4.1.2 Host Environment

The host environment can vary greatly since this is a Docker application. See **Section 3** for the operating systems supported at the time of this writing. For an up to date list of the supported operating systems see https://docs.docker.com/engine/installation/#platform-support-matrix .

4.2 Development Environment Setup

Development on the application was done on macOS systems. The following describes how the development environment was set up:

1) Install Git
   - Description: Git is the main version control system used in the development of the application. For more information on using Git see https://git-scm.com/ . There are several ways to use Git: built-in IDE functionality, GUI applications like Source-Tree, and the Git CLI (Command Line Interface). Developers are free to choose how they interact with the Git repository, but we highly recommend using the raw CLI. It is more difficult to learn but also offers more control and transparency.
   - Installation:
     - If you are running OSX 10.9 or greater then simply run the command "`git`". This initiates the install process using XCode tools.
     - For installation instructions on older versions of macOS, see https://git-scm.com/book/en/v2/Getting-Started-Installing-Git .
2) Install Python 3 and Pip
   - Description:
     - Python 3: At the time of this writing the default Python version installed on macOS is Python 2.7. This system uses Python 3, so it needs to be installed manually.
     - Pip: Pip is the package manager for Python. It is used to install, update, and manage python packages.
   - Installation:
     - We've found the best way to manage many development packages on macOS is through Homebrew. If you do not have Homewbrew installed see https://brew.sh/ for installation instructions.
     - Once you have Homebrew installed, run "`brew install python3`". This will install both Python3 and Pip.
3) Install Virtualenv
   - Description:
     - Virtualenv is used to manage simple "virtual environments" for Python projects. It will allow you to have different versions of packages used on different projects running in the same system. For more information on Virtualenv see https://virtualenv.pypa.io/en/stable/.
   - Installation:
     - With Pip installed (step 2) run "`pip install virtualenv`".
4) Install MySQL and MySQL Client Libraries
   - Description:
     - Although the production application uses MariaDB, either MariaDB or MySQL can be used interchangeably. For the development environment, we installed a local version of MySQL since MySQL had better support on macOS than MariaDB at the time of development.
   - Installation:
     - With Homebrew installed (Step 1) run "`brew install mysql`".

5) Get the Application Code
   - Description
     - The code for the application while developing this project was stored in a Git repository using GitHub. This repository may not be available after Spring 2017, and instead you may need to download the code from a CD, flash drive, or other storage system.
   - Installation
     - Using GitHub:
       - Open the terminal and navigate to the directory where you want to store the code.
       - Run: "`git pull` https://github.com/slick9115/SchedulingYourHorizons.git"
     - From Another Storage Source
       - If installing from another storage source, copy the root directory of the project to the directory you plan to work from on your machine.
6) Create the Application Virtual Env
   - Description
     - Setting up a virtualenv is good practice when developing Python applications. It helps minimize dependency issues when working on multiple projects on the same machine.
   - Installation
     - Open the terminal and navigate into the root directory of the application, wherever it was installed on your system in step 5.
     - From the command line run "`virtualenv venv`". This will create a folder called "venv" in the root directory.
7) Install Python Dependencies
   - Description
     - The application requires a handful of Python dependencies that are defined in the **requirements.txt** file.
   - Installation
     - Open the terminal and navigate to the root directory of the project.
     - Run "source venv/bin/activate". This activates the virtualenv for this project.
     - Next run "`pip install -r web/requirements.txt`". This will install all required Python packages.

4.3 Development Workflow

This section describes the development workflow used by the team for developing this application. This already assumes all the steps in Development Environment Setup (Section 4.2) have been followed.

1) Set Up a Test Database
   - Description
     - Each developer uses their own test database. If you already have a working test database installed on your development machine you may skip this step.
   - Instructions
     - Navigate to the root directory of the project.
     - Turn on virtualenv with the command "`source venv/bin/activate`"
     - Run "`python3 manage.py makemigrations`"
     - Run "`python3 mange.py migrate`"

2) Turn on test Server.
   - Description
     - Running the test server allows the developer to immediately see the effects of changes made to the code.
   - Instructions
     - Navigate to the root directory of the project.
     - Turn on the virtualenv with the command "`source venv/bin/activate`"
     - Turn on the Django server by running the command "`python3 manage.py runserver`"
     - Visit http://127.0.0.1:8000/signup/student and verify that the student signup page is displayed.

3) Commit Changes
   - Description
     - After making some desired changes to the code you commit those changes using Git. For more information on using Git see https://git-scm.com/book/en/v2/Getting-Started-Git-Basics .
   - Instructions
     - Add your changes using "`git add [file 1] [file 2] …`"
     - Commit your changes using "`git commit`"

4) Push Changes to the Git Repository
   - Description
     - After committing changes, you push them to the shared repository so that teammates can pull the changes. This step is optional and requires your team have a git repository.
   - Instructions
     - Run "`git push origin [name_of_branch]`"

5 Production Deployment

This section outlines how to deploy the application to a Linux server. The following steps are all run from the Linux server you plan to deploy to. They assume you have already downloaded the code related to the application, which should be provided to you on a CD or flash drive.

1) Install and Start Docker
   - Description
     - The application runs on Docker. For more information on Docker see https://www.docker.com/.
   - Installation
     - Installation is slightly different on each operating system. For your specific system please see the documentation at https://docs.docker.com/engine/installation/ .
     - Make sure that the installed Docker version is >= 17
       - To check your Docker version run "`docker version`" from the command line.
     - Make sure that docker-compose is also installed. At the time of this writing, docker-compose comes installed with Docker, but in future releases this may not be the case.
       - Try running "`docker-compose`" from the command line. If the command is not found install it using the instructions at https://docs.docker.com/compose/install/.
2) Install Systemd
   - Description
     - Systemd is used to manage daemon processes, and most importantly in this case, restart Docker and the application if the server is reset or updated.
   - Installation
     - Often modern Linux systems come with Systemd pre-installed. If not, contact your system administrator to have it installed on the server.

3) Navigate to the Application Root
   - Description
     - The startup script needs to be run from root directory of the application.
   - Directions
     - Navigate to the root directory of the project through the "`cd`" command.

4) Run the Start Script
   - Description
     - The logic to start the application is contained within the file **/scripts/run_application.sh**
   - Installation
     - Make sure you execute the command below from the root directory of the application.
     - Run "`sudo bash ./scripts/run_application.sh`". Make sure this command is run from the root directory of the project.
     - If this command has not been run before it will take an extended length of time to build the application image.

6 Application Guide

6.1 Overview

This section gives a high-level overview of the objects, files, and major components of the Django application.

The **System Architecture** (Subsection 2) describes the overall system architecture.

The **Settings and Configuration** (Subsection 3) describes the different files used for overall application configuration.

The **Registration** (Subsection 4) contains an overview of the major classes and functions involved in the Registration application.

The **Scheduling and Report Generation** (Subsection 5) describes the classes and files involved in the Scheduling and Report Generation application.

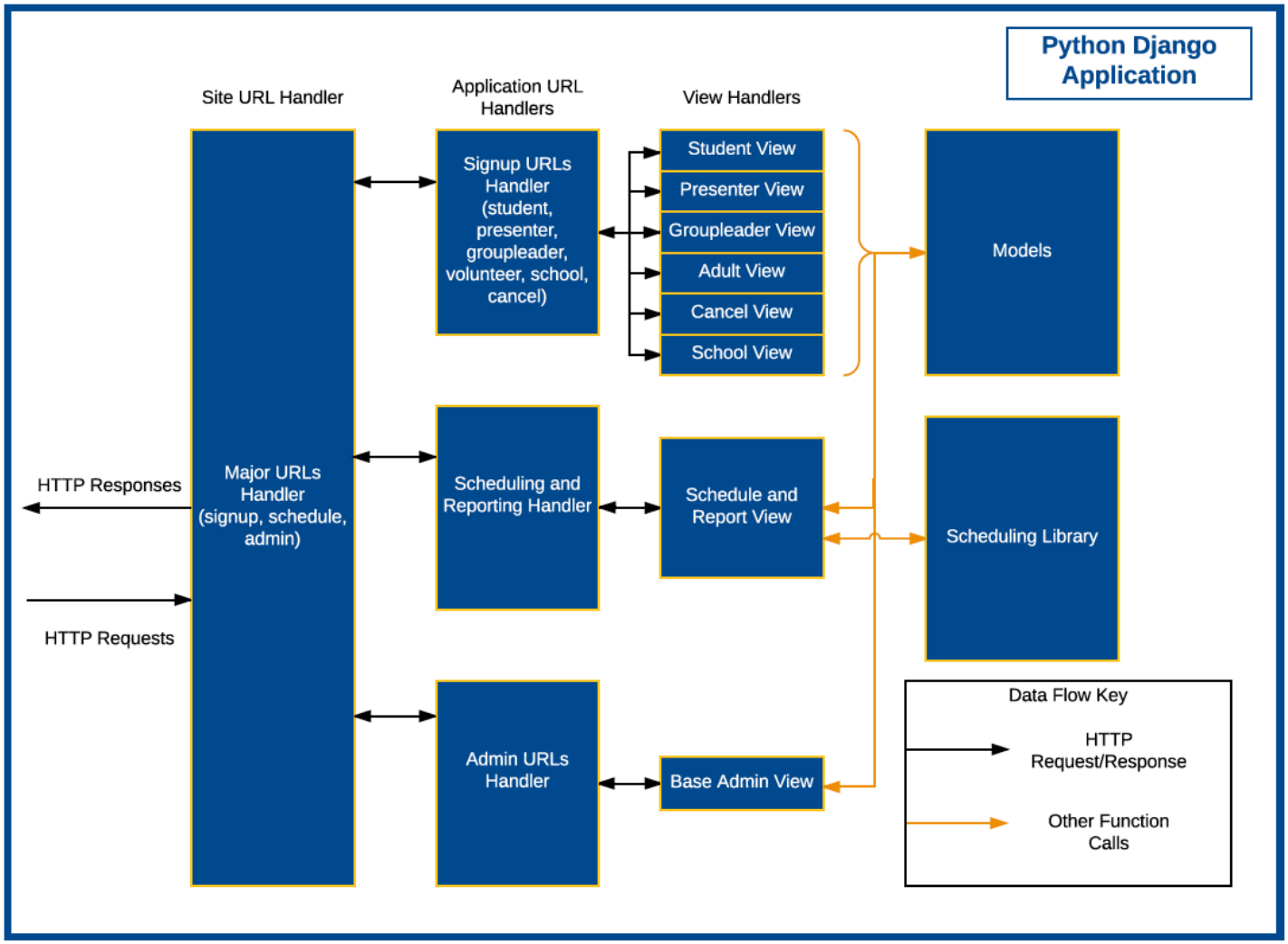
The **reCAPTCHA Implementation** (Subsection 6) describes the reCAPTCHA implementation used throughout the system.

6.2 System Architecture

The application will follow the standard Django architecture. The overall site URL handler will receive HTTP requests and direct them to the appropriate application based on the request URL. The three applications are:

- Signup: (/signup) The signup application will contain forms that Participants will fill out to sign up for the conference.
- Schedule: (/schedule) The schedule application will allow Administrators to generate and modify the conference schedule, download reports, and reset the system for new registration.
- Admin: (/admin) The admin application will allow Administrators to access and modify the database.

Once a request is routed to the correct application, the application level URL handler directs the request to the correct view object. These view objects interact with various model objects that act as the interface between the Python code and the database. The schedule and report view object makes use of our scheduling library via function calls to generate the schedule.



6.3 Settings and Configuration

There are a handful of global settings and configurations that affect the Django application, the server configuration, and the overall operation of the system. These settings are defined through either environment variables or through settings files.

6.3.1 Environment Variable Configuration

The following are environment variables used by the application. The standard Docker configuration comes with these values pre-set within the **docker-compose.yml** file. They should be changed within this file, not on the host server.

- SYH_KEY_FILE: The path of the key file within the system. This variable should be automatically set within the **docker-compose.yml** file through the run-application.sh script. See **Section 6.3.2.1** for a description of the key file.

- DJANGO_SETTINGS_FILE: The location of the **settings.py** file within the Docker container. Unless the directory structure of the application is changed this should always be “src_django.settings.settings”.

- SYH_PROD: Defines what mode the application will run in. If it is set to “True” then the application will run in production mode, using the **prod.py** file for settings. Otherwise it will run in production mode, using **dev.py** for settings. In the **docker-compose.yml** file this is automatically set to “True”.

- PYTHONPATH: Defines where within the container Python searches for modules. It must contain the path to the root of the */web* directory of the application. It is already set in **docker-compose.yml** and doesn’t need to be changed unless the directory structure of the Docker container is changed.

- APP_HOST: Specifies the canonical host name of the application used throughout the application and within the HTTP server configurations. This variable must be set to the fully qualified domain name as created on the DNS. For example, if the clients access the application by going to https://cscdevprod04.txwes.edu then this variable must be set to “cscdevprod04.txwes.edu”. This environment variable is automatically set when running **run-application.sh**.

6.3.2 File Configuration

The system has been created to use configuration files that affect operation where possible. This subsection summarizes those files and some of the more important configuration options they control.

6.3.2.1 Keyfile.txt

The **keyfile.txt** file is used to pass important and sensitive information to the application. It should reside outside of the Docker container and be modified by the administrator deploying the application.

The format of the key file consists of lines where each line represents a different key-value pair. The format is key:value, with no whitespaces. The default **keyfile.txt** is in the correct format with dummy values.

The following are the keys that are required for the **keyfile.txt** file along with descriptions about each key:

- SECRET_KEY: The secret key of the Django application. This key is used to provide cryptographic signing within the application. For more information on Django's built in cryptographic signing see https://docs.djangoproject.com/en/1.10/topics/signing/ . For more information on the SECRET_KEY Django setting see https://docs.djangoproject.com/en/1.10/ref/settings/#std:setting-SECRET_KEY
- CAPTCHA_SECRET: The secret key used to securely communicate with Google during reCaptcha authentication in production mode. For more information on the reCAPTCHA implementation see Section 6.6.
- CAPTCHA_PUBLIC: The public key used for reCAPTCHA authentication. For more information on reCAPTCHA see Section 6.6.
- DB_NAME: The name of the database the application is connected to. This is used by the application to access the database.
- DB_USER: The user name that the application will use to access the database.
- DB_PASSWORD: The password of the DB_USER that the application will use to interact with the database.
- DB_HOST: The host of the database. If the database resides on the same server use "localhost".
- DB_PORT: The port that the application will use to connect to the database.

6.3.2.2 Django Application Settings

Settings specific to the Django application are all specified within the */web/src_django/settings* directory. This folder contains settings that a developer or system administrator with Django experience could modify to customize the application.

The Django application operates differently when running in production mode vs. running in development mode. To distinguish between these two cases, the main settings file (**/web/src_django/settings/settings.py**) doesn't itself contain any variables the way a standard **settings.py** file does in Django. Instead, it determines the mode (production or development) that the application will run in and then loads the settings accordingly. It does this by checking the value of the SYH_PROD environment variable. If this variable is set to "True", then the settings for production mode are loaded from the file **/web/src_django/prod.py**. Otherwise the settings for development mode are loaded from the file **/web/src_django/dev.py**.

For a complete description of all Django defined settings variables, see https://docs.djangoproject.com/en/1.10/ref/settings/ .

6.4 Registration

The Registration application is responsible for all tasks related to registering participants for the event. It is not completely logically separate from the rest of the application, but most of its functions are independent of the operations inside the Scheduling and Report Generation and Admin applications.

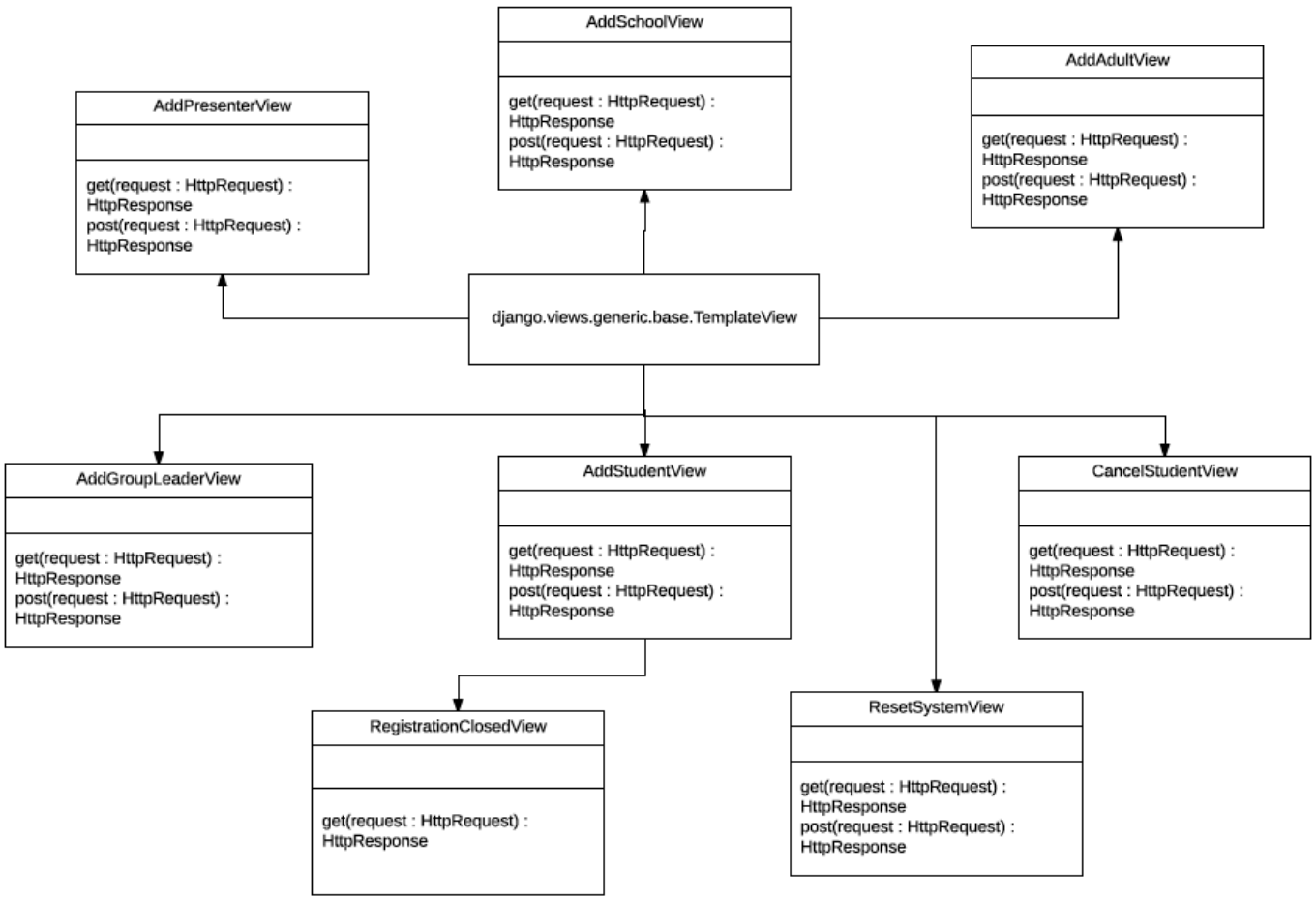
6.4.1 URL Mapping

URL mapping in the global application is defined through the **/src_django/urls.py** file. All requests through /signup/{form or page} are redirected to the Registration application. Below are the possible URLs the application will respond to.

URL Relative to Fully Qualified Domain Name	View Class (see Class Diagram)
/signup/student	AddStudentView
/signup/groupleader	AddGroupLeaderView
/signup/presenter	AddPresenterView
/signup/volunteer	AddAdultView
/signup/cancel	CancelStudentView
/signup/reset	ResetSystemView

6.4.2 View Classes

The main logic of the application is contained in the view classes. These are subclasses of Django's built in TemplateView class.



6.4.2.1 AddStudentView

- Description:
  - Controls the logic for registering Students for the event.
- File: **/web/registration/views/add_student_view.py**
- Class Methods
  - get: Overrides the TemplateView get method. It requests the Topics and Schools from the database, uses them to template the student registration form (**student-registration.html),** and then returns the templated response. If registration is closed, it redirects to the RegistrationClosedView.
  - post: Overrides the TemplateView post method. It handles interpreting the post request, attempting to add the student, and redirecting to the appropriate success or failure page.

6.4.2.2 AddGroupLeaderView

- Description
  - Controls the logic for registering Group Leaders.
- File: **/web/registration/views/add_group_leader_view.py**
- Class Methods
  - get: Overrides the TemplateView get method. It doesn't require any database access and returns the Group Leader registration page (**group-leader-form.html**).
  - post: Overrides the TemplateView post method. It handles interpreting the post request, attempting to add the group leader, and redirecting to the appropriate success or failure page.

6.4.2.3 AddPresenterView

- Description
  - Controls the logic of registering Presenters.
- File: **/web/registration/views/add_presenter_view.py**
- Class Methods
  - get: Overrides the TemplateView get method. It requires database access to retrieve Workshops and Topics. It returns the Presenter registration page (**presenter-registration.html**).
  - post: Overrides the TemplateView post method. It handles interpreting the post request, attempting to add the presenter, and redirecting to the appropriate success or failure page.

6.4.2.4 AddAdultView

- Description
  - Controls the logic for registering Adult Participants.
- File: **/web/registration/views/add_adult_view.py**
- Class Methods
  - get: Overrides the TemplateView get method. It does not require database access. It returns the Adult Participant registration page (**adult-registration.html**).
  - post: Overrides the TemplateView post method. It handles interpreting the post request, attempting to add the adult participant, and redirecting to the appropriate success or failure page.

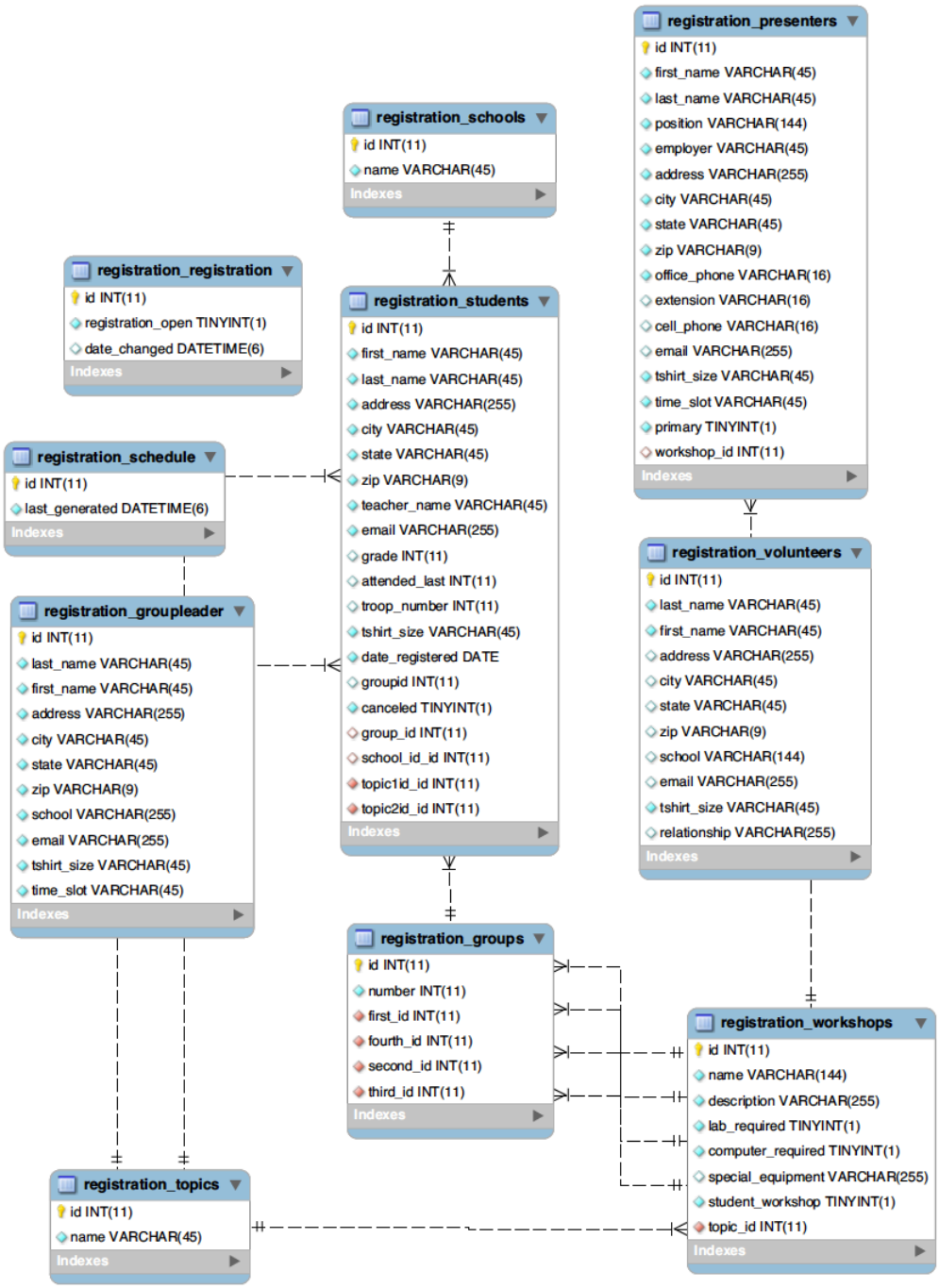
6.4.2.5 CancelStudentView

- Description
  - Controls the logic for canceling students.
- File: **/web/registration/views/cancel_student_view.py**
- Class Methods
  - get: Overrides the TemplateView get method. It does require database access to check that the unique URL used to access the page corresponds to an actual Student. It returns the student cancellation page (**student-cancellation.html**).
  - post: Overrides the TemplateView post method. It handles interpreting the post request, attempting to remove the student, and redirecting to the appropriate success or failure page.

6.4.2.6 RegistrationClosedView

- Description
  - Used to display the registration closed page (**registration-closed.html**)
- File: **/web/registration/views/registration_closed_view.py**
- Class Methods
  - get: Overrides the AddStudentView get method. It does not require database access. It returns the registration closed page. (**registration-closed.html**)

6.4.3 Database Schema



6.5 Schedule and Report Generation

The Scheduling and Report Generation application is responsible for generating and displaying the schedule, generating reports, and provide access to the system reset.

6.5.1 URL Mapping

URL mapping in the global application is defined through the /src_django/urls.py file. All requests through /schedule are redirected to the Schedule and Report Generation application. This makes use of the login_required method, which forces this page to require a login.

6.5.2 View Classes

6.5.2.1 ScheduleReportView

- Description
  - The only view the Schedule and Report Generation application.
- File
  - /web/schedule/views/schedule_report_view.py
- Class Methods
  - get: Overrides the TemplateView get method. It requires database access to retrieve Workshops and Topics. It returns the main Scheduling and Registration page (report-gen.html).
  - post: Overrides the TemplateView get method. It parses the post request and directs the request to generateNewSchedule, gnerateReports, moveStudent, addRemoveStudent, or printSchedule.
  - generateNewSchedule: The method called when the user requests to generate a new schedule. It accesses the database, generates the schedule, saves it to the database, and returns schedule and report view page with the updated schedule.
  - generateReports: The method called when the user requests reports. It parses out the requested reports from the post request, generates the appropriate reports, and returns an Http request containing a zip file with the requested reports.
  - moveStudent: This method is called when the user requests to move a student from one group to another. It attempts the move, saves the change to the database, and updates the page to show the change.
  - addRemoveStudent: The method called when a user requests to add or remove a student. It attempts to add or remove the student, saves the change to the database, and the updates the page to show the change.
  - printSchedule: The method called when a user requests to download the schedule. It uses the generateNewSchedule method to do this and returns a zip file containing the schedule.
  - getFirstPercent: An auxiliary method used to get the number of students who get to see their first choice as a percent.
  - getFirstAndSecondPercent: An auxiliary method used to get the number of students who get to see their first or second choice as a percent.

6.6 reCAPTCHA Implementation

The reCAPTCHAs were implemented to prevent bots from spamming the registration forms. The reCAPTCHA implementation contains code on both the client and server side. The implementation of the reCAPTCHA requires both a site key and secret key, which has been provided by Google (see Section 6.3.2.1 to see where these keys are specified). The current implementation uses secret and site keys issued to the Google account eyhconfirmation@gmail.com.

6.6.1 Client-Side Implementation

Each registration form lists Google's reCAPTCHA JavaScript code as a dependency in the form's HTML. This code can be found at https://www.google.com/recaptcha/api.js. A div with the class "g-recaptcha" is placed in the HTML to represent the reCAPTCHA. This div also contains a "site key" provided by Google that allows Google to identify which application is using the reCAPTCHA. This key is unique to each development environment and is stored in Django's settings module. When the user submits the form, the form's JavaScript checks to see if the reCAPTCHA has been completed by checking to make sure the key is not an empty string. If the key is an empty string, an alert telling the user to complete the reCAPTCHA appears. If the key is not empty, the JavaScript then puts the key into the value of a hidden, uneditable input field with the id "captcha" so the key can then be passed in the POST request to the Django server.

6.6.2 Server-Side Implementation

The server-side implementation of each registration's reCAPTCHA can be found in the registration form's corresponding view file. First, a POST request is sent to Google (URL https://www.google.com/recaptcha/api/siteverify) which contains both the applications secret key and the key provided by the client side's web browser. Google's servers then send back a response in JSON format. The JSON response will contain a Boolean titled "success" indicating whether the reCATPCHA has successfully validated the user. If the user was validated, then we continue the post function as normally. Otherwise, we render an error page to the user and return out of the function.

7 Glossary of Terms

Term	Phrase / Definition
EYH	Expanding Your Horizons
STEM	Science Technology Engineering and Math
SYH	Scheduling Your Horizons
TCU	Texas Christian University
TxWes	Texas Wesleyan University
WSGI	Web Server Gateway Interface. Used to connect an http server and the Django application.
reCAPTCHA	User validation tool provided by Google. For more information see https://www.google.com/recaptcha/intro/.
View Class	Python class that connects the frontend forms, databases, and models. For more information see https://docs.djangoproject.com/en/1.11/topics/class-based-views/.
Git	Free version control software (VCS) application. For more information see https://git-scm.com/.