GONZAGA UNIVERSITY
School of Engineering and Applied Science
Center for Engineering Design and Entrepreneurship

PROJECT PLAN
10/19/2018

YesWorkflow Web Components

Prepared by:

Mackenzie Brown	Brooke Huntington
Anthony Niehuser	Brewer Slack
Carl Lundin	

Reviewed by:

Shawn Bowers Faculty Project Advisor	Timothy McPhillips Project Sponsor/Liaison
Michael Herzog Design Advisory Board Member	

1 Project Overview

1.1 Project Summary

YesWorkflow is a software tool that allows researchers and scientists to visualize scripts through the use of scientific workflows. First, a scientist adds specialized YesWorkflow comments called ‘annotations’ to their workflow scripts. YesWorkflow then uses these annotations to generate a workflow that outlines how the data interacts. However, YesWorkflow does not currently have a framework for collecting data provenance (the process of tracing and recording the origins of data and its movement) or a user interface to visualize and manage workflows easily.

Our project is to create a Django web interface that can visualize and organize YesWorkflow output. The web interface will allow users to create an account and upload, edit, or delete their workflows. Users will also be able to access other users’ workflows from the web interface. Each workflow can be versioned and each version has associated data inputs and outputs. Users have the option to query workflows by date or with a general search feature that allows users to search over all workflow information, including title, tags, and authors.. In addition to the web interface, our project will allow users to use the command line to save YesWorkflow output data. A Django web interface gives users better access to their workflows, provides an opportunity for expanding and creating tools related to YesWorkflow, and a user interface to visualize data provenance.

1.2 Project Objectives

The primary outcome of this project is to increase the use of YesWorkflow. This will be achieved by creating a web based front end in order to enable access to data provenance from YesWorkflow. To that end, we will create a relational database to store information relevant for maintaining provenance of workflows. A Django website will utilize the database and serve information to the user relevant to workflows they want to see. This information includes version sets, data sets, dates, descriptions, and tags.

A secondary objective of this project is to create a system that works with any type of data science project. SKOPE is a project that will rely on YesWorkflow for data provenance. SKOPE will serve as a proof of concept for this project, and, if our project is successful, SKOPE will show concrete improvement through it’s adoption of the improved YesWorkflow tool set.

1.3 Project Stakeholders

1.3.1 Senior Design Group
We are a group of students studying Computer Science at Gonzaga University. Our role in this project is that of developers and project managers. While we will consult with our other stakeholders, we are the main driving force behind this project and we are responsible for its implementation, design, delivery, and overall outcomes.

1.3.2 Sponsors

The sponsors for this project are Professor Shawn Bowers and Dr. Timothy McPhillips. Dr. McPhillips is the primary contributor to YesWorkflow and as such we are helping him to achieve his goals on the project. Dr. McPhillips and Dr. Bowers will be the stakeholders we consult the most frequently and whose input has the largest weight. The success of this project is heavily influenced by the goals they created when sponsoring this project.

1.3.3 Faculty Advisory

Our faculty advisor is Professor Shawn Bowers. He will be our main point of contact in regards to how we can best organize our time in order to have a successful project. He will also be important for receiving feedback on our design and implementation. Professor Bowers is a long time faculty member and has previously advised several senior design projects. He will provide project management experience for our group as we have minimal experience with large scale projects.

1.3.4 Design Advisory Board Member

Our ‘DAB’ Member is Michael Herzog. Mr. Herzog is an experienced project manager at the company Itron in Liberty Lake, WA. We will be sharing our design plans and presenting to him for feedback. Mr. Herzog’s role is to help our group approach this project in an efficient and effective manner.

1.3.5 Target User Community

The target user community is the general data science community who would gain from visualizing their scientific workflows. These professionals are scientists that rely on software to help them collect and interpret data. The ability to graphically visualize workflows and keep track of data provenance is useful to these scientists so that they can collaborate without spending a significant amount of time attempting to figure out the techniques of their colleagues By gaining the ability to efficiently analyze old workflows, scientists can compare them to newer workflows and improve their processes. This allows them to get an overview of their workflows when publishing their work.

1.3.6 SKOPE

The SKOPE project is intended to enable “scholars to easily discover, explore, visualize, and synthesize knowledge of environments in the recent or remote past.” SKOPE intends to use YesWorkflow for the data provenance portion of the project.

1.4 Project Deliverables

1.4.1 Intuitive User Interface

Creating a user interface that simplifies the workflow analysis process is paramount to this project. Workflows can become complex, especially when including data provenance. Scientific workflows can go through hundreds of iterations with new variables and data in each iteration. The ability to to quickly analyze a workflow without becoming an expert in a field will be a metric that measures the success of this project. The user interface will make requests from the backend web API to decouple data management from the user interface.

1.4.2 Database Model

The database model will handle the data provenance implementation in YesWorkflow. The backend web API will interface with data stored in this database. Our model will be seamless and efficient as to not interfere with other tasks that a user might have. Managing the file system and mitigating redundancies are essential to our database model. Currently, YesWorkflow output is not persistent, so our goal is to create a save function that will transfer the output of YesWorkflow to our database so multiple workflows can be tracked.

1.4.3 Django Backend Web API

The web server that contains the tools and logic that enable access to provenance will be packaged in a Django web server. By doing this we create a distributable solution that can be altered and used by anyone. For example, an individual scientist can run this server on their local computer to help them maintain their personal workflows. Alternatively, a university could maintain a server so that students and professors could store and share workflows.

1.4.4 YesWorkflow Upload Workflow Method

Their will be a prototype for uploading data workflow and run data from the YesWorkflow tool. This function will initially be developed using the Java implementation of the YesWorkflow Command Line Tool. Implementing this functionality in the Java version of YesWorkflow will be faster than doing so in the C++ version of YesWorkflow. As a result, we will have more time to work on the backend web API and user interface.

1.5 Project Scope

The the scope of our project includes a website that complements the existing pieces of YesWorkflow, a database that will hold saved workflow data and a command to save YesWorkflow data from the command line. Currently YesWorkflow exists as a java jar file and a C++ implementation. We will be using the portable jar file to test uploading to the website adding the previously discussed save command to it. YesWorkflow provides diagram creation, script parsing, and the data model for workflows that we will use for our website. Our project is split into two primary components. The first component is a Django based website that allows users to interact with the existing YesWorkflow backend. The website will provide a means to access provenance as well as a way to easily interact with YesWorkflow data. The second component is adding a database backend with a schema to organize and store workflow data. We will to create a database schema as well as a new ‘save’ command in order to implement our database.

Upon completion, our project may be integrated with the SKOPE project. This will signal its completion and proof of concept by allowing SKOPE to have a component that maintains workflow history. This is a major goal put forth by our sponsors and a large part of the reason we are building this product.

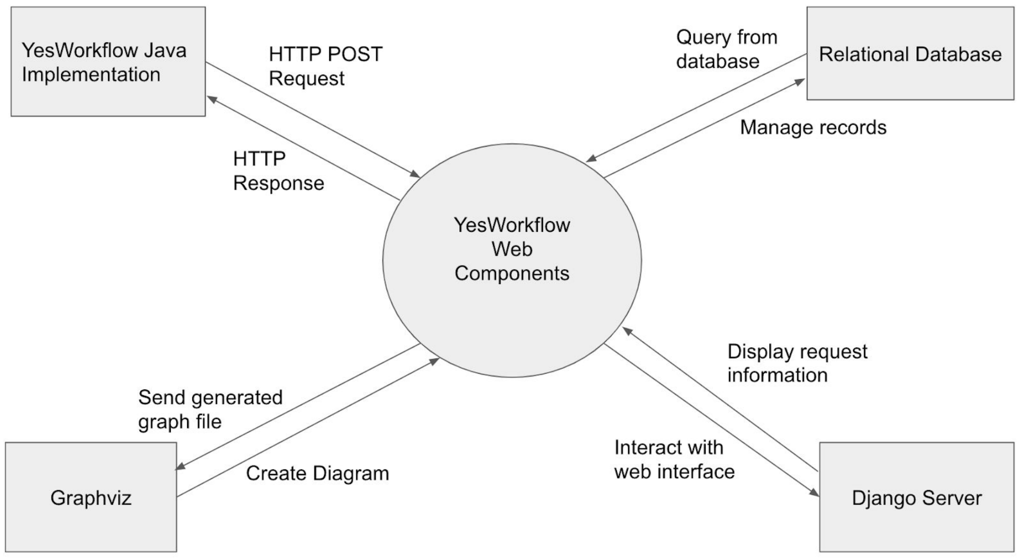

YesWorkflow Java Implementation
HTTP POST Request
HTTP Response
Query from database
Relational Database
Manage records
YesWorkflow Web Components
Display request information
Send generated graph file
Graphviz
Create Diagram
Interact with web interface
Django Server

Figure 1: Context Diagram

Figure 1: *This context diagram represents the scope of our project and what pieces are coming from exterior sources. YesWorkflow's Java implementation will be the source of our data. It generates and uploads the data associated with a single workflow. The relational database will take the uploaded information from YesWorkflow and store it. The Django service will then bridge user input and our database to service the workflows the user is interested in viewing. Graphviz is the service that parses the .gv files generated by YesWorkflow and creates a diagram.*

1.6 Related Work

There exists a number of workflow execution systems in the world today. They cover a variety of topics, but they all have one goal in common: to display a graphical visualization of a workflow. Apache Taverna [1] is a major contributor to the scientific workflow community, being the largest and most widely used platform for executing workflows based on bioinformatics, astronomy, and biodiversity. The system employs Taverna Workbench, a Java based graphical user interface used to design and enact (with the help of Taverna Engine) workflows. Taverna also offers a server client which can be used to execute workflows remotely on a server as well as a command line tool for executing workflows in the terminal. Taverna runs a workflow repository called myExperiment[2] that utilizes source control in a similar manner to GitHub. Scientists can share their workflows as open source files for others to use or contribute to. There is an appeal to working with Taverna because of the large community support behind it. Working in the Taverna suite gives users access to over 3500+ resources including BioCatalogue[3], which is an online catalogue of life science web services.

[1] About Apache Taverna, https://taverna.incubator.apache.org/

[2] About myExperiment, www.myexperiment.org

[3] About BioCatalogue, https://www.biocatalogue.org

Our project is aimed to provide YesWorkflow's capabilities with an easy to use modernized graphical interface with source control tools similar to what Taverna offers. In regards to the command line tool, YesWorkflow is feature complete. There are currently four commands in YesWorkflow: extract, model, graph, and recon. The extract command parses source code for comments in the code. Model finds any valid YesWorkflow annotations and parses these annotations out. Graph takes output from the model command and generates a .gv file which Graphviz uses to create a visualization. Recon also takes output from the model command and generates a list of all file inputs/outputs that were found in the local file system. However, we need to provide an additional feature to the command line tool to interface with the webpage. An additional command, save, will consolidate output from each command and upload the data to the web interface.

While we won't go over all available scientific workflow systems[4], there are a few that are closely related to the goals of this project. Discovery Net[5] is presented as one of the earliest scientific workflow systems. It had a robust feature set including a workflow server, graphical interface with drag and drop capabilities, clear separation between data and control flows, and data management within the workflow engine. Discovery Net has two main ways of representing workflows: they can be stored using DPML, (Discovery Process Markup Language) which is an XML based language for workflow graphs, or a modern drag and drop system that constructs workflows by connecting nodes together. The goal of our project is similar, but not as feature complete as what Discovery Net has to offer. For instance, we will not support a drag and drop editing system because our graphical interface focuses on moving away from the command line rather than having a creation engine. This is important because implementing a system like this would require redesigning the core YesWorkflow system of generating visual representations through annotations.

Kepler[6] is a workflow system that offers workflow tools similar to those provided in Apache Taverna Workbench. There are two main ways to run a workflow: inside of a graphical interface, or in a terminal. The graphical interface allows creation and running of workflows, while the command line interface supports scripts written in Java, Groovy, JavaScript, Jython, Python, and C/C++. To share Kepler workflows, a user must export their workflow into a Kepler Archive file (KAR). KAR files can be shared through traditional means personally or uploaded to the Kepler Component Repository[7]. Because it supports many languages, Kepler is flexible in handling workflow data for any situation. YesWorkflow works similarly in that any language can be used as long as they are annotated with YesWorkflow's tags. While there is a lot to draw on from these other systems, none of them capture the simplicity of YesWorkflow.

[4] List of Workflow systems, https://en.wikipedia.org/wiki/Scientific_workflow_system#Examples

[5] About Discovery Net, https://en.wikipedia.org/wiki/Discovery_Net

[6] About Kepler, https://kepler-project.org/

[7] About Kepler Component Repository, https://kepler-project.org/developers/teams/framework/archive/kepler-repository

2 Project Requirements

2.1 Major features

Our minimally viable product has two major parts. First, there must be a way to 'save' a workflow using the YesWorkflow command line tool. Second, there must be a method to visualize and organize workflows on a server. Saving a workflow will take place through a request to a server via the YesWorkflow command line tool. Visualizing and organizing workflows involves features such as sorting workflows by user, querying for sets of workflows, and displaying YesWorkflow output.

Less pertinent features include executing and editing scripts from the website, condensing the YesWorkflow tool into a single package, and creating a C++ based Python module. Condensing dependencies into a single package would make setting up YesWorkflow easier. However, setting up the backend for data provenance is of higher priority to our sponsors. A Python module would allows for easier YesWorkflow integration, but development on the C++ version of YesWorkflow is still in progress. Executing user submitted scripts would likely be unnecessary, the feature adds excessive overhead to the server for a marginally useful feature.

After identifying possible objectives with our sponsors, our team compared each objective looking for relevancy. The objectives that fit together best or were ready to be implemented now were selected as major features. Features on the edge were pushed to stretch goals. We received feedback from scientists who have used the system (eg, our sponsors), as well as from our own usage of the software to pick the most relevant features.

Table 1: Major Features

Feature	*Description*
Persistent YesWorkflow Storage	Persistent storage takes the form of a remote database to store YesWorkflow output and workflow data transformations. This allows users to save previous workflows.
Save YesWorkflow output from the YesWorkflow Java Implementation	Saving a workflow is an additional YesWorkflow command. This command lets a user send workflows to persistent storage on their device or upload the workflow to a web server.
Distributable Django Web Server	The Django server provides a front end to display and query YesWorkflow workflows. An admin can easily download and set up this server so users can visualize YesWorkflow output and query saved workflows.
Querying Workflows from Web-based UI	A user can look up saved workflows using a range of criteria. Querying provides users a method to find specific workflows from a generic list hundreds or thousands.

Display Workflows from Web-based UI	The display lets users view and interact with YesWorkflow based workflows from the web server. Users will have a convenient way to be able to see data and examine workflows they previously worked with.
Organize Workflows by User in Web-based UI	Each user has a set of associated workflows. By clicking on a user, one can view the files and workflows the user has added to the database.
Edit Metadata About Workflow from Web-based UI	A user can edit his/her own workflows. Functions available will include deleting a workflow, editing a description, adding and removing tags, and renaming the workflow.
Download Workflows and Associated data	A user can use our website as a source control and easily be able to download and retrieve copies of files that were used when creating a workflow.
Microservice-based Architecture	The beginnings of a microservice architecture. There will be a backend web API handling queries to the database. A front end website will make requests to the backend API to populate its pages.
Example Workflows	We will create and upload with the default web server package some example workflows so users can get examples to look at.

2.2 Initial Product Backlog

Table 2: Initial Product Backlog

Requirement	*Description*	*Major Feature*	*Priority*	*Estimate*
YW save	As a run-provider, I can enter a command to save the run data produced by my scripts and export to a server.	*Option to Save a Workflow*	1	100 hours
YW save version	As a run-provider, I can version and export a workflow to the server by adding a flag to the ‘save’ command.	*Option to Save a Workflow*	8	20 hours
YW save data	As a run-provider, I can export data to the current version of a workflow by adding a flag to the ‘save’ command.	*Option to Save a Workflow*	9	20 hours

Persistent Storage	As a run-provider, I can store the output of YesWorkflow commands in a database on a server so I can access them retroactively.	*Persistent Storage*	1	60 hours
Server config settings	As a run provider, I can specify a server in a config file so I can send my workflow to the correct server.	*Distributable Django Web Server*	3	20 hours
Username config settings	As a run provider, I can specify a username in a config file so I can save my workflow under my identity.	*Distributable Django Web Server*	3	20 hours
User login	As a user I can login to the server so that my uploaded files are protected.	*Distributable Django Web Server*	3	30 hours
Viewing other user's workflows	As a run-viewer I can access workflows from a specific user so that I can view them.	*Organize Workflows by User in Web-based UI*		50 hours
Version history list view	As a run-viewer, I can view a chronological workflow version history on the website so that I can see past versions of a specific workflow.	*Display Workflows from Web-based UI*	5	20 hours
Script directory view	As a run-viewer, I can view my script directory on the website so that I can see my collection of scripts	*Display Workflows from Web-based UI*	4	30 hours
Queries results view	As a run viewer I can page through a list of results so that my results are segmented.	*Querying Past Workflows from Web-based UI*	2	10 hours
Query by script name	As a run viewer I can query by script name to narrow results.	*Querying Past Workflows from Web-based UI*	4	10 hours

Query by run name	As a run viewer I can query by name to narrow results	*Querying Past Workflows from Web-based UI*	3	10 hours
Query by date	As a run viewer I can query by date to narrow results.	*Querying Past Workflows from Web-based UI*	3	10 hours
Query by workflow tag	As a run viewer I can query by workflow tag to narrow results.	*Querying Past Workflows from Web-based UI*	3	10 hours
Query by uploader	As a run viewer I can query by uploader to narrow results.	*Querying Past Workflows from Web-based UI*	4	10 hours
Displaying result options	As a run-viewer, I can click on a specified run to see refined details about this run.	*Display Workflows from Web-based UI*	1	10 hours
Displaying workflow	As a run-viewer, I can see the graphical representation of a workflow so that I can view the data transformations.	*Display Workflows from Web-based UI*	1	10 hours
Displaying workflow tags	As a run viewer, I can see the associated tags to the workflow so that I can find related workflows.	*Display Workflows from Web-based UI*	1	5 hours
Displaying workflow description	As a run viewer, I can see a description of a workflow so that I can get a better understand the workflow.	*Display Workflows from Web-based UI*	2	5 hours
Deleting a workflow from the website.	As a user I can remove my workflow from the website so that the workflow is no longer on the server.	*Display Workflows from Web-based UI*	2	10 hours

Viewable example workflows	As a team member I need to create example scripts that new YesWorkflow users can examine to get a better idea of how to make a workflow.	*Example Workflows*	8	25 hours
Editing a workflow description on the website.	As a user I can edit a description of a specific workflow so that I can update information about a workflow.	*Display Workflows from Web-based UI*	6	20 hours
Viewing a server identifier.	As a run-provider I can view a server identifier from the website so I can send my workflows to the correct server.	*Distributable Django Web Server*	6	20 hours

2.3 Additional Features

2.3.1 Nested Workflows
A workflow will often have components that are an abstraction of another workflow. Any node in a YesWorkflow graph could represent an abstraction of another workflow. As a user it would be useful to analyze the individual components of a workflow. This is significantly more complex than a basic flat view of a workflow that treats individual components as a black box.

2.3.2 Tree Structure for Versioning Workflows
The web interface gives users the ability to upload a new version of the same workflow. In the first iteration, this function will be completed in a linear fashion. The tree structure method of versioning is similar to git versioning where a user can create branches that have a different 'history' of workflows up until each branch shares a common version in their ancestry. A branching versioning system would model how researchers and data scientists experiment and manipulate their scripts with some trial and error.

2.3.3 Condensing YesWorkflow Java Command Line Tool into a Single Package
A single package lets users download YesWorkflow and then use the software immediately. This would mean that users can access the web interface as well as command line tools with a single installation. We don't believe that this is a priority since a single package isn't necessary for the system to work. This would be made possibly either through a docker image or statically compiling all dependencies with the C++ implementation of YesWorkflow, and then including these files with our project.

2.3.4 Creating a C++ based Python module
The ability to natively run YesWorkflow in a Python module would allow all of the functions of YesWorkflow to be used from within a Python script. This would integrate script and workflow development into a programming language so that users can manipulate data from within the script. This would allow users to automate their YesWorkflow processes in conjunction with their data analysis automation. Also, many data scientists and users of YesWorkflow tend to use scripting languages such as

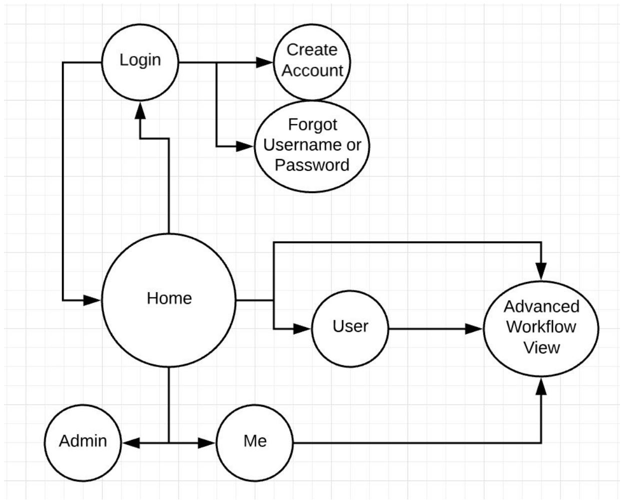


Figure 2: Storyboard
The storyboard shows users navigate between pages and the general flow of logic in the web interface.

Python to conduct their studies. This means that the barrier for entry would be much lower for scientists just beginning to use YesWorkflow. Finishing YesWorkflow's C++ implementation and subsequently creating a Python module would take considerable time for a feature that is independent of our project's goals and uses.

2.3.5 Reimplementing save feature on C++ version
Currently the Java version of YesWorkflow is deprecated and a branch that is no longer under development. The C++ implementation is the current future of YesWorkflow as it solves the problem of having many dependencies. This implementation of YesWorkflow will need our proposed save command in order to be compatible with our project.

3 Design Considerations

3.1 Initial User Interface Design

Our user interface is web based. We will have separate pages that users can cycle through as they interact with it. We arrived at our UI design by researching similar systems and discussing amongst each other how we could best represent YesWorkflows data. As the scope of our project evolved and what we deemed important changed, our UI sketches evolved as well to allow for a flexible, simple and intuitive experience for users. User's navigate with respect to the storyboard in Figure 2. Our website when opened by the user will land them on a home page, Figure 3, containing all uploaded workflows. Uploaded workflows will be displayed with the YesWorkflow generated diagram as well as a brief user submitted description. From the home page the user can navigate to several different pages. Users can query uploaded workflows and click on the workflow they wish to interact with. Clicking on a workflow navigates a user to a detailed workflow view, Figure 4a-4b. From the detailed workflow view, a user can click the 'download' button and download the files parsed and uploaded from the YesWorkflow tool's

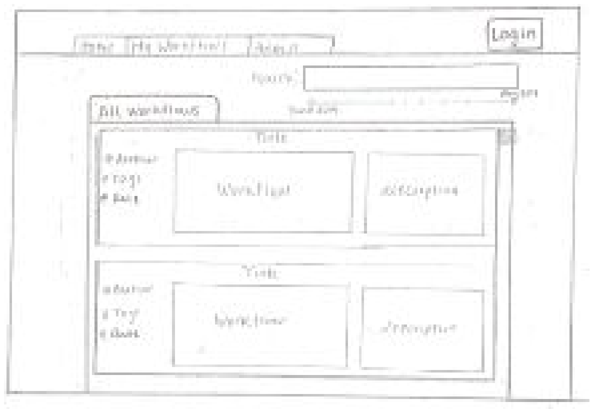


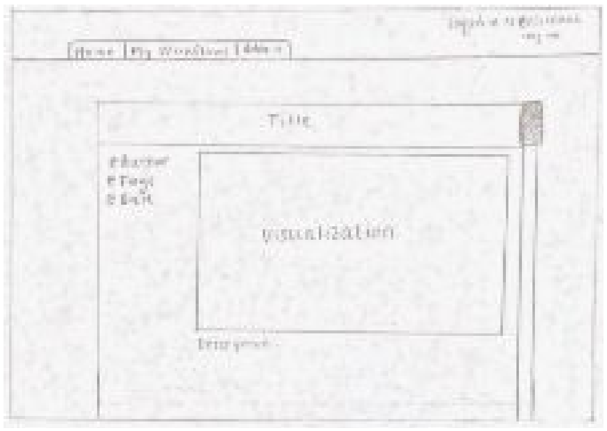
Figure 3: Home page

This image represents the homepage of our website. It contains a listview that contains all uploaded workflows visible to the user. The listview sections are split into a graphic containing the workflows diagram and then a brief description right beside it, as well as a title at the top of the view. There is a simple search bar at the top that queries for the title of a workflow. Below the search bar is a slider that allows a user to refine their query by the publishing date. A user can click on the title of a workflow and navigate to an individual workflow page, shown in Figure 4a and 4b. A user can also click on the author of a workflow to visit a page that lists all the workflows published by that author found in Figure 5.

save command. A user can also click an uploader's username on a workflow and navigate to a page that contains a view of only the uploader's workflows, Figure 5. If a user would like to manage their own workflows from the website, they may click the 'Login' in the nav bar and navigate to a login page, as shown in Figure 6. From here, a user can either sign in, or create an account. Signing in gives a user access to the 'My Workflows' tab. This tab navigates a user to a page displaying all of their workflows, shown in Figure 7. Creating an account navigates the user to another page, Figure 8, where they can create a password and username for their account. If a user logs in and has admin privileges, this user additionally has access to the 'Admin' tab. The admin view in Figure 9 gives a user the capability to manage any workflow or user associated with the server.

3.2 Initial Software Architecture

Our application uses Django as the framework for our web server and all data is stored on a database located on the host of the server. Graphviz is used to turn generated graph files into workflow diagrams from data exported from the YesWorkflow core. SQLite will be used to allow users to query data from the database and receive results, but users can use a different database implementation if they choose to. When the user uses the proposed YesWorkflow save command, they will send a POST request to our server that will organize and store all the data that was generated for their current workflow. An outline of architecture is found in Figure 10.



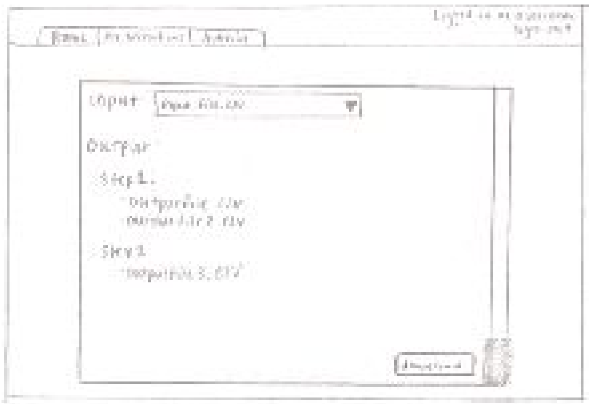


Figure 4a: Individual Workflow Page -- Top **Figure 4b: Individual Workflow Page -- Bottom**

Figure 4a: *This page is displayed when a user clicks on the title of a workflow from one of the available list views in Figures 3, 5, and 7. In this view, a user can see an expanded view of a workflow.*

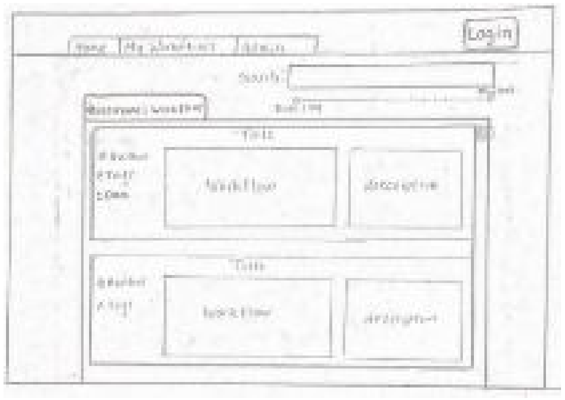
Figure 4b: *This is the individual workflow page scrolled down. Below the visualization and description, a user can select an initial input file from a dropdown list and see how this input file flows through the workflow. The user can see the output of data at each step of the workflow.*

3.3 Initial Software Test Plan

In order to test our project, we will have various stages of testing that combined will provide quality assurance. As we begin developing we will need to implement automated testing to keep our site's quality at acceptable levels. Fortunately, Django comes with a testing framework based of the standard Python unittest framework. This will allow for easy automation and running of our tests, as an example, the terminal input needed to run every test associated with our website would simply be *python3 manage.py test.*

Convenient test access is not enough. Tests must be structured so that when code changes occur we know the true impact of our work. Unit tests provide functional, sanity, integration and regression test automation for our website as we develop. Automated tests speed up the bug hunting process and ensure that the backend of our website functions as expected. It is important to have complete code coverage by our unit tests as the rest of our testing and implementation will depend on these components working.

Once that is complete it will be necessary to create integration tests that automate testing to be sure that our service works from end to end. This testing will need to be done near the end of our project completion, and will likely take less time than writing unit tests as it will only focus on our project from the highest level. Should it be deemed necessary, we will need to evaluate the performance of our chosen database and database schema. As we develop, it will be necessary to have our sponsors test our website and determine if they agree with our vision, as well as test our website in the same environment as a user would utilize it. As this is a website, we will need security checks. Fortunately, we will not be designing security components from scratch; instead, we will utilize the security features included with Django.



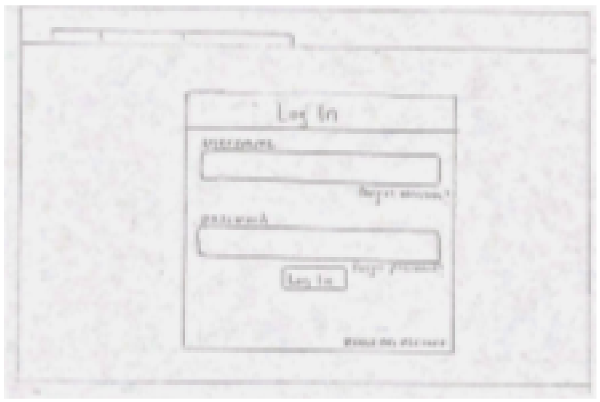


Figure 5: User's Workflow Page **Figure 6: Login Page**

Figure 5: *The User's Workflow page has the same characteristics as the home page--Figure 3--with the exception that all the visible workflows were published by the same user.*

Figure 6: *When a user clicks 'Log In' at the top of a page, they are brought to the Login view. In this view, users can enter a username and password to sign in, select that they forgot their username, select that they forgot their password, or at the bottom they can choose to make a new account shown in Figure 8.*

4 Project Risks

4.1 Poor Estimates

For most of our group, this is the first large scale project we have attempted. Additionally we are working with Django, a tool that we have little experience with. Given our inexperience, our estimates for how long certain tasks take may be inaccurate. If tasks take significantly longer than we expect, we are at risk of not completing our product deliverables. We will follow our sprint plan to mitigate this risk. After each sprint we will have a review to analyze what we completed and to see if our team velocity is at a reasonable pace. If at any point we see that we are a full sprint behind schedule, risk management must occur. Our plan B is to reduce the scope of the project, since we cannot add more developers, and we cannot extend the deadline.

4.2 Missed the Scope of the Project

Our group may stray from the desired scope of our project sponsors. Not delivering the right product means that our project has either failed, or is challenged. We will work toward preventing this risk by checking in with our sponsor, Dr. McPhillips bi-weekly and Dr. Bowers weekly once development begins.

During these meetings we will discuss progress and project direction to get feedback. If Dr. Bowers or Dr. McPhillips feel we are heading in the wrong direction, then we need to go to plan B. Plan B is to review the previous and next sprint with our sponsors and reorganize so that the sprints cater toward the needs of our sponsors.

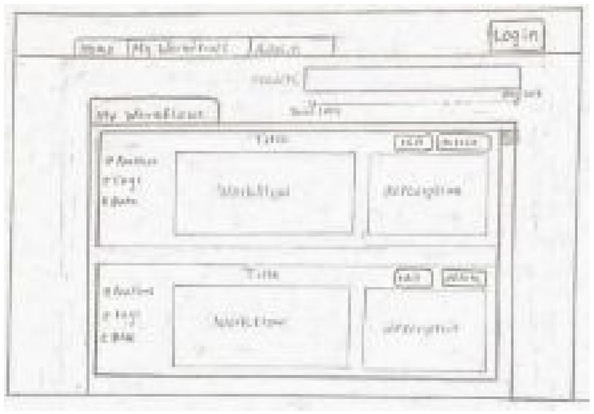


Figure 7: My Workflows Page

Figure 7: *By clicking the 'My Workflows' tab at the top of the page, users can see a list view of all the workflows they have published. This page has the same characteristics of the home page, Figure 3. From this page, users have the additional options of deleting or editing a workflow.*

4.3 Mismanaged time

This is a large project that is expected to take two semesters to complete. We need to keep a close watch on how we are managing our time and progressing. Time is a risk to our project because there are no deadline extensions. Mismanaging our time will lead to a scenario where project deliverables are not complete by the deadline. To mitigate this risk we have elected to follow the Scrum development methodology to manage project tasks. At the end of every sprint we will examine our progress and evaluate whether we are on pace or falling behind. If by the end of the first semester we are more than two weeks behind schedule, we will have to undergo a major restructuring to our project plan to get ourselves back on track. Strategies for restructuring the project plan include reducing scope, increasing our time investments or shuffling our responsibilities to improve team velocity.

5 Initial Product Release Plan

5.1 Major Milestones

We have created a list of milestones for our project so we can organize our plan around smaller steps. This will allow for a better way of tracking our progress and making sure that we are on pace with our goals. The list is ordered in this particular way because of dependencies and complexity. To get our project going faster we elected to put the easiest components as our earliest milestones. Than to ensure that our project progresses as intended we put tasks that are used by other pieces as earlier milestones.

Table 3: Major Milestones

Milestone	Description	Target Completion Date
Complete UI Design	Complete our UI design add it to our Project Plan	Third week of October
Project plan confirmed	Ensure our project requirements are in line with Tim's needs	Third week of October
Web server database implementation	Having a working backend to connect to our front end will make data management straightforward	First week of December
Integrate request from YesWorkflow to server's REST API	Ability to send a workflow over the wire	Fourth week of February
Front End Design Implemented	Have the UI sketches implemented in the Django website	Second week of May
Front End Functionality Implemented	Have all attributes of the website functional	3rd Week of April

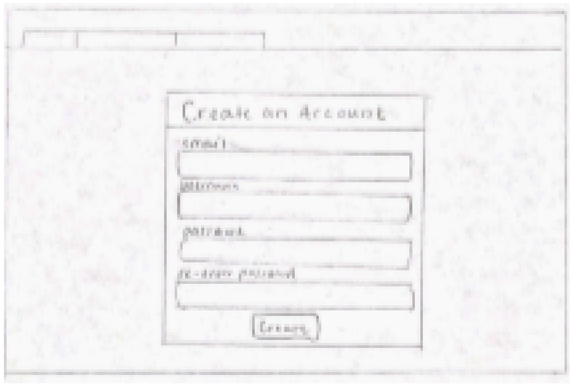
5.2 Initial Sprint Releases

Table 4: Sprint Release Plan

Sprint Date	**Sprint Goal**	**Backlog**	**What will we demo?**
1st Week of September to 2nd Week of September	Gather Project Requirements	As a team member, it is necessary to gather the requirements of our project from it's stakeholders so that we know what stakeholders want from our project	A compiled list of project requirements
2nd Week of September to 4th Week of September	Create Project Plan rough draft	As a team we need to work on creating a Project Plan proposal so that we can properly convene our plans for	A rough draft of our Project Plan that includes a UI design

		this project	
1st Week of October to 3nd Week of October	Revise Project Plan	As a team we need to revise our Project Plan so we that it is refined and presentable	A final copy of our Project Plan with a completed UI design
4rd Week in Oct to 1st Week in Nov	Begin developing a database schema	As a run-provider, I can store the output of YesWorkflow commands in a database so I can access them retroactively	A basic schema that captures YesWorkflow data and workflow specific metadata (Eg. date)
2nd Week in Nov to 3rd Week in Nov	Begin work on YesWorkflow 'save' command	As a run-provider, I can enter a command to send the run data produced by my scripts to the YesWorkflow website so that my run data is stored	The use of the save command in the command line with a successful POST request
4th Week in Nov to 1st Week in Dec	Develop database schema for Web Server	As a run-provider, I can store the output of YesWorkflow on the web server database so I can access them retroactively	Implemented database that is integrated with server
3rd Week in Jan to the 4th Week in Jan	Integrate YesWorkflow labeling and tagging on website	As a run-provider, I can label a run for organization in the database and website UI so that I can query for that label	The ability to label, tag workflows, and search for them
1st Week in Feb to the 2nd Week in Feb	Integrate REST API to server	As a run-provider I can send a REST request to a specified web server so that I can communicate with the web server	The integration of the REST API

3rd Week in Feb to the 4th Week in Feb	Develop method to send POST request to web server REST API	As a run provider, I can specify a server in a config file so I can send my workflow to the correct server	The method to send POST request to web server
1st Week in March in to 2nd Week in March	Implement the front end UI designs	As a run-viewer I can navigate and use the YesWorkflow website in an intuitive way so that my productivity is increased	The a frontend for the Django website
3rd Week in March to the 4th Week in March	Saved workflows are visible from the web front end	As a run-viewer, I can see all published workflows on the website so I can view the work of others	The workflow being saved and the created workflow on the website
1st Week in April to the 2nd Week in April	Allow users to query run data	As a run-viewer, I can run queries over metadata to extract specific runs	Query functionality within the website
3rd Week in April to the 4th Week in April	Implement a version history for workflows	As a run-viewer, I can view workflow version history on the website so that I can see past versions of a specific workflow	The different versions of the workflows are saved
1st Week in May to the 2nd Week in May	All aspects of the website are fully functional	As a run-viewer, I can see all published workflows on the website so I can view the work of others	The entire website's functionality



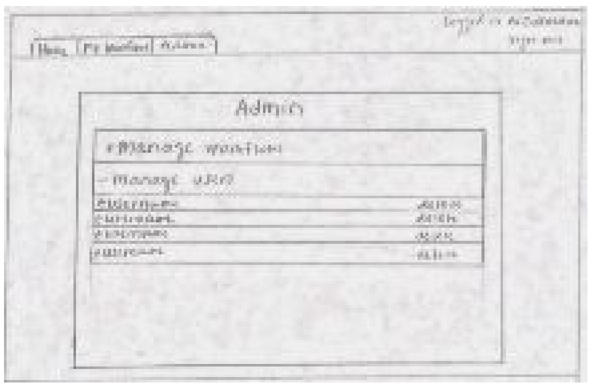


Figure 8: Create Account Page **Figure 9: Admin Page**

Figure 8: *On this page, a user can create an account with an email, username, and password.*
Figure 9: *An admin can click the tab that appears at the top of the page. This tab only appears when a user is logged in as an admin. An admin can delete any workflows from the site. These workflows are listed by title, author, and date. An admin can also delete a user from the website. Users are listed by their username.*

6 Maintenance Considerations

We will take into consideration that maintenance will be provided by the YesWorkflow team following our completion of the project. In preparation for this, we will write extensive documentation for features implemented. Maintenance will consist of server management (if an end-user hosts the product on a web server) and integration of our project onto YesWorkflow's Github profile. Adding these features will further extend YesWorkflow's viability as a product and a complete package. Experience needed for maintenance includes web development experience and a fundamental understanding of the YesWorkflow tool.

In order to make maintenance easier, we elected to make two major decisions in our design. First, the features provided by our website will be logically separated from the core YesWorkflow code base through an abstraction layer. This will allow for future users and developers to very easily modify our code without having to touch the core YesWorkflow code. In addition, we elected to use Django because it is a Python based framework. This means that data scientists that are familiar with Python will have an easier time modifying and maintaining their own flavors of our server.

7 Project Management Considerations

As a team, we will meet each week on Monday in the Herak building and then meet again on Wednesday in Jepson. This will allow us to get some work done together as well as distribute tasks. Any additional work will be completed individually on our own time.We will meet regularly with our sponsors to make sure our vision is aligned with theirs.. However, our meetings will be over the phone and on Skype due to distance. Our sponsor will receive updates both through email and through phone call meetings. We will update our advisor in weekly meetings and through office hour visits. Other stakeholders, like Skope, will

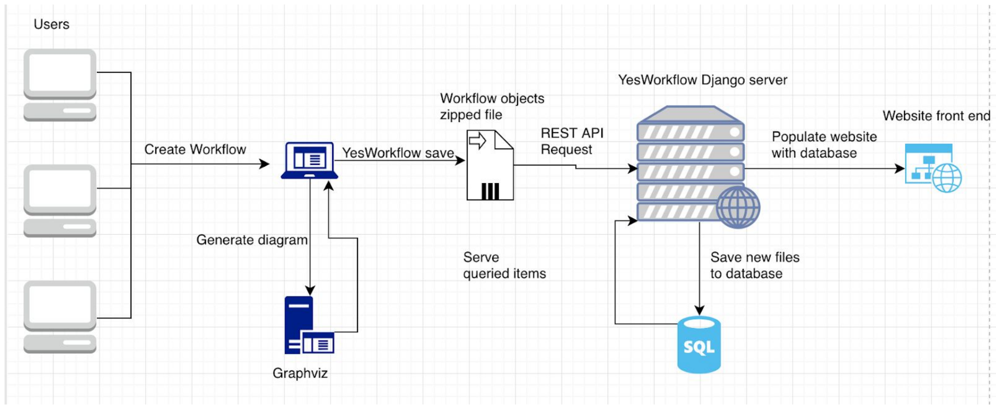


Figure 10: Architecture Diagram

Figure 10: *Shows the general architecture of our project form run-provider to the run viewer.*
be updated through our sponsor unless they request otherwise. Our DAB member will be updated through our advisor but additionally we will email him on occasion if we are looking for help.

The work on our project will be split up with Brooke and Mackenzie taking front end web development and Brewer, Anthony, and Carl taking back end development. While Brooke and Mackenzie will work together, Brooke will focus more on implementing design and Mackenzie on front end functionality. Brewer will focus on setting up our database, Carl will be working on implementing a REST API, and Tony will focus on adding the 'save' tool to the YesWorkflow toolkit.

We will use email to ask our sponsor concrete questions so we can have those responses on hand. In addition, we will use Skype and some kind of screen sharing software to allow our sponsor and other distanced stakeholders to use our product and deliver feedback. Finally, we will have people outside of the team test the product to elicit feedback on the design and functionality.