VividFlow

Technical Report

Group 9

UNIVERSITY OF WOLLONGONG AUSTRALIA

Autumn – Spring 2016

Project Details

Subject	CSIT321 – Project
Session	Autumn – Spring 2016
Subject Coordinator	Mark Freeman
Campus	Wollongong

Group Members

Student #	Name	Email
4773810	Joshua Coleman	jjc224@uowmail.edu.au
4752405	Philip Edwards	pme446@uowmail.edu.au
4669174	Thomas Nixon	tn941@uowmail.edu.au

Client

Dr Igor Kharitonenko

Faculty of Engineering and Information Sciences

School of Computing & Information Technology

Table of Contents

Executive Summary

VividFlow is a web application designed for prototyping algorithms focused on computer vision. By allowing researchers to make discrete modules of functionality and connect them together larger algorithms and systems can be quickly prototyped and iterated.

VividFlow successfully met all the requirements that were ascertained for the system, and some of the optional stretch goals were also completed. A modified scrum development methodology was used for the development and worked well with the team and the allotted time.

The system is client/server based and as such is developed primarily in Python and JavaScript. Many of the classes are implemented in both languages, and communication is handled between the front end and back end through JSON objects.

Testing of the system was accomplished through manual test scripts that the developers followed to repeatedly test the system after work had been finished after each sprint. This system was less intensive timewise than unit tests, however had the project gone longer, or if there were more developers than a new testing policy would be needed.

The installation of the system is straight forward, requiring just a single script to be executed by the administrator. VividFlow is also highly configurable by editing the JSON configuration file.

Overall the project was successful. The development team didn't run into any significant trouble, however there were many lessons learned that can be improved on in future.

Introduction

VividFlow is a web application designed for usage on desktop systems. It is designed to make the process of prototyping algorithms quicker and easier for computer vision researchers. By allowing researchers to make discrete modules of functionality and connect them together larger algorithms and systems can be quickly prototyped and iterated.

The system allows multiple researchers to concurrently access the system and work on their algorithms. Researchers can upload C++ programs they have written with the assistance of a helper library that is provided with the system. A researcher can define what inputs and outputs a module will produce. Once they have defined a module, it can be placed in the algorithm graph using the algorithm designer. The system can then run the algorithm and will allow the researcher to download the outputs that were produced by the algorithm.

This document will cover some of the technical aspects of the system. Some basic knowledge of web technologies and software development is assumed for this report, however some particular terminology specific to this project or some of the core technologies is outlined.

The system is designed to meet 26 functional requirements. The functional requirements covered in detail below. All 26 functional requirements have successfully been met. There are a number of non-functional requirements that have also successfully been met as well.

The 26 functional requirements can be described through 22 use cases. The system follows these use cases and testing scripts were written so the system could be tested in a repeatable manner to match the desired usage. These requirements and use cases are discussed in detail to show that they have been met or in some instances merged with others to reduce system complexity yet still meet user requirements.

The project was managed through a modified scrum method as traditional scrum methodology would not fit well with the size of the team or the available hours each week since it was not a full-time endeavour. The functional requirements were still used for focusing the efforts of each sprint.

The target platforms for VividFlow need to be addressed from both a server and client side perspective. Both aspects are detailed, including library dependencies and hardware requirements. Since the system is aimed at computer vision research, the base hardware requirements are relatively low, however the system will perform better with additional CPU and RAM resources allocated.

The design of the UI has been carefully considered to make it easy for researchers to understand and iterate quickly without getting frustrated by a UI in the way. The design decisions behind the UI are discussed and an outline of the UI workflow is presented.

The architecture of the software is presented in detail to help provide any future maintainer with a clear understanding of the Server/Client split and some of the more important design decisions. The

class hierarchy is also presented, with highlighting to show whether classes reside on the server or client side and to understand how all the modules in the system fit together.

The installation of the system is designed to be as automated as possible. Once Debian 8.6 is installed, the system can be installed just by inserting the CD and running a single script. Clear instructions on setting the system up are provided. Once installed configuration can be easily updated by modifying a text file.

Finally, the project is discussed in a reflective manor. Some of the more major issues that arose during the development of the system are discussed along with how they were resolved. Many lessons were learned during the course of development and some of these are highlighted so that they can be avoided in future. The timeline of the project is also presented to show how the system progressed.

Technical Glossary

Some basic knowledge web technologies is assumed for this report, however some particular terminology specific to this project or some of the core technologies is outlined below.

Term	*Definition*	*Reference*
Algorithm	In VividFlow an algorithm is a graph composed of nodes, modules and links that will process input and return output data.	
Algorithm Designer	The algorithm designer is the area of the program where a researcher can place modules, nodes and links and create a complete algorithm.	
Algorithm List	The page in VividFlow where a user can view all algorithms they have access to.	
CDN	Content delivery network. A service which can host common or shared resources on the internet	
Data Type	Data Type refers to the supported data types in the system: String, Integer, Float, Image, Video, Text	
Flask	Flask is a python web application framework that is used by VividFlow	http://flask.pocoo.org/
GUnicorn	GUnicorn is a WSGI HTTP Server that is used by VividFlow	http://gunicorn.org/
Job Scheduler	A daemon that runs in the background on the server that monitors for new jobs to run.	
Link	A link is a connection between 2 sockets that will determine how data flows through an algorithm	
Module	A module is an object in the system that contains a C++ program and can have input and output sockets defined.	
Module Code	Module Code refers to the C++ program that is contained in a module	
Module Designer	The module designer is the area of the program where a researcher can update code and specify sockets.	
Module List	The page in VividFlow where a user can view all modules they have access to.	
Module Node	A module node refers to a module that has been placed in an algorithm	
nginx	nginx is a HTTP server that is used by VividFlow	https://nginx.org/en/
Node	A node is an element of an algorithm. It can be one of several types, and can have many input or output sockets which determine what data goes into and out of the node	
OpenCV	OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.	http://opencv.org/about.html
Output Node	An output node is a way of specifying resulting output of an algorithm	

Resource Manager	The Resource Manager allows a researcher to upload and modify resources
Resource Node	A resource node is a way of input an item from the resource library into an algorithm
Route	A route is a defined URL pattern that will execute a specific python function in the web application, potentially with parameters.
Socket	A socket is a child of a node, allowing data to be input or output of a node through a link
Value Node	A value node is a way of specifying a literal value in an algorithm. It can be of many datatypes: String, Integer, Float, Text

System Requirements

Requirement Summary

Our client initially requested a flexible and convenient framework that would facilitate rapid algorithm prototyping for computer vision researchers. It should minimise the load required to develop algorithms by allowing the reuse of modules and resources. The modules that were used in the algorithms had to be able to use OpenCV. The interface needed to be simple and easy to use.

The client requirements were broken down into 26 functional requirements and then designed and built a system that would meet them. VividFlow is the resulting system. The development process used SCRUM for an agile project management, this let us prototype features quickly.

Requirements Traceability Matrix

The matrix below shows the requirements of the system as asked by the client. We implemented all the original requirements and some of the additional ones. The additional items were only requested after the demonstration of the prototype to the client. We were unable to implement any stretch requirements.

Types of requirements

- Initial (core functional requirement requested from the beginning)
- Stretch (asked for but not required, nice to have)
- Additional (asked for after the prototype demonstration)

Item	*Priority*	*Type*	*Description*	*Comments*	*Delivered*
1 Create Module	High	Initial	A new module will be created. It will make a new entry in the database, create folders, create files.		Yes
2 Update Module	Medium	Initial	Make a modification to a module. Change name, input/output (names, types, default values), upload new code.		Yes
3 Define Module Input & Output	High	Initial	Set the input/output name, type, default value. Type is selected from predefined list stored in database.		Yes
4 Upload Module C Code	High	Initial	Allow user to upload C code to server. It will be saved in the module folder.		Yes
5 Create New Algorithm	High	Initial	A user can create a new algorithm. User defines a name. This will set up the database entries, folders and files. User will be able to start modifying it from here.		Yes
6 Open Algorithm	High	Initial	A user can open an existing algorithm. This will be from a list of existing algorithms created by the current user.		Yes
7 View Algorithm	High	Initial	The algorithm will be displayed to the user.		Yes
8 Add Module to Algorithm	High	Initial	A module can be inserted into the algorithm. Inputs and outputs can then be added.		Yes
9 Add File to Algorithm	High	Initial	A file can be added to the algorithm. This will let the user insert it as input into a module or have a module output data into it. File type can be defined.		Yes
10 Add Value to Algorithm	High	Initial	A value can be added to an algorithm that can be feed in as input or have output feed to it.		Yes
11 Remove Module from Algorithm	Medium	Initial	A module can be removed from the algorithm. This will delete the links between any input and output as well.		Yes
12 Remove File from Algorithm	Medium	Initial	A file can be removed from the algorithm. This will delete the links between any input and output as well.		Yes
13 Remove Value from Algorithm	Medium	Initial	A value can be removed from the algorithm. This will delete the links between any input and output as well.		Yes
14 Create an Algorithm Link	High	Initial	The user can create a link between a module, file or value. The link will be from output to input. This will be restricted by data type of the input and output.		Yes
15 Run Algorithm	High	Initial	The user can run the algorithm. This will first save the algorithm. Then start the makefile to run the algorithm.		Yes
16 View Algorithm Output	High	Initial	The algorithm output will be displayed in the algorithm process.		Yes
17 View Output Values After Algorithm Execution	Medium	Initial	The algorithm items (modules, files, values) can be clicked on. If the file type is supported (checked in database) then the file will be shown and/or allow downloaded.		Yes
18 Save Algorithm	High	Initial	The algorithm can be saved by the user. This will update the database and any files.		Yes
19 Add User	Medium	Initial	A user will be added to the database and a resource folder will be created. A username and password will need to be specified.		Yes

20 Delete User	Low	Initial	A user can be deleted and the database will be updated to reflect this.		Yes
21 Login	Medium	Initial	A user can login into the site. Credentials will be checked in database.		Yes
22 Modify User	Low	Initial	A user can have their username modified. This will update the database.		Yes
23 Reset Password	Low	Initial	A user can have their password reset. This will update the database.		Yes
24 Upload Resource	High	Initial	A user can upload a resource file. This will be stored in the user resource folder on the server and the database updated.		Yes
25 Modify Resource	Medium	Initial	A resource can be modified. The file and database will be updated.		Yes
26 Delete Resource	Medium	Initial	A resource can be deleted. The file will be removed as well as the database entry.		Yes
Different icon types in node	Low	Additional	The value, output and resource nodes should have an image on them to show they are different.		Yes
Program skeleton generation	Low	Additional	A new module has a default template and socket code can be generated.		Yes
Add link to download header in module designer	Low	Additional	Provide a link for the user to download the header source code from the module designer.		Yes
Group modules in trees in Module Selector	Low	Additional	Have the module selector in the algorithm designer break the modules up into a tree structure.	Decided to not implement due to complexity and time constraint	No
Extra module context menu item to edit module	Low	Additional	In the algorithm designer have the modules in an algorithm have a link to the module designer page to quickly edit module code.	Decided to not implement due to complexity and time constraint	No
Encrypted C++ module code	Low	Stretch	Have module code encrypted in some capacity	No time remaining to implement	No

Non-functional Requirements

The system had several requirements that were not functional requirements.

Non-functional Requirement	*Comments*
Must use C++ for modules	VividFlow is by default configured to use the C++ compiler from the GNU Compiler Collection
Support OpenCV library	VividFlow is by default configured to link programs with version 3.1.0 of OpenCV
Keep versions of modules	Module versions are stored on the server, and algorithms are automatically upgraded on load if they reference a module that is in development. Algorithms using published modules will not be upgraded.
Nice user interface that is easy to use, intuitive	The user interface is designed to be responsive, pleasing to the eye and uses the latest web technologies and libraries to accomplish this.

Use make files to represent algorithm	Makefiles are the backbone of algorithm execution. By traversing the algorithm graph a makefile is generated which automatically handles correct algorithm operation.
Define supported formats for displaying (images and video), or download	VividFlow by default is configured to link with several open source media libraries to support a wide variety of image and video formats.
The system will have different users.	VividFlow handles multiple concurrent users, and appropriately limits algorithm execution to ensure the system will remain stable under heavy usage
Secure module C++ source code	Users can only access their own modules, preventing others getting access to secure code.

Iteration Management

Scrum like sprints were used when developing the system. They started in the 2016 Autumn session during the mid-session break and each ran for three weeks. In total, there were 7 sprints to complete the system. The group had some restraints that forced a modified version of Scrum, these were

- Meetings with client were infrequent
- Group members had time constraints with work
- Timeframes were extended because of fewer hours to work on project
- Documentation and assessments took away from development time and interrupted the sprint items
- Small group number created higher overhead for group management as well as many hats worn

The Modified Sprint

Planning meeting on first day of sprint. Sprint goals and sprint items were identified, hours were allocated and items assigned to members. Each week the group would meet and discuss any issues and work on their sprint items. Remaining hours would be updated each week. When the sprint finished, a burndown chart would be generated and a summary of the sprint would be sent to the client. A new sprint would start the next day. Group members would wipe and rebuild their development virtual machines at the start of each sprint.

This meant that stand up meetings were done each week instead of each day. We also had less client buy in on the sprint. Assessment items were left out of sprints as they did not line up with sprint timelines.

Use Cases

Name:	Create Module	ID: 01
Stakeholders and goals:	Researcher – Add a new module to the system	
Description:		
Allows a user to create a module to be later defined		
Actors:	Researcher	
Trigger:	User presses 'Create module' button	
Normal Flow:		
1. User presses "Create Module" button 2. System presents the Add Module interface 3. User enters name of new module 4. User defines inputs and outputs 5. User enters C code 6. User presses save 7. System validates module settings 8. System saves module		
Sub-flows:		
Alternative/Exceptional flows:		
4A. User presses Add Socket button 4B. System presents Add Socket interface 4C. User selects socket type (input or output)		

Name:	Update Module	ID: 02
Stakeholders and goals:	User – update a created module	
Description:		
Allows a user to modify an existing module: • Name of the module • Update its code • Input/output: names, types, default values, etc.		
Actors:	User	
Trigger:	User presses 'Update module' button	
Normal Flow:		
1. User selects an existing module 2. User presses "Create Module" button 3. System presents the Add Module interface 4. User enters name of new module 5. User defines inputs and outputs 6. User enters C code 7. User presses save 8. System validates module settings 9. System saves module		
Sub-flows:		
5A. User presses Add Socket button 5B. System presents Add Socket interface 5C. User selects socket type (input or output)		

5D. User enters name of socket 5E. User selects socket data type 5F. User clicks add 5G. System presents previous module interface 5H. User moves the new socket to the correct order in the socket list
Alternative/Exceptional flows:
8A. System detects invalid settings 8B. System highlights invalid fields

Name: Edit Module C Code	**ID: 03**
Stakeholders and goals:	Researcher – upload code that defines a module
Description:	
Allows a researcher to edit C code respective to a module to the server.	
Actors:	Researcher
Trigger:	User presses 'Edit C Code' button
Normal Flow:	
1. System displays a page with a syntax highlighted field 2. User enters and modifies c code 3. System stores this within the module folder within the server	
Sub-flows:	
Alternative/Exceptional flows:	

Name: Compile Module C Code	**ID: 04**
Stakeholders and goals:	Researcher – upload code that defines a module
Description:	
Allows a researcher to edit C code respective to a module to the server.	
Actors:	Researcher
Trigger:	User presses 'Compile' button
Normal Flow:	
1. User presses Compile 2. System submits code and compiles 3. System returns compile result to user	
Sub-flows:	
Alternative/Exceptional flows:	

Name: Create New Algorithm	**ID: 05**
Stakeholders and goals:	User – create new algorithm
Description:	
Allows a user to create a new algorithm	
Actors:	Researcher
Trigger:	User logs in
Normal Flow:	

1. User presses “Add”
2. System displays Algorithm Designer page

Sub-flows:

Alternative/Exceptional flows:

Name: Open Algorithm	**ID: 06**
Stakeholders and goals:	Researcher – inspect an existing algorithm
Description:	
Allows a user to load an existing algorithm that the user has previously developed.	
Actors:	Researcher
Trigger:	Algorithm is selected from algorithm list
Normal Flow:	
1. System presents a list of available algorithms 2. User selects an algorithm 3. System fetches algorithm code and relating data, then invokes a view event for rendering to display	
Sub-flows:	
Alternative/Exceptional flows:	

Name: View Algorithm	**ID: 07**
Stakeholders and goals:	Researcher – render opened algorithm to the display
Description:	
Algorithm is rendered so that the user can see algorithm and begin making changes	
Actors:	Researcher
Trigger:	User selects an existing or creates a new algorithm
Normal Flow:	
1. System renders an opened algorithm to the display	
Sub-flows:	
Alternative/Exceptional flows:	

Name: Add Module to Algorithm	**ID: 08**
Stakeholders and goals:	Researcher – link a module to an algorithm
Description:	
Allows a user to add a module to the current algorithm as a node	
Actors:	Researcher
Trigger:	User presses ‘Add module’ button
Normal Flow:	
1. User is presented with a list of existing modules 2. User drags module onto graph 3. System inserts the module into the algorithm	
Sub-flows:	
Alternative/Exceptional flows:	

Name: Add Resource to Algorithm	ID: 09
Stakeholders and goals:	Researcher – add file to algorithm for input
Description:	
Allows a user to add a resource to the current algorithm, so that a module can accept it as input	
Actors:	Researcher
Trigger:	User presses 'Add Resource' button
Normal Flow:	
1. User clicks "Add Resource" button. 2. System displays the resource selection screen 3. User selects or uploads a resource 4. System links the specified file to the algorithm for later processing	
Sub-flows:	
Alternative/Exceptional flows:	

Name: Add Value to Algorithm	ID: 10
Stakeholders and goals:	Researcher – add value to algorithm for input
Description:	
Allows a user to add a value to an algorithm, so that a module can feed it as input	
Actors:	Researcher
Trigger:	User presses 'Add value' button
Normal Flow:	
1. User is presented with a view that contains a value and type 2. User submits the form 3. System links the referenced value to the algorithm for input	
Sub-flows:	
Alternative/Exceptional flows:	

Name: Add Output to Algorithm	ID: 11
Stakeholders and goals:	Researcher – add value to algorithm for output
Description:	
Allows a user to add an output node to the algorithm to denote an output filename and the data that should be stored in it	
Actors:	Researcher
Trigger:	User presses 'Add Output' button
Normal Flow:	
1. User presses "Add Output" Button 2. System prompts for filename and data type 3. User enters filename and datatype 4. System adds node to graph	
Sub-flows:	
Alternative/Exceptional flows:	

Name: Remove Node from Algorithm	ID: 12
Stakeholders and goals:	Researcher – remove a specified module from an algorithm
Description:	
Allows a user to remove a node from an algorithm, along with the links to its input and output.	
Actors:	Researcher
Trigger:	User selects a node and presses “delete” key
Normal Flow:	
1. User is prompted for confirmation 2. System unlinks the module and its respective input and output from the algorithm by updating the algorithm’s code on the server	
Sub-flows:	
Alternative/Exceptional flows:	

Name: Create an Algorithm Link	ID: 13
Stakeholders and goals:	Researcher – Link input/output between nodes
Description:	
Allows a user to add a link between two nodes in the current algorithm	
Actors:	Researcher
Trigger:	User presses ‘Create link’ button
Normal Flow:	
1. User can select the two nodes and the socket it will link to 2. The input and output will be limited by the type of the sockets 3. Link will be added to algorithm 4. Database will be updated	
Sub-flows:	
Alternative/Exceptional flows:	

Name: Run Algorithm	ID: 14
Stakeholders and goals:	Researcher – build and execute an algorithm
Description:	
Allows a user to save, compile, and execute an algorithm within a single action.	
Actors:	Researcher
Trigger:	User presses ‘Run’ button
Normal Flow:	
1. System saves the algorithm’s code and relating data 2. System Builds a makefile based off the nodes 3. System executes makefile	
Sub-flows:	
Alternative/Exceptional flows:	

Name: View Algorithm Output	ID: 15
Stakeholders and goals:	Researcher – output what is happening at runtime of an algorithm
Description:	
Allows a user to view the output an executed algorithm as it processes.	

Actors:	Researcher
Trigger:	User presses 'View output' button
Normal Flow:	
1. System displays output node files	
Sub-flows:	
Alternative/Exceptional flows:	

Name:	**Save Algorithm**	**ID: 16**
Stakeholders and goals:	Researcher – save an algorithm for use	
Description:		
Allows a user to save an algorithm		
Actors:	Researcher	
Triggers:	1. User presses the save button	
Normal Flow:		
1. System updates records in the database, along with respective files.		
Sub-flows:		
Alternative/Exceptional flows:		

Name:	**Add User**	**ID: 17**
Stakeholders and goals:	Researcher – create a new user	
Description:		
Allows an admin to create a new user		
Actors:	Researcher	
Triggers:	User presses "Sign Up" button	
Normal Flow:		
1. User opens log in page and clicks "Sign Up" 2. System presents new user form 3. User enters in information 4. System creates the user in the database		
Sub-flows:		
Alternative/Exceptional flows:		

Name:	**Login**	**ID: 18**
Stakeholders and goals:	Researcher – Access site	
Description:		
A user can login into the site. Credentials will be checked in database.		
Actors:	Researcher	
Triggers:	User navigates to site	
Normal Flow:		
1. User is presented with a username and password form 2. User enters username and password and clicks login 3. Credentials are checked with database		
Sub-flows: 1. If unsuccessful the user is notified and can try again		

2. If successful user is taken to the program
Alternative/Exceptional flows:

Name:	**Reset Password**	**ID: 19**
Stakeholders and goals:	Researcher – Change user information	
Description:		
A user can have their password reset. This will update the database.		
Actors:	Researcher	
Triggers:	User clicks ‘Reset Password’ button	
Normal Flow:		
1. User clicks ‘Reset Password` button 2. User is presented form to enter email address 3. System emails new temporary password to user 4. Database is updated with new information		
Sub-flows:		
Alternative/Exceptional flows:		

Name:	**Upload Resource**	**ID: 20**
Stakeholders and goals:	Researcher – Upload resource	
Description:		
A user can upload a resource file. This will be stored in the user resource folder on the server and the database updated.		
Actors:	Researcher	
Triggers:	User clicks ‘Add Resource’ button	
Normal Flow:		
1. User selects file from local system 2. User clicks ‘Upload’ button 3. File is uploaded to server and data stored in database		
Sub-flows:		
Alternative/Exceptional flows:		

Name:	**Modify Resource**	**ID: 21**
Stakeholders and goals:	Researcher – Upload resource	
Description:		
A resource can be modified. The file and database will be updated.		
Actors:	Researcher	
Triggers:	User clicks ‘Update Resource’ button	
Normal Flow:		
1. User selects resource from selection 2. User clicks ‘Modify’ button 3. User can select new file and upload 4. User can change file name 5. User clicks ‘Update’ button 6. File is uploaded to server and data stored in database		

Sub-flows:	
Alternative/Exceptional flows:	

Name:	**Delete Resource**	**ID: 22**
Stakeholders and goals:	Researcher – Upload resource	
Description:		
A resource can be deleted. The file will be removed as well as the database entry.		
Actors:	Researcher	
Triggers:	User clicks ‘Update Resource’ button	
Normal Flow:		
1. User selects resource from selection 2. User clicks ‘Delete’ button 3. File is deleted from server and data stored in database		
Sub-flows:		
Alternative/Exceptional flows:		

Target Platforms

VividFlow is a web application and as a result there are 2 target platforms. The server platform is the one that has been most closely researched, configured and setup for VividFlow. The client side consists of the browsers we tested with, and we have chosen cross platform browsers for the testing.

Server

- Debian 8.6 AMD64
- OpenCV 3.1.0
- Python 2.7
- MySQL 5.5
- Flask 0.11.1
- Nginx 1.6.2
- Gunicorn 19.0

Clients

The client side website uses newer standard web technologies and libraries and should be compatible with most new browsers, however we have specifically targeted and tested the following browsers:

- Mozilla Firefox 47
- Google Chrome 53.0.2785.143

Technology Overview

Server

Language/Runtime

For the main server-side programming language Python 2.7 was chosen over Python 3 since there is more library support (Habib 2016). It is also the default version for Debian 8 (Debian 2016), which factors into the decision since the deployment environment is a Debian 8 Server. It also has official bindings for OpenCV (OpenCV 2016), which influenced our choice over other languages.

Database

MySQL 5.5 is a database management system (MySQL 2016), and is familiar to all the developers on the team. It will more than meet the requirements of this project and by choosing the familiar option over an unfamiliar database will reduce the technical risk associated with this project.

Compiler

As modules that are developed by researchers in the language C++, we have utilised the compiler G++ for module compilation due to it being well supported on Debian (Debian 2016).

Libraries

OpenCV

OpenCV is an open source library that has many of the computer vision and image processing algorithms desired already implemented (OpenCV 2016). Implementation of OpenCV 3.1.0 is a core requirement specified by our client, and hence provides much of the functionality for modules developed by researchers.

Flask

Flask is a python framework for implementing WSGI applications (Flask 2016). It is WSGI compliant, and therefore allows usage of many different webservers to serve the application. Furthermore, it handles processing of HTTP requests, as well as routing of the requests to the appropriate functionality. It was chosen due to the aforementioned reasons, as it enables flexibility and an improvement in productivity by reducing workload by incorporating functionality that handles otherwise tedious tasks; furthermore, as Python is an implemented decision for the core of the server-side, it enforces the design of the server-side, and using a framework based on the language allows for a reduction in learning curves.

Jinja2

Jinja2 is the default HTML template engine supported by Flask (Flask 2016). Furthermore, it is also commonly used (Jinja 2016). Jinja2 was incorporated as it provides us with the ability to pre-render HTML pages on the server while also decoupling HTML code from the application code, allowing a cleaner and more easily understood codebase (Flask 2016).

Client

A requirement of the client was platform independence. Due to this requirement, the web-browsers targeted conform to platform independence and, additionally, are open source and relatively standards compliant (Widder 2016). The libraries for JavaScript and frameworks for CSS utilised for the client-side is automatically downloaded from the web-server, and consequently require no user-interaction.

JavaScript Libraries & Features

JQuery

JQuery is a small yet feature-rich JavaScript library that allowed us to easily interface the functionality of the backend for user interaction.

One aim was to achieve a lightweight interface with control over its structure, along with necessary rendering of dynamic content. Furthermore, it is well documented and widely supported, and all team members are familiar with JavaScript. It is necessarily compatible with our targeted browsers.

Ace

Ace is a portable, syntax-highlighted code editor developed in JavaScript that is embeddable into any web or JavaScript application. It includes the performance and features of common popular editors, is highly-customisable and has syntax-highlighting support for over 110 languages. Due to its customisability and performance, it is ideal for any user and web-applications. It is the basis for module development, and while modules are only concerned with one language (C++), Ace provides client-side support for any language that may be added in the future as an extension without any new technology.

CSS Frameworks

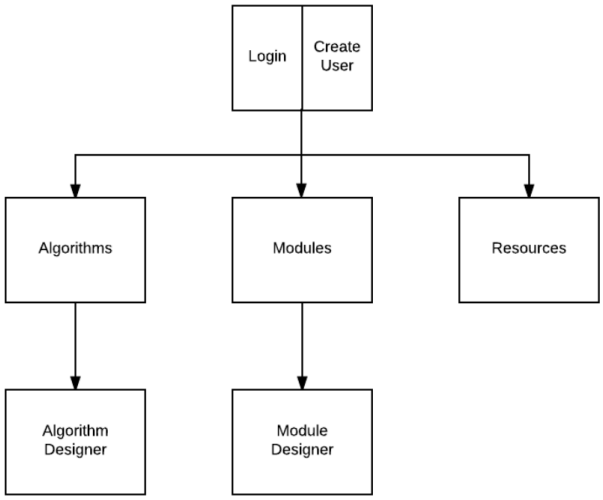
Bootstrap

Bootstrap is one of the most popular HTML, CSS, and JavaScript frameworks to date. Bootstrap was targeted for its features regarding CSS development in order to maximise timely design by utilising the available resources offered by the framework. Bootstrap v3.3.6 was implemented in this project, as it was the latest during development.

User Interface

UI Storyboard

Before the design is discussed in detail, a brief overview of the UI can be seen from the storyboard flowchart. This storyboard is the logical layout of how to access different parts of the system, and influenced the design of each of the forms and pages in VividFlow.

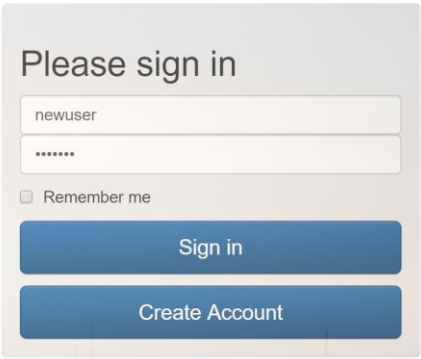


UI Design

Login

The login form is the first point of contact for a user accessing our system. It consists of a modern and minimal design, a theme that is carried throughout the site. The login form promotes both the logging in of existing users and the creation of new accounts: the only difference being which button is engaged. Logging in will redirect the user to the homepage that allows for core system interaction to take place. Creating an account via this form subsequently invokes a login, initiating the same redirection.

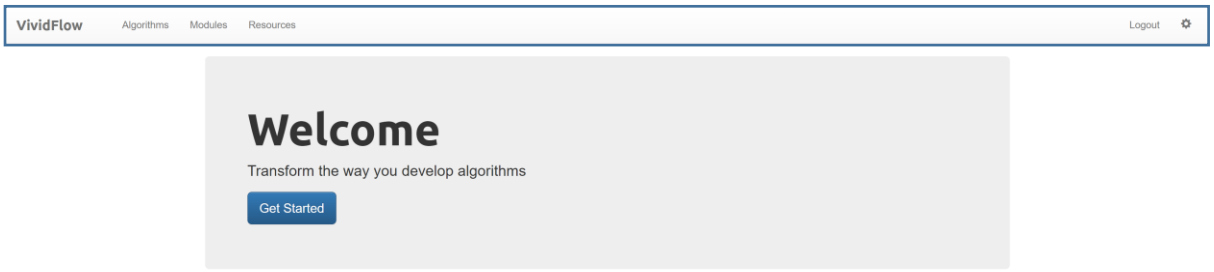
Optionally, a user may tick the “Remember me” checkbox, which conveniently invokes the necessary functionality for keeping a user’s session alive, so they do not need to unnecessarily login when system interaction is sporadic.



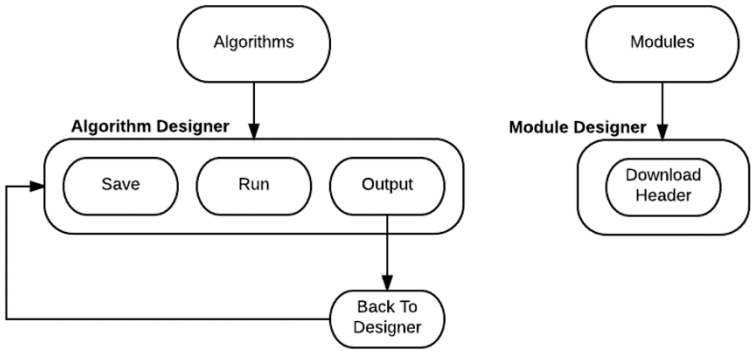
Getting Started

Upon logging in, the user is presented with the following homepage.

Located at the very top of the view, as outlined, there is a navigation menu with an alluring design that conforms to the overall theme of the application. This navigation menu is context-based in that the links accessible through the menu change dynamically depending on where the user is in the application. The elements within this navigation bar is the basis for all contexts, allowing links to primary site features existing throughout the site regardless of location.

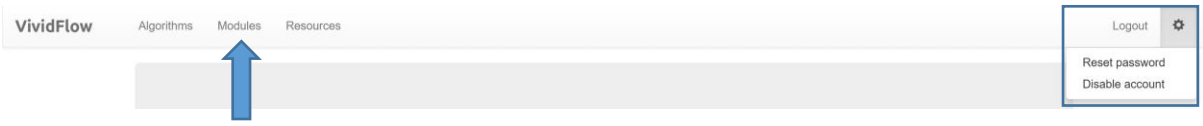


The following diagram illustrates the alternate contexts which are available. These contexts implicitly inherit the elements that are outlined in the previous image. Note that the additions supply functionality relevant to location.



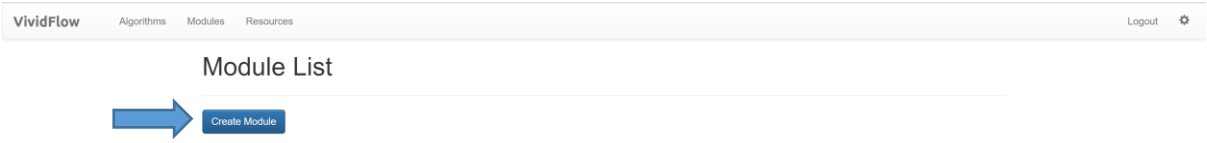
As outlined in the image below, functionality related to user management is appropriately separated from core system features. This design promotes the management theme which exists throughout the application. For instance, management of algorithms, modules, and resources are also interfaced separately. Isolating functionality in this fashion promotes intuitive navigation, allowing the user to substantially narrow their search to particular locations when looking for specific functionality.

To get started, the user may begin by navigating to the module lists, as directed by the arrow in the following image.

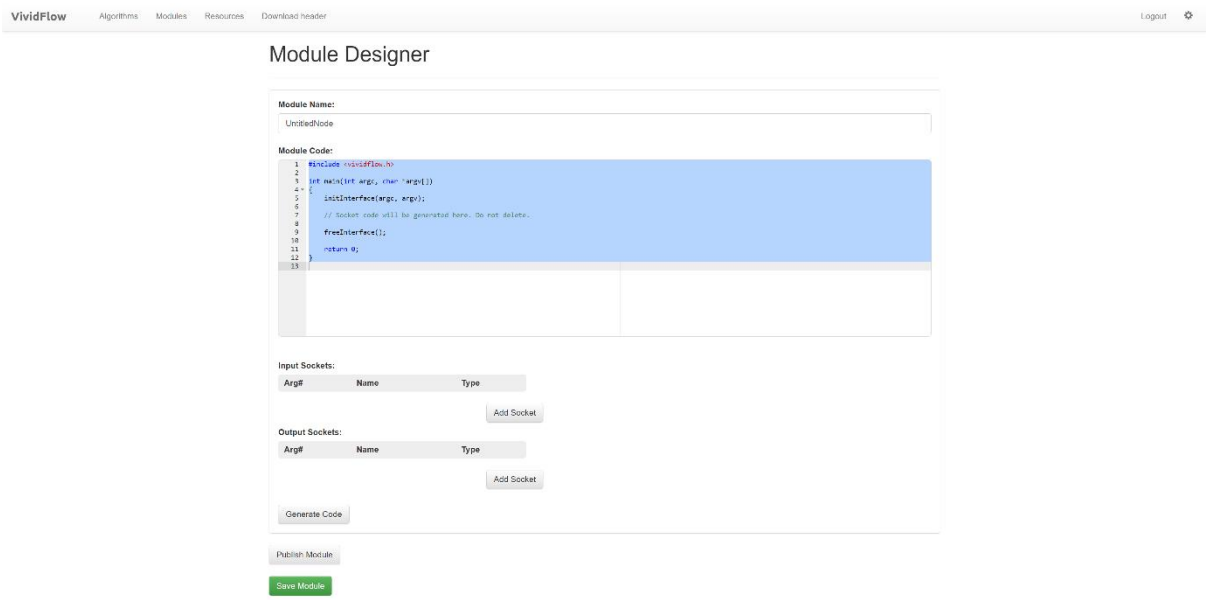


Modules

The initial view presented upon navigation is the modules' list. This view is responsible for providing an overview of existing modules created by the user, as well as the starting point to designing entirely new modules. At least one module is a necessary prerequisite to designing an algorithm, and creating one is as simple as engaging the button directed by the arrow presented as following:

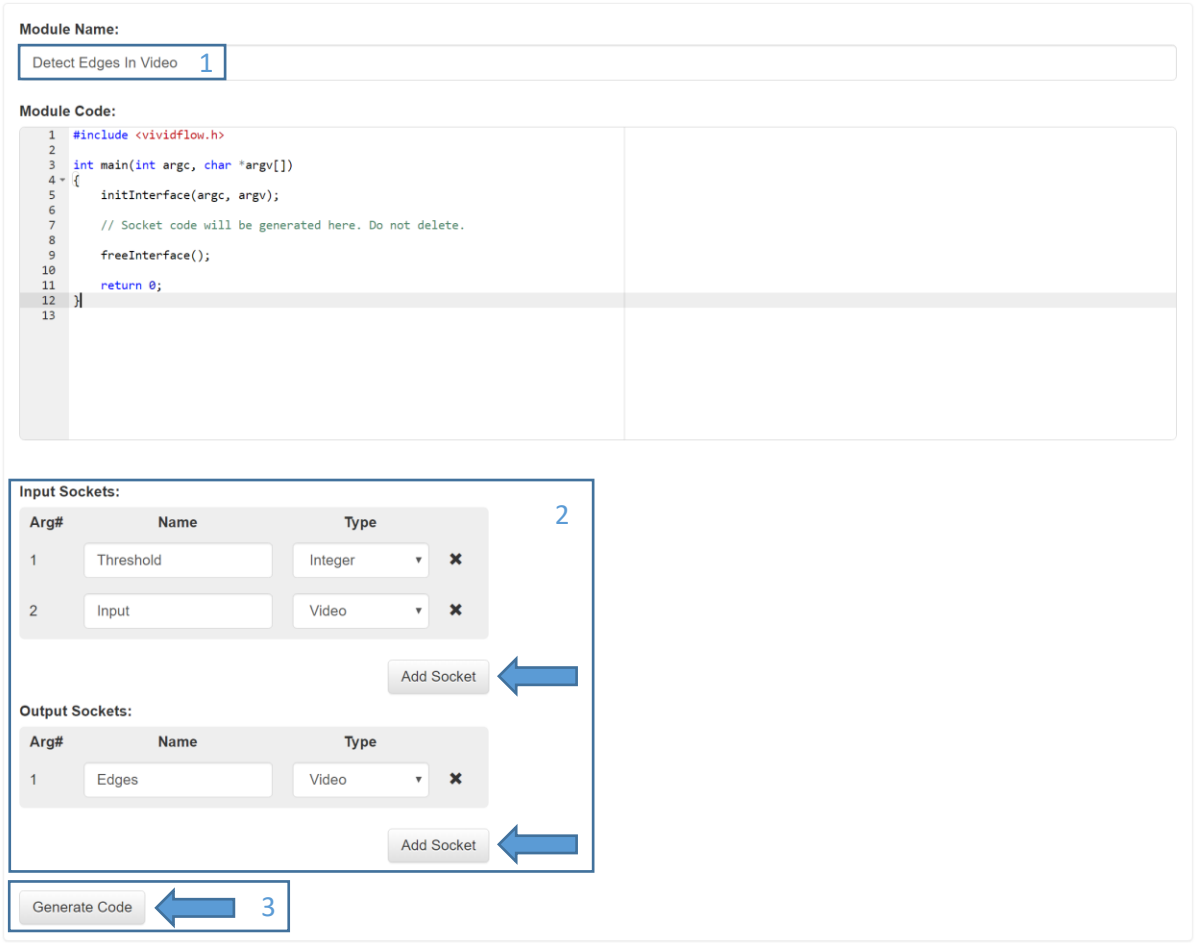


Upon clicking, the user is subsequently redirected to the module designer where placeholder data is populated into editable text-fields awaiting modification. The initial data exists to provide a starting point for the developer. This is particularly important with the second text-field, where the module's code is defined. Initially, it is composed of a simple template which contains the necessary basis for getting started with core development of the module. This template is further extended by including a feature which generates the code for the input/output sockets specified by the user.



Proceed to substitute the placeholder data with data relative to the module by:

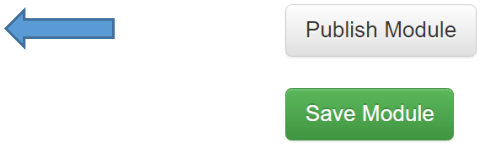
1. Setting the module's name
2. Adding input and output sockets necessary for the module
3. Clicking "Generate Code"



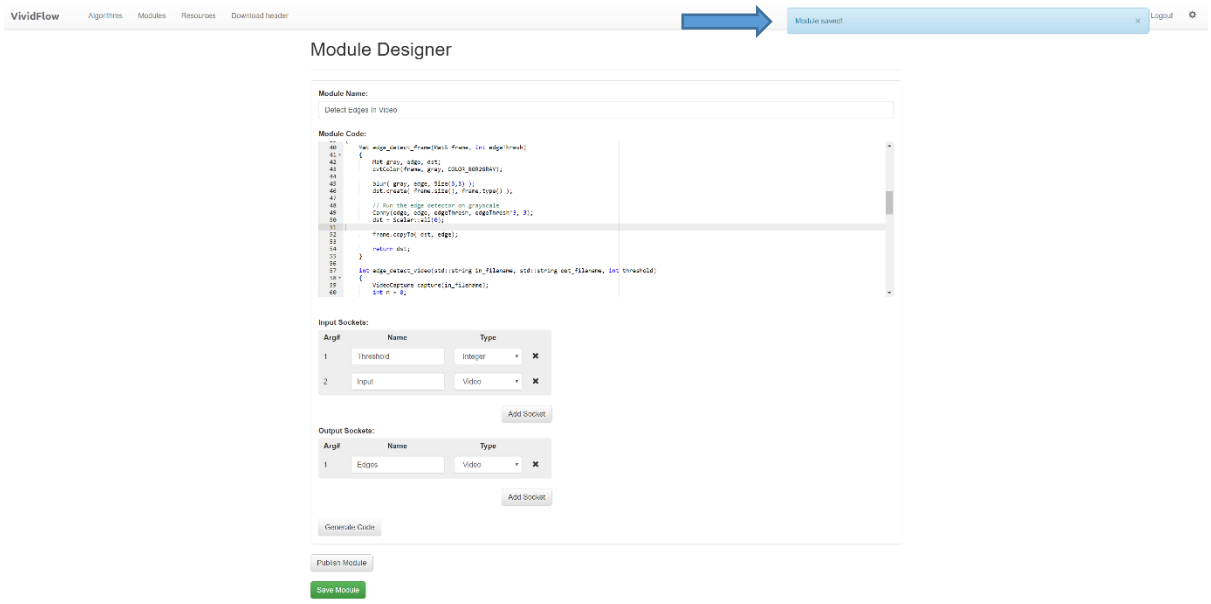
Upon clicking the button to generate the code, the text-field containing the module's code is subsequently updated:

1. Header comments describing the input/output sockets defined.
2. Generate code that handles and references the supplied input/output sockets.

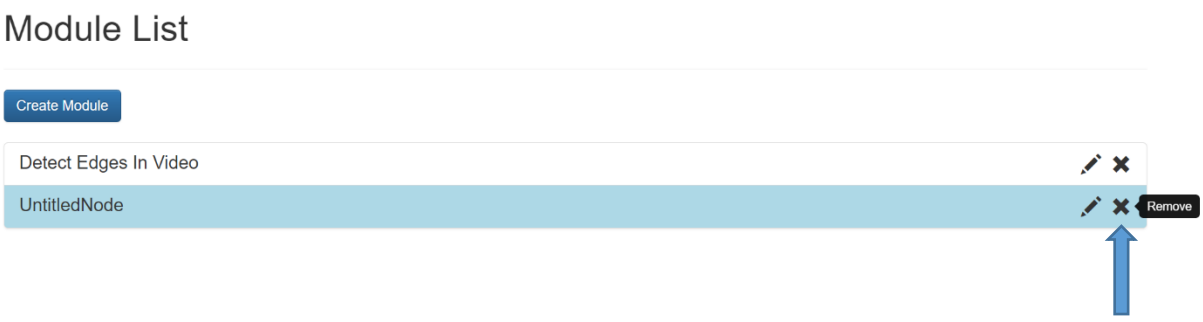
Upon satisfying the fields, the module may be published if development is complete. Alternatively, it may be saved and returned to later. The module must be published in order to apply it in algorithms.



A notification stating that the module was saved is added to provide systematic feedback to the user, an additive to the theme which further promotes interactivity between the system and the user.

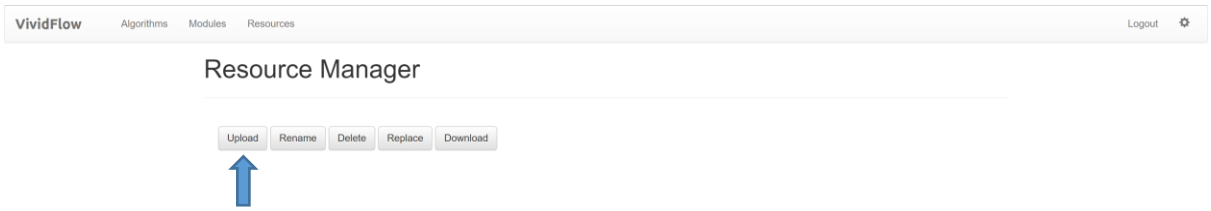


Revisiting the module list shows the newly created module that can be used within algorithms. This view is designed to have module management functionality self-contained in that modification and removal can be executed trivially through elements within each cell.

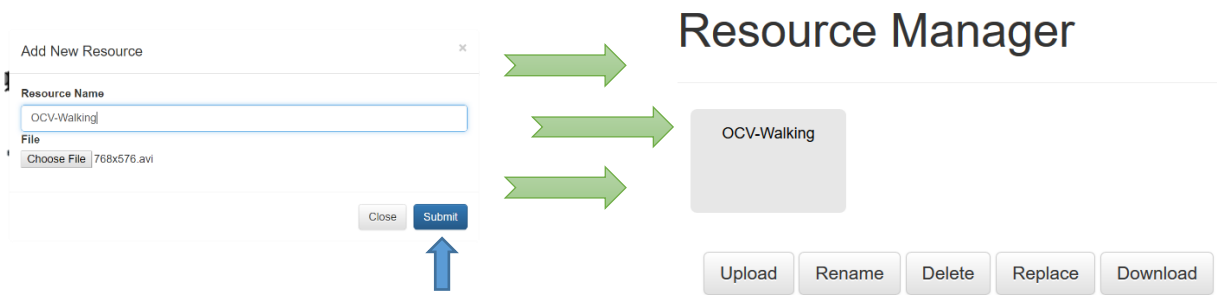


Resources

As algorithms apply modules on resources such as images or videos, the following step is to define the resources required. The resource manager is the primary interface for managing these resources. The resource manager holds the self-contained and minimal consistency that is available across the application. In order for an algorithm to process the given input file expected by the previously defined module, the resource must be uploaded.



Satisfying the dialog prompting a name for the resource and its location relative to the users file system will result in the resource being uploaded to the system and displayed as shown. This view allows the user to easily make changes to or delete existing resources simply by clicking on them and engaging the appropriate button.

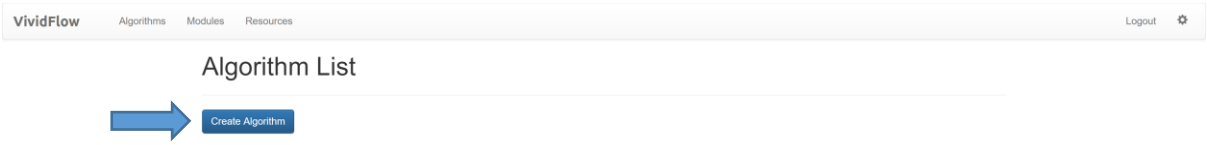


The module is defined and the resource is uploaded. Now that the base requirements for the envisioned algorithm are fulfilled, the algorithm can begin being designed.

Algorithms

The initial view presented upon navigation is the algorithm list. This view is responsible for providing an overview of existing algorithms created by the user, as well as the starting point to designing a new one. Its overall design is consistent to that of the module list.

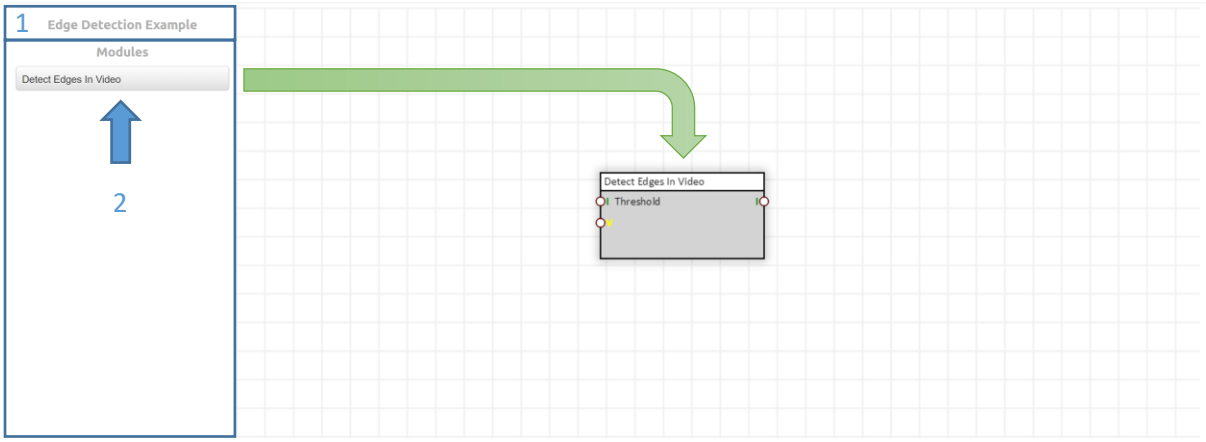
Creating a new algorithm is as simple as engaging the appropriate button, as directed by the arrow.



Upon clicking the aforementioned button, the user is redirected to the algorithm designer. The algorithm designer is the interface in which algorithm development occurs and rapid prototyping ensues. The algorithm designer divides the functionality necessary to define and prototype algorithms like so:

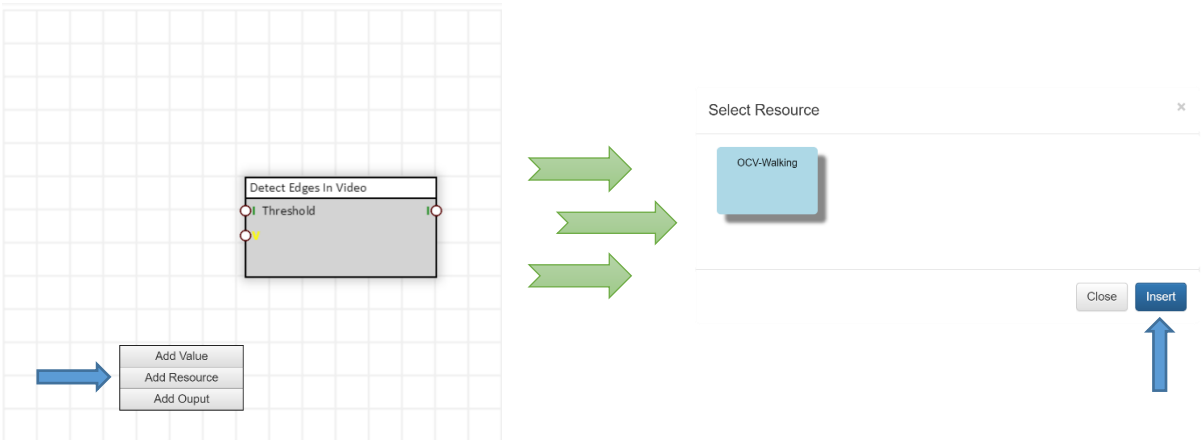
1. The algorithm's name
2. A list of existing modules
3. A node-based gridded canvas for core development
4. Extensions to the navigation menu relative to algorithm prototyping

The user is therefore expected to define the algorithm with respect to these fields. The algorithm is given a name, and any modules to be coalesced into the algorithm are added by clicking the module from the list.

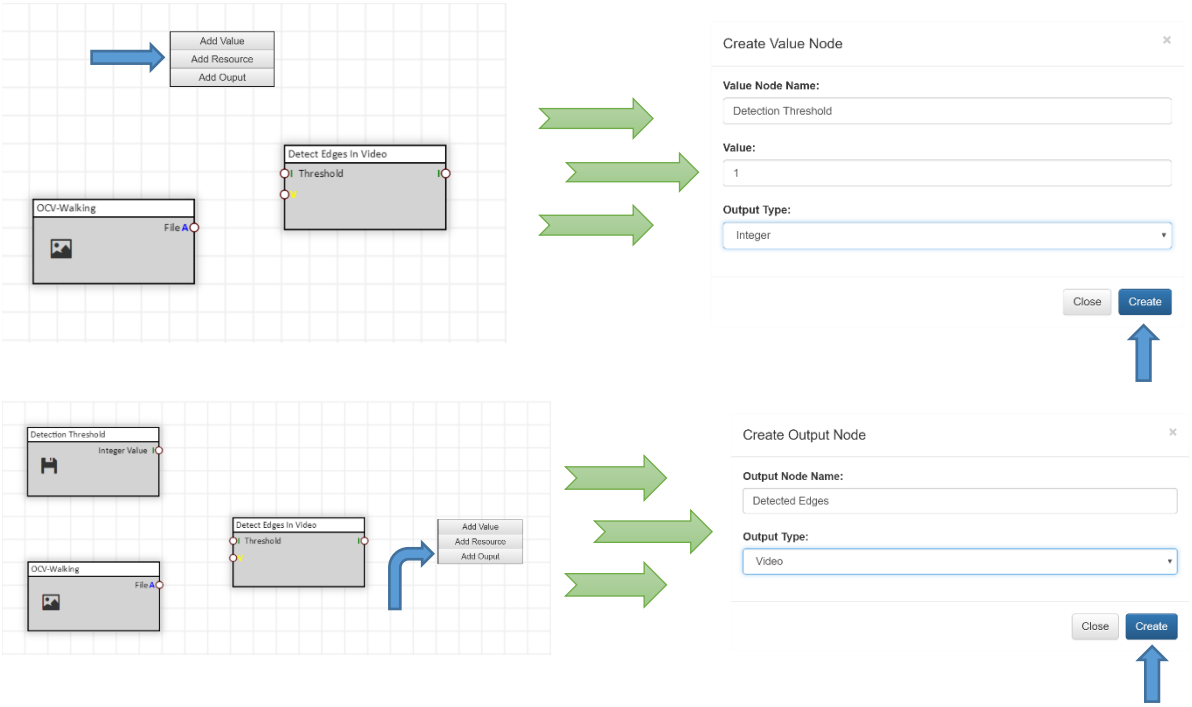


Following this, the algorithm expects a resource to apply the algorithm onto. The canvas provides a context menu that allows the user to add resources, as well as other types of nodes. The menu is accessible upon right-clicking within the canvas, where the position of the right-click determines the contents displayed by the list. The context menu is designed in this way to provide a means of accessing core functionality in a way that speeds up production by limiting movements, as well as reducing polution of the user interface by having functionality be self-contained and dynamically accessible based on context that is intuitive, rather than static menus outside of the canvas.

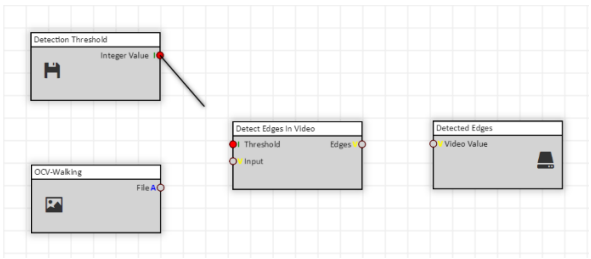
The right-clicking of empty spaces, by design, allows the addition of new nodes. Any resources required by inserted modules may be added by using the context menu:



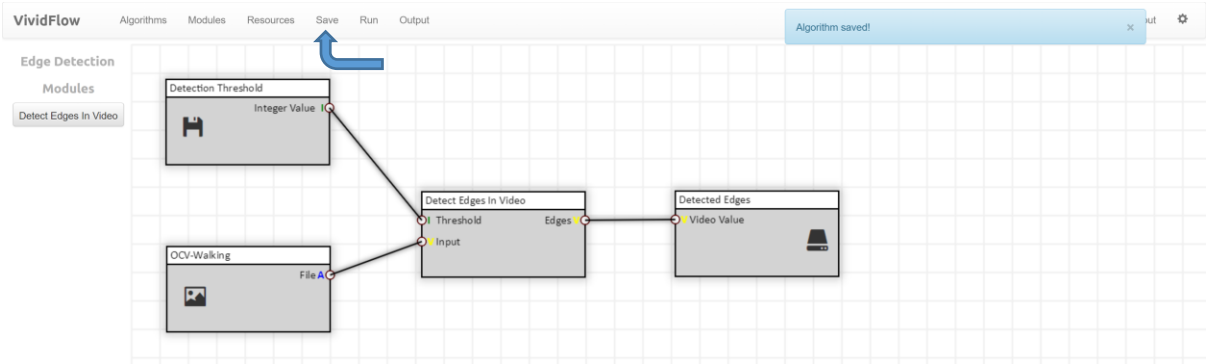
The modules and resources necessary for the algorithm are in-place. The module needs to be satisfied in terms of its inputs and outputs. This is achieved similarly to the addition of resources via the context menu.



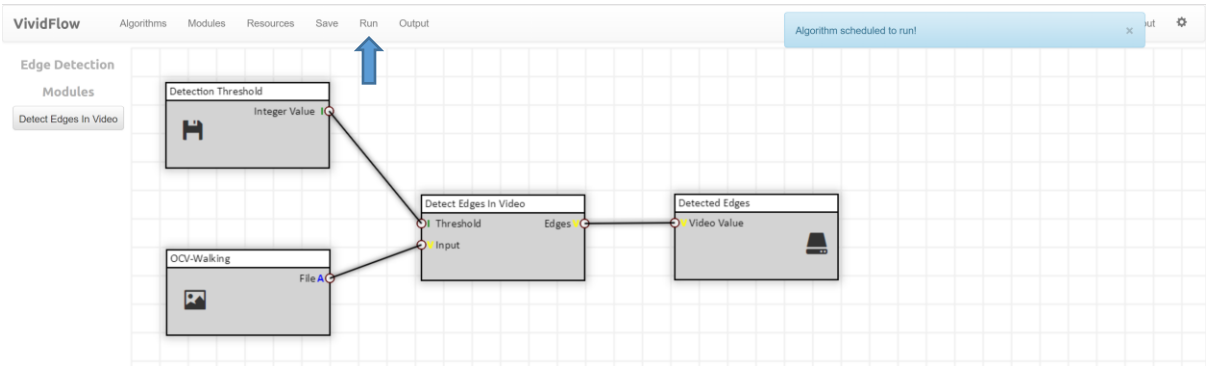
Upon creating all the nodes which represent the algorithm, the input and output sockets need to be linked as defined by the module(s). By design, compatible data types will cause the end-points to be highlighted in red. Furthermore, the nodes include distinct icons to distinguish between the node type and data type. Attempting to link incompatible types will result in the link failing. This reduces the mistakes potentially imposed by a developer.



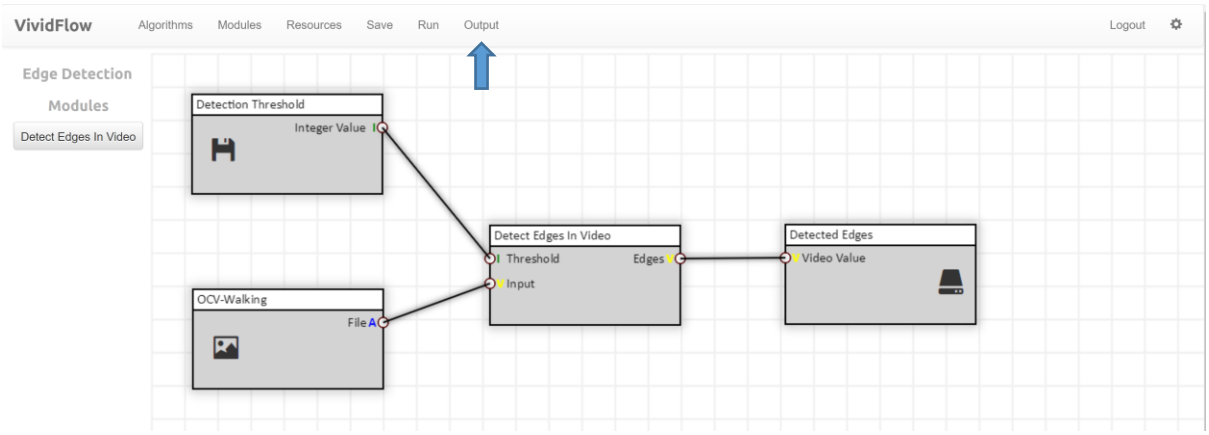
Upon satisfying all necessary requirements, the algorithm is necessarily saved prior to its execution. When saved, a notification akin to the one presented by the module designer appears. This provides consistency in theme between pages, further building and satisfying user expectation of interaction from the system.



When the algorithm is saved, it may subsequently be scheduled for execution in order to test the correctness of the designed algorithm. A notification correspondent to this action also presents itself.



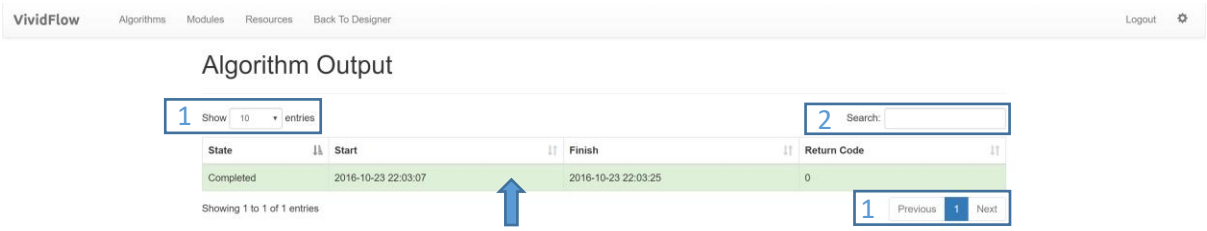
Details of the scheduling/execution can be observed directly following the scheduling, allowing for rapid review.



When the algorithm has been scheduled, a new entry in the table of outputs is added. The entry's state will be set to "Completed" when the algorithm has been compiled and executed, as per the scheduler. If the algorithm compiled and executed successfully, the entry is highlighted in green to promote the success aesthetically. Furthermore, the table has functionality considerate to a high number of evaluations:

1. Entries may be limited; divided into pages
2. Outputs may be found via search

Clicking the cell allows the user to view the output specified by the algorithm.



Upon clicking the cell representing the output, the user is presented with a dialog that contains downloadable references to the output files. When finished viewing files, the dialog may be closed and further developing and prototyping may ensue.

Architectural overview

Front End

The frontend of VividFlow heavily relies on JavaScript and Jinja2 templates that are rendered from the backend. The front end makes many calls to the backend to submit and request data. This communication gives us the capacity to make the site responsive and have an extra layer of abstraction from the backend.

User Management

The website is user sensitive and requires a user to log into the system to access all the functionality. Users only see the things they have created in the system because access is restricted on the backend. The login page lets a user login or create an account. The login takes the username and password and sends it to the backend to be compared for a valid login. If the login fails then the user is directed back to the login page, otherwise they proceed into the system. When a user tries to change their password, or disable their account the front end will send a request to the backend to do this.

Algorithm Designer

The algorithm designer page is broken up into three areas

- Algorithm Canvas
- Module Selector
- Functional Buttons

Algorithm Canvas

The algorithm canvas uses a HTML Canvas to display the algorithm. Each algorithm component comes from a draw object instance. These components are nodes, sockets and links. Since they are all draw objects they are able to share the same functionality. It then becomes trivial to have the objects drawing, detecting click events and checking if a position is on the object.

We have opted for a MVC (Model-View-Controller) like design pattern where we have model objects (algorithms and components), control functions (events, decisions) and view objects (drawing). This gave the algorithm canvas the flexibility and structure we required to build a complex node based editor.

The default mouse operations are disabled on the canvas so that we can catch left and right mouse clicks and deal with them.

The positions of the draw objects are relative to the canvas offset. Dragging the algorithm is achieved by changing the offset of all the draw objects.

The context menu shows different items depending on what draw object is underneath the mouse at the time. The menu is give based on the type of the draw object since JavaScript allows for the use of instanceof.

Due to the speed that the canvas can render the algorithm the Algorithm Canvas has no optimised drawing techniques. When something changes, the entire canvas is redrawn. This reduces the complexity of handling individual events and drawing circumstances. Also, the construction of the draw objects from the algorithm is quick so each change rebuilds all the draw objects from the algorithm, this reduces the chance of conflicts between the view and the model.

Module Selector

The module selector is a list of buttons that can be clicked on to add modules to an algorithm. The list of modules is feed from the backend through an ajax request. When a module is clicked on it will add a module to the algorithm model object and have the draw objects rebuild based on the modified algorithm. Since the modules are stored as JSON strings and the algorithm can be built from JSON strings we just pass the JSON string for a module into the algorithm and it handles building the new module node.

Functional Buttons

Save: The algorithm instance takes the algorithm data and serializes it into a JSON string. This is then sent to the backend with an ajax post. When the frontend receives a success/fail response, a message is displayed to the user.

Run: Clicking on the run button sends an ajax message to the backend to schedule a job for the current algorithm open. A success response will have a message show to the user telling them that the job has been scheduled.

Output: This will take the user to the algorithm output page that loads all the job scheduler data. The data is shown in a table. Finished jobs can be clicked on and a modal with open with the files outputted from the algorithm running. These files are stored in the public static directory so only a link is required to download them.

Module Designer

The module designer allows a user to build and modify modules. All the data in the module designer is loaded and saved through ajax requests. Modules are retrieved from the backend as a JSON string. The module is constructed from this string and then the instance of the module can be manipulated by the user in the interface. Once this is done and the user saves the module object is serialised back to a JSON string and sent to the backend to store.

A module can have input and output sockets. The sockets are separated by this grouping. To make things easier we developed a sockets class so that we can handle an inputs and outputs instances of them.

Module code is displayed with ace. This allows us to use syntax highlighting. A default code template is loaded when a new module is created. The code for sockets can automatically be generated by the

user once the sockets are defined. This is done by access the socket data and converting it into the appropriate C++ code. This is then inserted into the code over the top of a comment in the default template.

Resource Manager

The resource manager shows a user what resources they have uploaded and allows them to modify them or add new ones. The user needs to select a resource before they can modify it, this functionality is required when a resource needs to be selected for insertion into an algorithm. To reduce code and maintenance we wrote a class that is re-used for both functions. The class handles selecting, retrieving and manipulating a resource. The functions show in the resource manager hook into this class.

Uploading files required some work on the backend to all files to store and be accepted at the size we require. Users could be uploading large video files and the frontend and backend need to handle this. AJAX calls are made to upload and modify a resource. Resources are stored in the public files of the site and can be downloaded by a link based on the name of the file and the id in the database.

Site Navigation

The combination of Flask and Jinja2 allowed us to use a template engine to render our webpages. The rendering in Jinja2 let us use inheritance to minimise code duplication. The user site pages are all inherited from a base template. This let us have a consistent style across the site and changes to all the pages could be done from one file.

The base template has content blocks that can have page information inserted into them. These blocks are

- css: Designated for CSS
- js: Designated for JavaScript
- header_items: Designated for adding extra header list items
- content: Designated for page content

Sometimes we need to override a content block, like if we didn't want to show the header on a page we could blank it out or replace it with something else. If we do not change the block, then the default code in the template is used. The blocks in the template that can be overridden are

- content_override: replace the html body
- header: replace the header bar

Back End

The back end of the system is developed in Python. It uses Flask as the application framework and communicates with a MySQL database. There is a separate process that runs to schedule and monitor jobs.

Representation of Algorithms

Nodes are objects that can be placed into an algorithm. The nodes can represent data that is put into the algorithm, data that comes out of the algorithm, or how data should be processed in the algorithm.

Nodes have multiple types. The first group of nodes are “DataNodes”, which are then broken into 3 further nodes. The first type is “ResourceNode” which can represent resources that have been uploaded by the user through the resource manager. This allows larger resources to be input into the algorithm. The idea of these being uploaded separately mean that they don’t need to be copied to working folders and can be updated easily across algorithms. When a resource is uploaded, it is written to disk. When an algorithm runs, these paths are passed directly to processing modules.

The second type of DataNode is “ValueNode” which can represent literal values, such as integers, floats, strings etc. These can be placed in algorithms and have their values set and updated through the algorithm designer. When the algorithm is run, these values are written out to files which are then passed into the processing modules.

The final type of DataNode is “OutputNode” which can be placed to specify output data from the algorithm. Any data that is plugged into these nodes will be accessible for download via the algorithm output.

ModuleNodes are instances of modules that have been defined by the user. A module as defined through the module designer, and a ModuleNode that is placed in an algorithm are represented by the same class in code since they share so much data and functionality. Modules can be in 3 states. Any module that is in development or published can be placed into an algorithm, modules that are retired cannot be placed into an algorithm. Upon load of an algorithm, if it contains a module that was in development, the backend tries to swap it out with the latest version of the module. Modules are associated with a C++ program that can be updated through the front end. When the program is updated from the frontend the compiled version is deleted. Modules are compiled when the algorithm runs. Inputs and outputs that are defined in the module designer appear as sockets in the algorithm designer. These sockets are passed into and out of the C++ program as command line parameters that specify filenames which the program can then open and process appropriately.

An algorithm is the main product of the system. The algorithm will contain instances of modules, data nodes and links.

The algorithms are executed by generating a makefile. Makefiles are generated by generating a step for each ModuleNode. ModuleNode prerequisites are determined by traversing the graph and determining any ModuleNode that feeds into the current ModuleNode. A final rule, RunAlgorithm, is determined by identifying any ModuleNode that feeds into an OutputNode.

Running Algorithms

Due to the nature of computer vision algorithms, many of these algorithms could take a long time to run, so they need to be separated from the web application. To run an algorithm, a scheduled task is

added to the database and put into the “pending” state. A daemon runs in the background and will check for scheduled tasks in a pending state. The daemon will spawn a separate job worker for any pending tasks it finds up to a configured concurrent job limit.

The JobWorker will fetch the assigned task from the database and immediately mark it as “InProgress”. The job worker will then fetch the algorithm, generate the makefile and run the algorithm. Any files that are generated and connected to output algorithms are added to the ScheduledTask as an output, and a log file is also added to the list of downloadable files.

All modules in the system attempt to read their configuration settings from a common settings file. The settings are stored in a series of hierarchical JSON objects in a text file. Some path settings are stored as relative paths in the settings file, and are expanded at load time to be absolute paths. These calculated settings are still stored in the configuration file for reference, however their value is saved as “DYNAMIC” to inform anyone modifying the file that the setting will be ignored at runtime.

Communication and Serialization

Communication between the frontend and backend is handled via a REST-like API. The API is not a true RESTful API because actions are mostly represented in the URL path, not by the HTTP methods of GET, PUT, POST, DELETE. The API is designed to be as stateless as possible, and typically the only state that matters is the user session to ensure that no unauthorised actions are performed on the system.

All critical information is conveyed back and forth between the client and server as JSON objects. This had some important design and implementation issues that needed to be considered so that code size didn’t increase drastically and to reduce the possibility for errors to be introduced. The system works the same in Python and JavaScript.

Any class that will have instances passed between the layers inherits from a base class, SerializableObject, and is registered with a static class, SerialObjectTypes. The base class, SerializableObject provides a method called serialize_json(). This will return a string version of the JSON object that is desired. This is generated by calling a function called build_json_dict() which is overridden in child classes. The child class versions need to call the super version of this function, and then append any additional values to fully capture the objects data.

To rebuild an object into a native instance from a JSON object, the static class SerialObjectTypes provides a function build_object_from_json(json_object) which will inspect the JSON object, identify the object type and create a new instance of that class. It will then pass the JSON object to the class by calling reconstruct_from_json(json_object) on the new class. This function is automatic, and uses the reflection properties of the languages to appropriately populate variables from the JSON object. If the JSON object contains an embedded child class, it will automatically call build_object_from_json to instantiate the child class.

This serialisation system means that a new class only needs to define how to serialise it’s new members and the serialization system will automatically handle all other functionality.

Users

User management is handled on the backend with Flask sessions. The session allows the system to track which user is logged in and display only the associated algorithms, modules and resources. When a user logs in from the home page the session is started. All database queries will be filter with the user id.

Database

The database is the backbone for the storage of data in our system. All the algorithms, modules, resources, users and scheduled jobs are stored here. The backend of the system controls access to the data.

Since we have multiple users in our system we need to maintain the user relationship between algorithms, modules and resources. These are restricted in the backend and the database stores the relationship that the user has with these components.

Due to the complexity of data in the algorithm we opted for JSON strings to store objects. This removed the need to constantly track ids and relationships. We have moved towards more of a relation document based solution where an algorithm is mostly a self-contained document. This simplified the code in the frontend and backend as well as reducing the complexity of the database design. The only issue we had was with modules which can be updated and the algorithm needs to have the appropriate version of the module. So, when an algorithm is loaded the modules are checked and updated where necessary.

Since our system needs to support large files we decided to not store files in the database. It also reduces the work required to run an algorithm as we would have had to extract all the files and put them in a location that the algorithm can access.

Tables

User

The user table maintains a list of the users that have registered on the website. The username and password are stored so that the credentials can be checked when a user tries to login into the system. All other tables have a foreign key back to user so that user created content is only shown to the user that created it.

Algorithm

The algorithm table stores an algorithm created in the front end. Since the algorithm is a very complex data structure we decided to store the object as a JSON string. When saving, the algorithm object is serialised into a JSON string and stored in a mediumtext field. When retrieving an algorithm from the database the JSON string is de-serialised into the algorithm object again. Useful properties of the algorithm are stored in the tables fields, such as id, name, version.

Module

The module table is similar to the algorithm table; the complex data structure of the module object is stored as a JSON string. The C++ code of the module is stored separately in the file system since

we need to use the code for compiling. The code file is named based on the id of the module. Modules also have states for “InDevelopment” and “Published”, these let the user specify when a version of a module is complete.

Scheduled_tasks

The scheduled_tasks table keeps track of the algorithms that are to be run, running and have run. The states of the task give an indication of where it is up to. It can have the states Pending, InProgress or Completed. The tasks relate to the algorithm by its id, so a list of tasks for an algorithm can be easily looked up.

Resource

Each resource a user creates has a record in the database. Since the resource is a simple object there is no need to use JSON strings. The file is stored to the server in the website's static content so that the user can view/download the resource when required. Resources can be deleted; however, they are only marked as deleted, and the file will remain on the server.

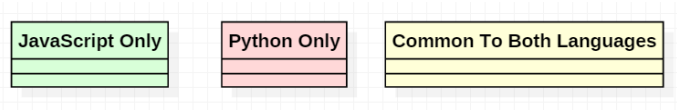
UML Class Diagrams

Due to the nature of languages used in this project the UML diagrams represent the conceptual design in terms of constant and static variables, abstract classes etc., but they are not defined as such in code due to the languages lack of support for these concepts.

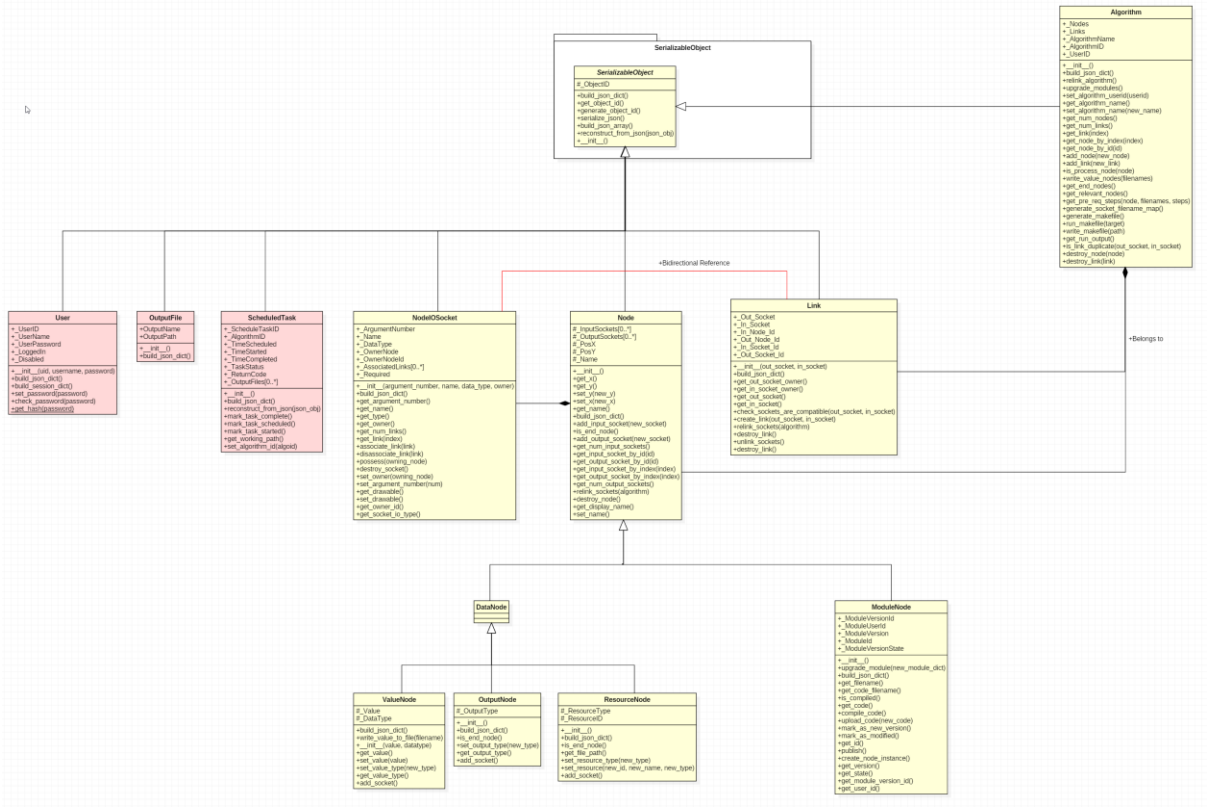
The core classes are represented in both languages, however not all functions are implemented in both functions. Functions were only implemented in either language if they were required. This helped to reduce code overhead and the cost of maintenance of functions.

Since both primary languages have dynamic typing, types have not been documented in return values or input types since in many instances multiple types can be used, and in instances where certain types are expected our variable names are designed to convey such information. The languages do not support all concepts of OOP that can be represented in UML, but the UML diagrams have class accessor scope specified as per our design, and our name conventions. Our naming conventions were such that any member function or variable that was specified with a single underscore was protected, and double underscore was private as per the PEP8 python standard.

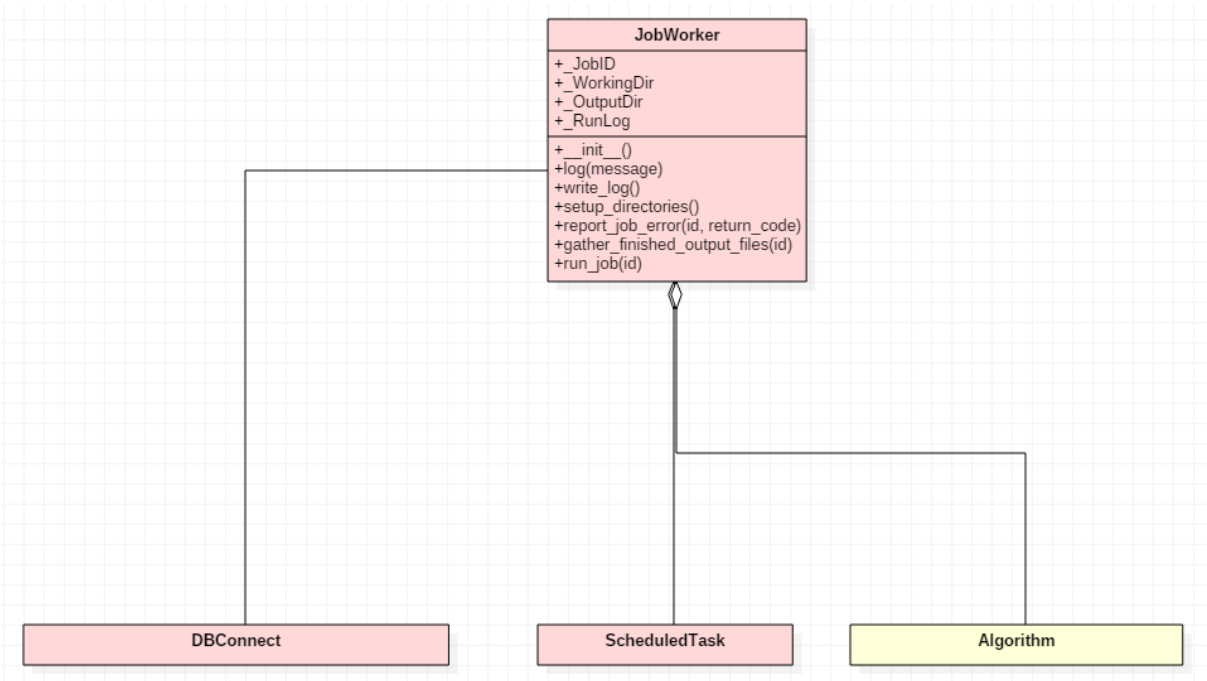
Key



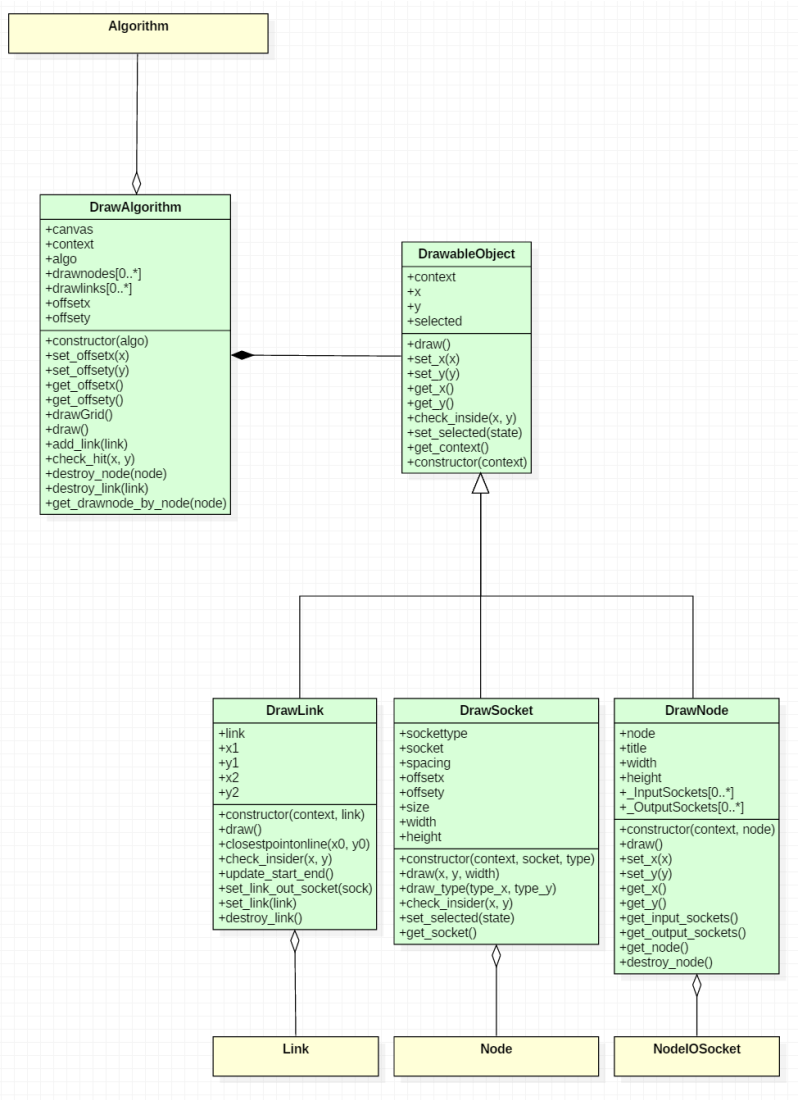
Serializable Objects and Algorithm structure



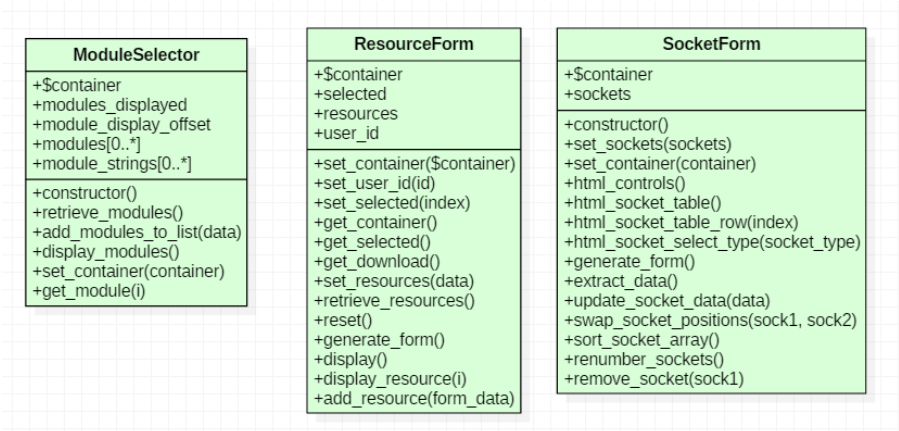
Job Worker



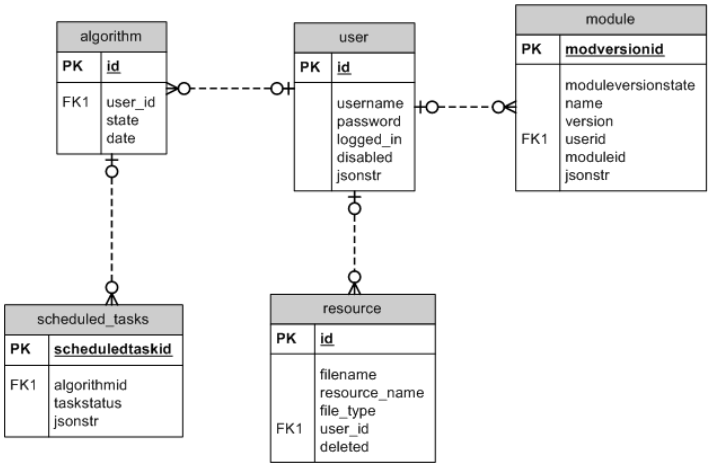
Algorithm Designer Canvas Rendering



Front End Forms



Database Entity Relationship Diagram



Testing

Testing VividFlow is a vital task as the system is large and complex. Due to the fact that the system incorporates 2 languages and runs across multiple computers, unit tests were deemed impractical in the time that was allocated for the project. Instead the approach taken was to use testing scripts that were followed as new use cases were fully implemented. These scripts were then tested every sprint thereafter. The small team size allowed this method to work, however with a larger team or longer project, this would need to be revisited.

Testing was conducted throughout development. As new features were implemented and a use case was covered, a test script was written to cover that use case. These test scripts were used by the developers to repeatedly test use cases when work was performed on code that could affect those areas. If a test failed, developers could not commit code until it was fixed, or if it was discovered to be broken after the fact, the developer would raise it with the team. The issue then could be fixed straight away or added to the next sprint.

Test Name	**Create Module** **ID: 1**
Expected Result	A module and all associated information is saved and retrieved
Procedure	1. Navigate to Module List page 2. Click Create Module 3. Change module name 4. Enter some code 5. Add input 2 sockets 6. Add 2 output sockets 7. Save module 8. Refresh page 9. Ensure all configuration was retained.
Result	The changes remain in the module designer

Test Name	**Update Module** **ID: 2**
Expected Result	An existing module and associated data is updated on the system.
Procedure	1. Navigate to the Module List page 2. Click the module to modify 3. Change the desired input fields that reflect the module 4. Click Save Module 5. Ensure all configuration was retained
Result	Modified properties are retained correctly upon updating.

Test Name	**Create New Algorithm** **ID: 3**

Expected Result	A new algorithm is saved to the system and reflected to the user for development.
Procedure	1. Navigate to the Algorithm List page 2. Click Create Algorithm
Result	The algorithm is instantiated and saved to the system; the user is redirected to the development interface with respect to the created algorithm as expected.

Test Name	**Open Algorithm**	**ID:**	**4**
Expected Result	An existing algorithm and related functionality is reflected to the user for viewing and modification/development.		
Procedure	1. Navigate to the Algorithm List page 2. Click the algorithm to open		
Result	The algorithm and related functionality is correctly reflected in the development interface upon opening.		

Test Name	**Add Module to Algorithm**	**ID:**	**5**
Expected Result	A module is linked with an algorithm; a module node is added to development interface.		
Procedure	1. Navigate to the Algorithm List page 2. Click the algorithm to modify 3. Click the desired module from the module list 4. Click Save 5. Ensure the module node was populated correctly		
Result	References to the module are recorded as expected; the module node exists in the interface consistently.		

Test Name	**Add Resource to Algorithm**	**ID:**	**6**
Expected Result	A resource is linked with an algorithm; a resource node is added to development interface.		
Procedure	1. Navigate to the Algorithm List page 2. Click the algorithm to modify 3. Right-click on the grid-layout 4. Click Add Resource 5. Click the desired resource 6. Click Insert 7. Click Save 8. Ensure the resource node was populated correctly		
Result	References to the resource are recorded as expected; the resource node		

	exists in the interface consistently.

Test Name	Add Value to Algorithm	ID: 7
Expected Result	A value is linked with an algorithm; a value node is added to development interface.	
Procedure	1. Navigate to the Algorithm List page 2. Click the algorithm to modify 3. Right-click on the grid-layout 4. Click Add Value 5. Change the Value Node Name input field 6. Change the Value input field 7. Select an Output Type from the drop-down menu 8. Click Create 9. Click Save 10. Ensure the value node was populated correctly	
Result	References to the value are recorded as expected; the value node exists in the interface consistently.	

Test Name	Remove Module from Algorithm	ID: 8
Expected Result	The module is unlinked from the algorithm: the module node and its associated socket links are removed from the interface.	
Procedure	1. Navigate to the Algorithm List page 2. Click the algorithm to modify 3. Right-click on the module node 4. Click Delete Node 5. Click Save 6. Refresh to ensure the module node no longer exists	
Result	References to the module are removed; the module node is permanently removed from the interface.	

Test Name	Remove Resource from Algorithm	ID: 9
Expected Result	The resource is unlinked from the algorithm: the resource node and its associated socket links are removed from the development interface.	
Procedure	1. Navigate to the Algorithm List page 2. Click the algorithm to modify 3. Right-click on the resource node 4. Click Delete Node 5. Click Save 6. Refresh to ensure the resource node no longer exists	

Result	References to the resource are removed; the resource node is permanently removed from the interface.

Test Name	**Remove Value from Algorithm**	**ID: 10**
Expected Result	The value is unlinked from the algorithm: the value node and its associated socket links are removed from the development interface.	
Procedure	1. Navigate to the Algorithm List page 2. Click the algorithm to modify 3. Right-click on the value node 4. Click Delete Node 5. Click Save 6. Refresh to ensure the value node no longer exists	
Result	References to the value are removed; the value node is permanently removed from the interface.	

Test Name	**Create an Algorithm Link**	**ID: 11**
Expected Result	A link is created in the algorithm between two sockets	
Procedure	1. Select an algorithm from the Algorithm List 2. Add a value node for an integer 3. Add an output node for an integer 4. Left click and drag from the value node output socket to the output node input socket.	
Result	A link will be displayed between the two sockets	

Test Name	**Run Algorithm**	**ID: 12**
Expected Result	An existing algorithm is compiled and executed; details of its execution are logged to the system.	
Procedure	1. Navigate to the Algorithm List page 2. Click the algorithm 3. Click Run	
Result	The algorithm compiles correctly and executes as specified; execution is logged to the system as designed.	

Test Name	**View Algorithm Output**	**ID: 13**
Expected Result	A table with each entry representing each algorithm run is presented; the entry contains references to downloadable output files.	
Procedure	4. Navigate to the Algorithm List page 5. Click the algorithm	

	6. Click Output 7. Click the desired entry 8. Click each hyperlink to download all output files 9. Ensure all outputs are as expected
Result	The table reflects all recorded outputs correctly: each entry contains references to existing, accessible output files.

Test Name	**Save Algorithm** **ID: 14**
Expected Result	An existing algorithm is updated on the system; committed changes reflect in the development interface upon saving.
Procedure	1. Navigate to the Algorithm List page 2. Click the algorithm 3. Click Save 4. Refresh to ensure changes were committed
Result	Any modifications made are correctly updated in the system, and are reflected permanently in the development interface.

Test Name	**Add User** **ID: 15**
Expected Result	A new user will be populated in the database with username and password
Procedure	1. Enter user details into website 2. Submit details 3. User logins in with account created
Result	User was populated in database, can successfully login

Test Name	**Delete User** **ID: 16**
Expected Result	A user in the database will be marked as disabled
Procedure	1. User logins in 2. Clicks on user control 3. Clicks Disable 4. Logout 5. Try logging in again
Result	User account is disabled and user can no longer login

Test Name	**Login** **ID: 17**
Expected Result	An existing user can successfully log into system, users that don't exist cannot login. User should be able to access site features now.
Procedure	1. Enter user login details 2. Click login

Result	User gets passed login screen and has access to site features

Test Name	**Modify User**	**ID: 18**
Expected Result	User can access reset password and disable	
Procedure	1. User logins 2. Clicks on user control 3. Reset Password and Disable can be used	
Result	Functions are accessed and work	

Test Name	**Reset Password**	**ID: 19**
Expected Result	The user's password changes to a new specified password	
Procedure	1. User logins 2. Clicks on user control 3. Clicks on Reset Password 4. Enters new password 5. User logs out 6. User logs in with new password	
Result	The user can now access the system with the new password	

Test Name	**Upload Resource**	**ID: 20**
Expected Result	Resource will be uploaded to server and show in Resource Manager	
Procedure	1. User opens Resource Manager 2. Clicks on Upload 3. Enters name of resource 4. Selects file from local computer 5. Submits for upload	
Result	File now appears in Resource Library with the name specified. Resource can be downloaded to confirm.	

Test Name	**Modify Resource**	**ID: 21**
Expected Result	Resource can be renamed and file replaced	
Procedure	1. User opens Resource Manager 2. Selects a Resource 3. Clicks Rename 4. Enters new name 5. Submits name 6. Selects a Resource	

	7. User Clicks Replace 8. Selects new file from local computer 9. Submits for upload
Result	Name of resource will change, resource download will be the new file.

Test Name	**Delete Resource** **ID: 22**
Expected Result	Resource no longer exists in the Resource Manager and is deleted from the server.
Procedure	1. User opens Resource Manager 2. Selects a Resource 3. Clicks Delete 4. Confirms action
Result	Resource can no longer be seen in the Resource Manager

System Requirements

Server System Requirements

The base install of Debian requires very little, the server and services of VividFlow have a small overhead as well. However, running algorithms can take a significant amount of CPU and RAM, as such, the more resources that are allocated, the better VividFlow will perform. More users will mean that more resources will need to be allocated.

The base install of Debian and VividFlow takes up about 7 Gb depending on any extras selected in the Debian installer. Using an SSD drive will also improve performance since VividFlow uses the file system heavily.

The system requires a network connection, which may need to be configured depending on the environment. It is the system administrator's responsibility to set up a FQDN if required. Since the system is designed to be used on an internal network internet speed should not be an issue. Even a stable ADSL connection will work well.

The system works well running as a virtual machine and has been tested on Oracle VirtualBox extensively.

Minimum

CPU:	1Ghz 64bit Dual Core
RAM:	2GB
HDD:	15GB

Recommended

CPU:	3Ghz 64bit Quad Core
RAM:	4GB
HDD:	64GB SSD

Client System Requirements

All the client requires is a network connection to the server and a compatible browser. The supported browsers are the current version of Mozilla Firefox and Google Chrome. It is recommended that the display has a resolution of at least 1680x1050. Clients require internet access as some external CSS and JavaScript libraries are referenced from CDNs.

Installation Instructions

1. Install Debian 8.6 AMD64 from CD or DVD.
    a. The Debian Desktop environment is not required, but can be installed if desired
    b. A network mirror must be configured for pre-requisite installation.
    c. Do not choose to install the webserver. VividFlow installs nginx and configures it appropriately for the system. Running VividFlow alongside another website or webserver is untested and unsupported.
    d. SSH is recommended for ease of management
2. Login to the terminal as root
3. Extract the VividFlow archive onto the machine
    a. tar -xvf VividFlowInstall.tar.gz
4. Navigate to the extracted directory
    a. cd VividFlowInstall
5. Run the setup script
    a. ./installvividflow.sh

Configuration

VividFlow allows system configuration through a JSON text file. The file is organised into a hierarchy to group related options. Any option in the default configuration file that is populated with "DYNAMIC" indicates that the option is available within the system, however is assigned at run time.

Database

The database settings can be modified under the "db" object. You can change the following:

hostname	The server name where the database resides
username	The username to use when connecting to the database
password	The password to use when connection to the database
dbname	The name of the database

Module System

The module system can be configured under "modules".

path_rel_code	The relative path where source files are saved
path_rel_executables	The relative path where generated binaries should be stored
path_abs_code	RUNTIME ONLY
path_abs_executables	RUNTIME ONLY
code_file_extension	The extension to be added to source files submitted to VividFlow

compiler	The compiler binary
language-standard	The language standard the compiler should target
additional-compiler-flags	These are passed to the compiler to allow extra libraries to be linked in
path_rel_vividflow_lib_dir	The relative path where the VividFlow header is located
path_abs_vividflow_lib_dir	RUNTIME ONLY
vividflow_lib_header	INTERNAL USE ONLY – the name of the library header
vividflow_lib_template_source	INTERNAL USE ONLY – the name of the module template

Job Scheduler

check_frequency	How often the system should check for pending jobs. This value is in seconds
max_concurrent_jobs	How many jobs can run simultaneously

Jobs

out_log_filename	The filename that the log file should be written to.
path_rel_working	The relative path of working folders. A working folder is where a job will write intermediate files during execution of the algorithm
path_rel_output	The relative path of output folders. This is where final outputs are written
path_abs_output	RUNTIME ONLY
path_abs_working	RUNTIME ONLY

Resources

path_rel	The relative path where resource files are stored when uploaded
path_abs	RUNTIME ONLY

System

path_abs_root	RUNTIME ONLY
path_abs_static	RUNTIME ONLY

Project Closeout

Lessons Learned

The group has worked together and with the client for the past two sessions. In that time, we found some things worked very well, while others required change.

As we used a modified scrum for our project management we found that there were some initial issues applying it. The assessable items for the subject were added into our sprints during the first session, this didn't seem to agree with the sprint cycles. We found that the work wasn't done in time and often a lot of effort had to be put in at the last minute. In Spring, we changed this and removed the assessable items out of the sprint and worked on them as needed. Our results were better and we were happier with the work we produced.

The sprint item time estimates slowly became more accurate after each sprint as we got used to assigning time to a task. Tasks were often large and not broken down into smaller sub tasks which made estimating very difficult. In Spring, we spent time trying to break each task down to have a more detailed list of tasks. Our estimates greatly improved as well as our sprint burndown times.

Trello was initially used keep track of sprint items, however we found that there was too much overhead in managing the lists and producing the sprint burndowns. In the mid-year break we began using Google Spreadsheets to manage them and found this saved time.

In Autumn, we didn't work together unless we needed to. In the Spring, we started having working sessions after meetings. This increased our work output and helped solve issues with the develop quicker. We maintained this through the rest of the session

Post Project Review

After reviewing the functional requirements and use cases that were detailed during project planning, it's clear that this project has succeeded in meeting the client requirements.

VividFlow has been thoroughly tested on a variety of systems, and has an uptime of 3 weeks on a test server on an Amazon Web Services EC2 instance. A variety of modules with many different input files have been tested. There are no outstanding bugs that have been observed since development completed.

There were no significant issues within the team, or unexpected obstacles during development. Overall the project progressed extremely smoothly.

There is room for improvement in the system. Some potential areas to focus future development efforts are: further improvement to the user interface, improved modules allowing more complex module programs or support for additional programming languages.

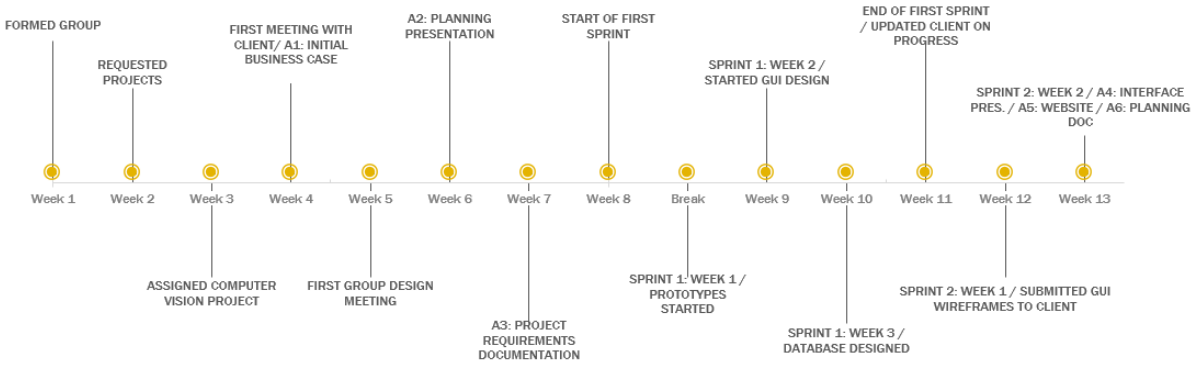
Additionally, the project was completed on time, and met all milestones during development. Overall our team considers the project highly successful.

Project Acceptance

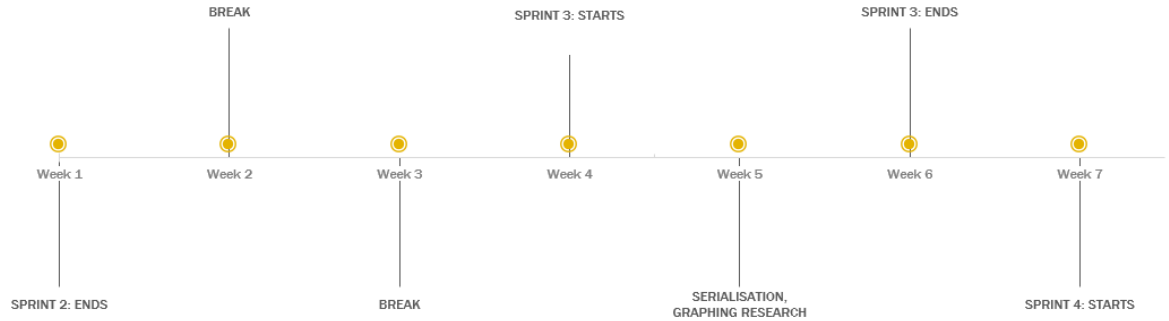
The group will now finish up with the project and hand it back to our client. This has involved packing and documenting all the work that has been completed. From there the client will use the system or have further development done to it.

Project Timeline

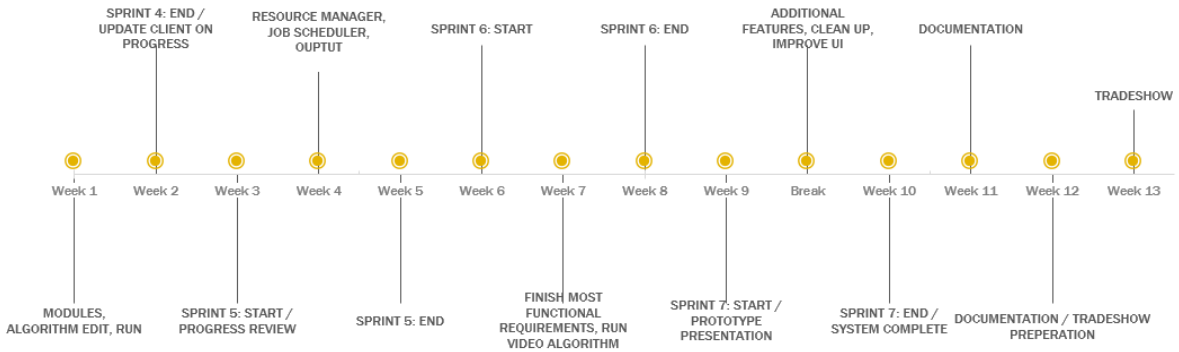
Autumn



Mid-year Break



Spring



References

Debian 2016, Debian — *Details of package gcc in jessie*, viewed 14 April 2016, <https://packages.debian.org/jessie/gcc>.

Debian 2016, Debian — *Details of package python in jessie*, viewed 10 April 2016, <https://packages.debian.org/jessie/python>.

Flask 2016, Design Decisions in Flask — Flask Documentation (0.10), viewed 10 April 2016, <http://flask.pocoo.org/docs/0.10/design/>.

Habib, O 2016, The Key Differences Between Python 2 and Python 3 - Application Performance Monitoring Blog | AppDynamics, viewed 10 April 2016, <https://blog.appdynamics.com/devops/the-key-differences-between-python-2-and-python-3/>.

Jinja 2016, Welcome | Jinja2 (The Python Template Engine), viewed 10 April 2016, <http://jinja.pocoo.org/>.

MySQL 2016, MySQL :: MySQL 5.5 Reference Manual :: 1 General Information, viewed 10 April 2016, <https://dev.mysql.com/doc/refman/5.5/en/introduction.html >.

OpenCV 2016, PLATFORMS | OpenCV, viewed 10 April 2016, <http://opencv.org/platforms.html>

Widdler, B 2016, Chrome vs Edge vs Firefox vs IE vs Safari vs Opera | Digital Trends, viewed 10 April 2016, <http://www.digitaltrends.com/computing/best-browser-internet-explorer-vs-chrome-vs-firefox-vs-safari-vs-edge/>