3DCytoFlow

A high-performance computing based cyber-infrastructure analyzes flow cytometry big data and creates three-dimensional interactive virtual reality for cancer pathologists

April 26 of 2016 - Sponsor: Dr. Sumit Jha

Group 4 – Manuel González, Ryan Gonyon, Luis Lavieri
1. Executive Summary:

We were assigned the task of designing, implementing, and testing a web interface that could help health professionals, specifically cancer pathologists, interact with three-dimensional models rendered from previously uploaded flow cytometry big-data to a cloud computing service. These models are used to help them analyze, diagnose, and evaluate certain cancers or the risk of recurrence in a specific patient. The heart of this project is an algorithm that translates the machine data into three-dimensional models; known as SANJAY.

SANJAY had already been developed by Dr. Jha and a team of graduate students. Remarkably, they have increased accuracy by a factor of 2.14\([1]\). That is twice as accurate as competing approaches. The idea was to bring this algorithm to the market to help professionals in different fields interact with elegant three-dimensional models rather than cumbersome data.

Designing a system like this was an exciting opportunity for us. With the fields of security and health in computing being so vast and having so many applications, having real-world experience in that field seems not only important, but extremely useful. This system had to be secure because of the private information that is to be processed and stored on the server. It was also exciting that we had a chance to work on a project that has such a huge impact on the lives of patients and of the doctors working with them to keep them healthy.

Technology is at its heart nothing more than a complex series of systems put in place to make our lives easier and better, and this project in particular is without a doubt a prime example of how indispensable the skills and projects we get to build and work on can be.

Besides making people’s lives better, we gained experience that will eventually help us in our careers. This project was composed of several specialized subsets in the CS field. Some examples of these subsets range from computer graphics to big-data analysis with some cyber-security and cloud computing as well. Also, it definitely helped us work as a team. A real-world task is accomplished by several team members with different backgrounds and specializations. Therefore, this is the perfect opportunity to design, implement, and test a real-world solution with more than one developer immersed in the project.

In addition, the challenging nature of it made us very excited. Taking a revolutionary tool and making it available to the people who need was not an easy task, especially with a problem involving highly complex mathematical operations which arise from the projection of higher dimensional data. That’s why we know that completing this project has realized us as fully capable computer scientists, as well as being recognized by most employers and even setting up the path for graduate level studies. The experience that we gathered and the thrill
of making such an impactful project reality will surely be invaluable for our future endeavors.

a. **Broader impacts:**

As stated before, flow cytometry is the technology that helps health professionals in making decisions towards whether a patient has cancer or has had cancer before. It is a state-of-the-art test that classifies certain cancers, such as leukemia and lymphoma. Additionally, flow cytometry might also be used to measure the amount of DNA in cancer cells. Therefore, the impact this project has towards innovation in medicine and oncology is immense.

With this technique we can theoretically answer questions such as, if a patient already had cancer before, what are the chances of recurrence? Or, if someone has it now, how much treatment will this person actually need? There is no doubt that the scope of this tool is beyond our own imagination. However, we needed to simplify the output of this machine. A flow cytometer can measure up to 50 properties of every cell, and a real sample can have 10 million cells. That translates to almost 2 GB of information per test per patient [1]. This exceeds the cognitive capacity of a pathologist or even a data scientist. This is where 3DCytoFlow will come in handy and can make an impact.

Having the ability to rapidly analyze millions of cells per second in an interactive three-dimensional model will allow pathologists to give quick answers. This kind of innovation will lead to prompt treatment and ultimately better outcomes. For instance, leukemia is a disease that if not promptly treated could get worse rapidly.

b. **Goals:**

Create a secure web interface that allows health professionals to interact with processed flow cytometry data by **SANJAY**.

c. **Sub goals:**

- Create a secure web portal that restricts people without permission, by requiring registration.
- Allow users to upload large data from .fcs files to a cloud computing service in a secure way.
- Enable the use of high-performance computing at the back-end of the website.
- Render interactive three-dimensional projections of the data that has been processed by **SANJAY** algorithm and save the results for future use.
- Make the website mobile friendly to allow end-users interact with results from handheld devices.
d. Requirements:

For completion of our senior design project, the following requirements had to be met:

● Have built a working professional-looking web portal for cancer pathologists.
● The system had to be able to keep track of a minimum of 20 doctors.
● Our system must be able to track every result for every doctor that uses it.
● The web portal must have an administration section for setting up doctor’s accounts.
● The system should be able to securely send the flow cytometry files from the doctors computer to our server, handling a minimum of 10 MB files, with the possibility of bigger files (anywhere from 100MB to 1 TB), depending on early completion of the initial requirements.
● Uploading files to our servers using the web portal should be easy and intuitive.
● Our system must compute a three-dimensional representations of the data set using the SANJAY algorithm and output them on the front end in a way that is both clear and informative for the doctors. No time constraints will be taken into account.
● Three-dimensional representations of the files must be interactive, being able to:
  o Zoom in and out
  o Rotate the camera around the center
  o Select sets of points and tag them
  o Remove some points from the view (such as points tagged by the user)
2. Technical Content

a. High level architecture

Our system is composed of five main components, as described in Figure 1. All of them are deployed on a single the cloud service environment Microsoft Azure and using mainly ASP.NET and C# and C++ inside the Virtual Machine.

1. Web Application:
   Our Microsoft ASP.NET MVC 6 application follows the Model View Controller (MVC) architecture, which is also based on the Web API design model.

2. Database Server:
   All of our abstract models in our Web Application are closely related to concrete entries in a database. MVC brought a set of tools that made this process very easy, the Entity Framework.

3. Sample Virtual Machine:
   This Virtual Machine emulates the behaviour of a real machine outside of our system and performs the computation necessary for running the SANJAY algorithm on an .fcs file to produce the three-dimensional representations of the dataset.

4. Web API:
   Microsoft ASP.NET MVC 6 allows Web API controllers along with MVC Controllers to be used in the same project, this made it significantly easier than before. However, we still treated them differently during the development process. The Web API handles calls from the Sample Virtual Machine, asynchronous calls from the results visualization, and calls from external applications.

5. Blob Storage:
   The Blob Storage ultimately contains all of the .fcs and results file used by our system as well as the Virtual Machine filesystem image. It is the Microsoft Azure version of a filesystem in the cloud.

The system will be used by mainly two types of users:

1. Administrators:
   We primarily designed this type of user to be the ones that give access to doctors restricting anyone that has not contacted by our team from registering. This increases security and prevents any huge spikes in user count during deployment and testing phase of our project, providing a controlled environment.
2. Doctors:
This type of user is the main focus of our project. Doctors are now able to access list of their patients, a list of those patients’ analyses, request an analysis and send the .fcs file to view the results. Note that although we are referring to these users as Doctors, they may not be necessary doctors in the real world, but for simplicity we are using this convention.

Both type of users require to be logged in every time they access the system. Figure 2 shows the overall Use Case Diagram of our project.
b. SANJAY algorithm

Our entire project is based on the work done by Faraz Hussain, Zubir Husein, Neslisah Torosdagli, Narsingh Deo, Sumanta Pattanaik, Chung-Che (Jeff) Chang and Sumit Kumar Jha. We have given the project a “web-enabled cyber infrastructure” [1] allowing public access to the algorithm that will ultimately lead to a user base that can provide feedback and suggestions to other research questions.

One of the analyses that can be carried from flow cytometry data requires an expert to look at a set with as many as millions of different cells with 10 or more different properties and then arrive to conclusions. It is clear that no human being can make sense of that amount of information without forgetting some of it in the process. Moreover, most of the information will be lost if it is presented to an expert in its raw form. Therefore, a method for presenting this information in a
different format is very important for increasing the accuracy of the diagnosis. This is the problem that SANJAY solves.

Several methods are currently being used for solving this problem. One of them is choosing a pair of experimentally important properties of the cell and plotting them in a Cartesian plane. The next step is looking for anomalies, and then look at another pair of properties and remember which of those points correspond to the ones before. Something like this becomes very tedious and basically makes the flow cytometry process useless. Another method and the one that the SANJAY algorithm is compared to is Multi-dimensional Scaling (MDS). MDS uses the distance from all points to all points and tries to maintain those relations when projecting into lower dimensions. The results provided by SANJAY are in average 2.41 times less distorted than the ones produced by MDS. Figure 3 taken from the research paper of Hussain et al. describes this situation.

In simple terms, SANJAY is a “new algorithm technique for automatically generating low-dimensional visualizations of high-dimensional flow cytometry data”[1]. For our specific implementation we are using the three-dimensional version of this algorithm. An .fcs file (Flow Cytometry Standard) [2], which will be explained in the next section, is formed by millions of rows with several columns of float values that describe a specific property of a particular cell in sample. An R script is used to parse the .fcs file, which will be explained in the next section, and extract these properties. With this table of values a graph is generated where edges between cell properties are created if the Manhattan distance between them is less than a certain threshold T.

The resulting graph is called a T-FCN (flow cytometry network). A theoretical value for T is computed that is guaranteed to generate the T-FCN that contains the most accurate description about the entire dataset that was originally in the .fcs files. Then the T-FCN is broken into several “communities”, and each point is projected into two or three dimensions using the distances between each point and a set of centroids corresponding to the community in which they are (3 for two dimensions, 4 for three dimensions). The resulting projection has an error (\(\epsilon\), \(\epsilon > 0\), also referred as distortion) derived from the fact that is impossible to not lose any information from the original dataset when projected into lower dimensions [1].

Most of the computations are using bit-vector decision procedures. For solving them, SANJAY is using Microsoft Research’s Z3. Z3 is a program that solves the SMT problem. SMT is a generalized version of the SAT problem. The SAT problem is the answer to whether or not a Boolean formula is satisfiable. A Boolean formula is a set of Boolean variables linked with disjunctions (ORs) or conjunctions (ANDs) whose final value is true or false. A Boolean formula is
satisfiable if there’s a set of values that can be assigned to each Boolean variable in it that make the final value true. This problem is proven to be NP-Complete [3], which is considered to be computationally complex. So, if you can describe a problem as a yes or no question, Z3 can solve it very fast (at least as fast as it can get).

Figure 3 Comparison of the results of 30 flow cytometry datasets and the resulting distortions using MDS and SANJAY algorithm [1]

Summarizing, SANJAY is generating answers by asking yes or no questions. So rather than saying “Give me the number that is the sum of 3 and 4”, a question, such as “Is there a number x that is the sum of 3 and 4?”, is asked instead. Of course, the answer to this question is definitely not a very informative answer, but for highly complex problems such as the projection of high-dimensional data points into low dimensions just finding whether there exists a solution to the problem is just as hard as finding the actual values that make it true. So, when
generating the projection the question is: “Given this T-FCN is there a set of three-dimensional points that represent it with at most this distortion?” If the answer is yes, the points are then computed.

There’s a theoretical minimal distortion in which no lower value of distortion can give a positive response. To find this value, SANJAY will iterate through several values of distortion refining the distortion value in each iteration. At every iteration, the best projection will be given. Note that the more we approach this value, the longer the calculation takes [1].

Our system is able to replace the old projection inside with the new one seamlessly, so users simply submit the .fcs file and get the projection, every time a significantly better projection is generated, the user should be notified (completing a sufficiently complex iteration may take days). However, after a certain number of iterations, the difference in the distortion becomes statistically insignificant, at this point the algorithm halts and the resulting projection becomes the final version. A description of the format of the fcs file will be explained in the next section.

c. fcs files

FCS stands for Flow Cytometry Standard. It provides the specifications to completely describe flow cytometry data sets contain the experimental data. This is the type of file users are going to upload to 3DCytoFlow and use the SANJAY algorithm to render 3-dimensional representations of the dataset.

This standard was developed and has been maintained by the International Society for Advancement of Cytometry (ISAC) since 1984[2]. Flow cytometry data is represented in the file in the form of a 2-dimensional array where the columns represent the parameters studied (fluorescence and scatter channels) while the rows represent the cells. The number of events for each sample usually ranges between the low thousands and the low millions. For instance, there are three scatter channels, 13 fluorescent channels for the first 30 (of hundreds of thousands) cells as shown in Figure 4.
Because .fcs files collect very specific data for each cell in a sample, they can get rather large. The regular size of an .fcs file can range from as low as 10 MB to 200 MB. However, 3DCytoFlow can handle file uploads up to 200 GB per file thanks to the power of Microsoft Azure and the Microsoft Stack.

d. The Microsoft Stack (.NET Framework)

Because some of our team members have worked with the Microsoft Stack before, we decided to use it in 3DCytoFlow. The latest version of the .NET framework is 4.5. It includes LINQ, ASP.NET, and Entity Framework (part of ADO.NET).

Figure 5 shows how all these technologies are related in the framework.

Some of the technologies that are part of the stack that we are using in 3DCytoFlow are:
1. **MVC ASP.NET**

It is Microsoft’s server-side web application for web development. We used the MVC design pattern, and it uses C# as the main server-side programming language. With ASP.NET, we produced the dynamic website needed by 3DCytoFlow. Some benefits of ASP.NET are:

- It is open-source, so not only is it free to use but is rich with features that other users find important and useful.

![Figure 5 The Microsoft Stack [4]](image-url)
• ASP.NET is built on top of the CLR (Common Language Runtime) which allows us to write ASP.NET code in any language that is supported by .NET.
• The framework is currently being implemented into a modular web framework which makes it even easier and faster to combine with other frameworks for compatibility and simple communication between frameworks is invaluable.

There are also many useful extensions to the ASP.NET framework that we found useful, for example ASP.NET AJAX is an extension that features client-side and server-side components, these components help the quick development of web pages with AJAX functionality, which we used for updating information in real-time. Another important extension is the ASP.NET MVC framework extension; this is exactly what was used for the majority of our design and implementation so it has been drastically useful for this project.

2. IIS 7

It is the web server used by Windows. It was used to test 3DCytoFlow locally. The web server is designed and distributed by Microsoft specifically for use with the Windows NT family. IIS is considered one of the most popular web servers to use, only behind Apache.

3. Entity Framework and LINQ

Entity Framework is a free open-source Object/Relational Mapping framework that exists to make life easier for software developers who are working with databases. It is specifically a framework that models applications based on entities and relationships.

The Entity Framework team describes it as “an object-relational mapper that enables .NET developers to work with relational data using domain-specific objects.” [5] In other words, it is used to automate writing the majority of data-accessing code that could be considered somewhat tedious to write. Instead of writing this monotonous code, queries are issued via LINQ (Language-Integrated Query).

Object/Relational Mapping or O/RM is an automated way of storing data from objects into a database using very little C# programming. O/RM is made up Domain class objects, relational database objects, and information on mapping domain objects to relational database objects such as tables, views, and stored procedures. This way of mapping data allows for the separation of database and domain class designs, making the application easier to maintain and scope up if it is ever necessary. Standard operations such as creating, reading, updating, deleting data, and tables are also automated using O/RM.

Figure 6 helps to visualize the architecture behind Entity Framework [6]
The Entity Data Model (EDM) has three parts: the conceptual model, which has model classes and relationships amongst them; the storage model, including tables and views as well as the relationships and keys that they share; and mapping, which is made up of information on the way the conceptual model maps to the storage model.

LINQ to Entities is a query-based language that can be used for writing queries against the object model, and returning entities that are defined via the conceptual model. Entity SQL is also a query-based language like LINQ to Entities. It is considered slightly harder to use than LINQ to Entities.

Object Service, this is one of the main entry points through which data can be pulled from and inserted back into a database. Entity Client Data Provider, this layer is primarily responsible for converting LINQ to Entities or Entity SQL queries into SQL queries that are understood by the database. The layer communicates with ADO.Net which is used to send data into the database, or pull data out. The ADO.Net Data Provider is used simply to communicate with the database via ADO.Net.

This entire framework was extremely useful for our database design, creation, and access. 3DCytoFlow's database was one of the first things that we designed, and Entity framework was perfect for this. It offered tools to design a schema via a visual designer application that could then be used to construct the database and the necessary classes. LINQ added native data querying capabilities to C#. So, we did not have to write pure SQL Queries for the project.
Microsoft Azure is Microsoft's own cloud computing platform and infrastructure. It was created for building, deploying, and managing applications and services (Paas). Because 3DCytoFlow is built almost entirely based on the Microsoft Technology Stack (C#, ASP.NET, etc.) that was mentioned before, we made the decision to use Azure as the host of the entire project. Besides providing the environment to easily publish and manage the application, Azure also hosts the main database that 3DCytoFlow uses, the blob storage used by the VM and the users to upload the flow cytometry data and store the results, and the Linux VM with the cloud service where the SANJAY algorithm runs. Figure 7 is a screenshot of what the main dashboard of Azure looks like.

Figure 7 Microsoft Azure Portal showing Web App Running

Azure also provides easy management of the users participating in the project and the version control system to manage the source code as well. We used Git as the main version control system for the source code since Azure supports it, and also because it is the one which we had more experience working with.

The first Azure subsection that we are going to talk about because it’s strictly related to where flow cytometry data will be collected is the Blob Storage.

1. Blob Storage

A blob (binary large object) storage is needed to collect the .fcs files, collect the results, and store all the information needed for the Linux VM. 3DCytoFlow could still become massively scalable. This means that it could eventually process hundreds of terabytes of data to support the big data scenario required by the flow cytometry samples. So, to support this scenario, we used Microsoft Azure Storage.
Blob storage specifications:

Storage Account

All access to Azure Storage is done through a storage account that is created in the main dashboard of Azure portal. The storage account provides a connection string and a password to programmatically connect to the storage using the proper library.

Maximum size of a blob

Because the size of an .fcs file is exponentially proportional to the number of parameters being analyzed in the sample, the files can get very large pretty quickly. However, the maximum size to upload a block blob to Azure Storage is slightly more than 195 GB [7]. This gave us enough range to experiment with different data set sizes in the project.

File system

The blob storage works as a very simple file system. Basically, every blob is organized into folders called containers. After the cloud service has processed an .fcs file with the algorithm, the system stores a result in .json format in the storage. The samples are never deleted after analyzing the data. However, a result will get replaced by a more recent version of it when a user requests a new analysis.

The SANJAY algorithm lets the users specify the amount of error than a sample can handle. More on this, the amount of error is inversely proportional to the time that the algorithm takes to analyze a sample. While the amount of error desired by a user is small, the algorithm will take a longer time to analyze a sample, and vice versa.

Blob naming standard

As mentioned before, the file structure of the blob storage is based on containers. However, containers are not able to store other containers. So, for the 3DCytoFlow specifics, we needed to establish the following container and blob naming standard:

- Every user has their own container and the name of the container is the last name followed by the first name of the user in the database. For example, smith-stan would be the container of a user called Stan Smith.
- An .fcs file of a user’s patient is stored in the user’s container. The blob is in another container named as the patient's last name followed by the patient’s first name and then the blob file is named as the the date that the sample was uploaded. For instance, smith-stan/doe-john/2014-05-01.fcs
- A result will be stored in .json format. And, the name of the file will be exactly as the .fcs file related to it, but with the correct extension.

A general storage scenario for the blobs that 3DCytoFlow collects is represented in Figure 8.

![Figure 8 Naming Conventions in Blob Storage](image)

**Figure 8 Naming Conventions in Blob Storage**

**Exchanging data with the storage**

To communicate with the storage, we have used a couple of technologies provided by the .NET Framework. The first one is the Microsoft.WindowsAzure.Storage namespace. This namespace provided the set of classes needed to send/receive data through the back-end of 3DCytoFlow. It uses the storage account credentials to effectively connect to the storage.

For the front-end, we used AJAX to asynchronously send chunks of data through the website. Now, because the SANJAY algorithm has been optimized to run in a Linux VM, we needed to be able to send/receive data between the storage and the VM. For this we decided to use Azure CLI.

Figure 9 displays the two different storages that 3DCytoFlow utilizes. One hosts the Linux VM and the other the .fcs files and the .json results.
2. **Azure CLI**

Azure CLI stands for Azure Command Line Interface. It is a set of open source, cross-platform commands for working with the Azure Platform. Basically, we were able to do the same functionality found in the Azure Management Portal, but using bash commands. Azure CLI is written in JavaScript, and required Node.js to be able to use it. Similar to the Microsoft.WindowsAzure.Storage namespace, Azure CLI uses the storage account credentials to be able to send/receive blobs from the storage.

The idea behind using Azure CLI is that when a user has finished uploading an .fcs file to the storage, the web application sends a request to the Linux VM to run the SANJAY algorithm. Within this request, the information needed to download the correspondent .fcs file from the Blob Storage will be embedded, and Azure CLI will be the interface that will download the file for the algorithm to run. Figure 10 displays how Azure CLI looks like when running.

3. **Virtual Machines**

We built a system that ultimately will be using other machines outside our servers to do the calculations, and so, we needed something that would emulate the behavior of a typical machine completely independent of our Web Application, also, we wanted to reduce the time and money costs of setting it up, so we decided to use a Virtual Machine to emulate this behavior without the need of setting up another completely different environment just for the development process.
The central component of a Virtual Machine is the hypervisor (or Virtual Machine manager). It’s capable of emulating any hardware components, such as CPUs, memory, Hard Drives, Graphics Cards, Network Cards, etc. This allows us to have multiple Virtual Machines that share the same underlying hardware, but are, in principle, completely isolated.

It was clear that the cost of maintaining one virtual machine is relatively the same as maintaining two, three, ten, a hundred; offering a highly scalable system that is extremely easy to maintain. Virtual Machines simplify “backups, disaster recovery, new deployments and basic system administration tasks”[8].

Even though using Virtual Machines offered a lot of convenience, it had its drawbacks. Virtual Machines usually need “higher bandwidth, storage and processing capacity than a traditional server or desktop” because they are essentially many machines into one. It does not mean, however, that the combined power of three machines should be packed in one virtual machine for them to run properly: one machine may be using 30 percent of the CPU while the other two are taking 20 and 50 percent respectively, later on, the usage of the first one may spike up to 70 percent, but the other two may have finished their task and go down to 15 and 15 respectively.

It’s ultimately up to the IT professional to control the balance of the system to ensure that every Virtual Machines runs smoothly. [8] Another risk arises from the fact that several machines may be using the same hardware. If such a hardware fails, all machines will fail, making it increasingly risky the more Virtual Machines get packed into a single physical hardware set. Again, the management of the system is very important here also: a backup of Virtual Machines should be made systematically, and any failures in the hardware should be promptly addressed to ensure end user satisfaction.
For all these reasons, Virtual Machines are the backbone of cloud computing services. Setting up a Virtual Machine can be automated, requiring no service from IT specialists at all. The customer can simply visit the website, ask for a Virtual Machine, and the system can regulate itself and allocate the necessary space for the new Virtual Machine to run. This fact is suitable for industrial scale.

Having thousands of different servers stacked inside a server farm, hosting millions and millions of Virtual Machines across the globe sounds like a dream, but it is in fact what most of the internet is becoming. This is what Microsoft Azure offers.

Every Web Application, SQL Database, Blob Storage and Cloud Service is a Virtual Machine itself. Of course, this fact is very much abstracted from us, the end users, but understanding how this works was essential for our team to be able to use Microsoft’s Azure servers, as our design had to take into account the asynchronous behavior of the platform.

Azure offers a wide variety of Virtual Machines with pre built Operating Systems and Environments already installed. It ranges from Ubuntu and Windows Servers, Oracle Databases and Barracuda Web Applications to Joomla! on a LAMP Server and WordPress on Ubuntu 14.04. There are over 700 different setups available for Virtual Machines, as shown in Figure 11.

When working closely with Dr. Jha and the students involved in the development of the SANJAY algorithm, they suggested that we use a Linux Virtual Machine, because most of their work was already set up and tested using this environment. Figure 12 displays the Linux VM already running in the Azure portal.
The machine holds the entire **SANJAY** code and a set of custom **3DCytoflow** packages that manage the inputs and outputs to and from the **3DCytoflow** Web API.

The setup of this Virtual Machine included installing third party tools that the **SANJAY** algorithm uses:

1. **Git**: Used for cloning the repository that contains **SANJAY** source code.
2. **Make**: Used for compiling the **SANJAY** source code.
3. **R statistics**: Used for parsing the data from the .fcs files.
4. **z3**: The SMT solver used in the core portion of the algorithm.
5. **GNU Parallel**: Used to parallelize the core portion of the algorithm
6. **Azure CLI**: Used to transmit the .fcs files and the result files to and from the Azure Blob storage.
7. **Curl**: Used to interface the VM with the Web API through HTTP requests.
8. **jq**: Used to parse/build json responses/requests from/to the Web API.
9. **Cron**: Used to schedule the scripts that check the Web API for new jobs or upload the estimated time for completion (ETC) to the Web API.

The working folder of the **3DCytoflow** Package will contain the following subfolders:

1. **downloadedFcs**: All the downloaded .fcs files downloaded from Azure will be stored here, ordered by patient name. The complete path for the .fcs
file related to an analysis submitted on April 18th 2016 at 12:33 pm will be
\texttt{downloadedFcs/patientLastName-patientFirstName/04-18-2016-12-33.fcs}

2. \textbf{results:} All the resulting three dimensional representations will be stored here, ordered by patient name. So, the complete path for the .json result file related to an analysis submitted on April 18th 2016 at 12:33 pm will be
\texttt{results/patientLastName-patientFirstName/04-18-2016-12-33.json}

3. \textbf{git:} Contains all the source code related to \texttt{SANJAY} algorithm

4. \textbf{scripts:} Contains all the bash scripts that request analyses, call the scripts from the \texttt{SANJAY} algorithms and uploads the results

Also, inside the working directory, a series of files are kept that are used by the scripts, namely: \texttt{analysisLock}, \texttt{fcsPath}, \texttt{jobsNumber}, \texttt{resultPath}, \texttt{containerName}, \texttt{ETC}, \texttt{pointsNumber}, \texttt{vmToken}. These files are used to send data from one script to another during execution. Figure 13 shows the structure of the working directory inside the Linux Virtual Machine.

![Figure 13 Main directory of the 3DCytoflow Packages inside the Linux VM.](image)

The scripts subfolder is the core component of the 3DCytoflow Packages. The following scripts are inside of it:

1. \texttt{request_download_fcs.sh}: Connects to the Web API using \texttt{curl} and parses the resulting .json object using \texttt{jq}. If the resulting .json indicates that an analysis is ready to be processed, it will download the corresponding .fcs file using \texttt{Azure CLI}

2. \texttt{json-generator.sh}: Generates a .json object from the resulting points in a format that the three.js code on the Web App front end can understand

3. \texttt{upload_json_file.sh}: Uploads the generated .json using \texttt{Azure CLI} and notifies the Web App that the analysis is ready using \texttt{jq} and \texttt{curl}

4. \texttt{script.sh}: Entry point for the Linux VM portion of the 3DCytoflow platform

Once all the third party tools are installed, the \texttt{SANJAY} source code gets cloned using \texttt{git} and built using \texttt{make}. Our 3DCytoFlow Packages scripts folder should also be cloned with git. Then we should schedule \texttt{./scripts/script.sh} as a background process using \texttt{cron}. Note that that if the 3DCytoflow working directory is not inside the home directory of the user who is issuing the scheduled job then “\texttt{cd \\{}(PathTo3DCytoflowDirectory)\\}\texttt{” should be called right before \texttt{./scripts/script.sh} gets called.

This script will do all the necessary work. First, it will verify that no other analysis is being processed right now. We use \texttt{analysisLock} file to check this: we will
only continue if `analysisLock` is empty. If we continue, we make sure the next scheduled job doesn’t try to start a job by writing “lock” inside `analysisLock`. Then `request_download_fcs.sh` will execute, if an analysis needs to be analyzed, `request_download_fcs.sh` will download the .fcs file to the downloadedFcs subfolder. If no analysis needs to be analyzed, `analysisLock` gets emptied and `script.sh` terminates. If an analysis does need to be analyzed, the SANJAY algorithms gets called using the `run_pfromd.sh` inside the `git` folder. Note that this function has the number of jobs to be used during the analysis and the number of points to be analyzed as arguments. Both of these are determined by the Web API and received by the VM during `request_download_fcs.sh` execution. Once `run_pfromd.sh` finishes, `script.sh` checks if there’s a valid result. If not, `analysisLock` gets emptied and `script.sh` terminates. On the other hand, if there’s a valid result, `json_generator.sh` is called to format the results into the .json object and `upload_json_file.sh` gets called right after to upload the result using Azure CLI and notify the Web API using curl. In addition, if the script detects that there’s an analysis being analyzed at that moment it will notify the Web API of the estimated time for completion (ETC) of the analysis. Figure 14 shows a state diagram describing the entire process.

SANJAY algorithm uses GNU Parallel to parallelize the generation of points. The ETC is calculated using the ETA of parallel.

4. Microsoft Azure SQL Database

The dynamic content of most websites is structured by the utility that databases offer. A database is a set of data held in computer memory that has some structure to it. The usefulness of databases comes from the ability to access and modify data stored inside of it conveniently and efficiently.

Web applications, more often than not, depend on the accessibility of this content to be fully functional. This portion of the report will explain the database of our application and the interactions with it. As well as all of the research that we did to investigate the different ways the database could have been structured and accessed, to assure we found the best option for our circumstances.
Figure 14. State diagram of the 3DCytoflow Packages scripts inside any VM. Note that running the SANJAY algorithm may run for multiple cron scheduled cycles. Update ETC refers to the ETC of the last SANJAY algorithm run that is still running.
MS Azure SQL Database for **3DCytoFlow** is the last main component that we used from the services provided by Azure. The cloud-based database allows users to make relational queries against stored data, which can be structured, or even unstructured documents. Some of its features are: querying data, search, data analysis and data synchronization.

f. WEB API

The **SANJAY** algorithm had to be able to run in any computer environment from anywhere in the world and still be able to easily integrate with the **3DCytoflow** Manager. To do that, our system accommodates to the needs of the service providers, they only need their environment of choice to have an internet connection and the **SANJAY** and **3DCytoflow** packages are running and pointing to the server where the **3DCytoflow** Manager is working. We met these requirements using a *Virtual Machine*. This allowed for a variety of systems with different setups to be providing, all at once, computing power for the three-dimensional projection portion of a request. For those reasons, we used Microsoft’s ASP.NET MVC 6 API Controllers to handle the interface between those machines and the **3DCytoflow** Manager.

Any registered Virtual Machine in the system can receive a request that a projection for a specific analysis is needed. The **3DCytoflow** packages inside them act as another Web API and return whether they will start working on it. If they do so it inquires the **3DCytoflow** Manager Web API for any information related to that analysis immediately after. Using that information, the Virtual Machine downloads the .fcs file using the given location through Microsoft Azure CLI. Once the download is completed, it runs the **SANJAY** algorithm until completion, uploads the results to the blob storage and sends the location of those results with a token that identifies that specific calculation to the corresponding **3DCytoflow** Manager Web API, and then waits for the next request. On the **3DCytoflow** Manager’s side, once it gets notified that a result has been computed successfully, it updates that location in the database only if the error corresponding to that specific calculation is better than the one that we already have or if it’s the first time it has been computed, else, it is deleted from the blob storage.

Additionally, for the purposes of allowing our work to be expanded on now that we are finished with this project, the **3DCytoflow** Manager’s Web API had to be able to provide a JSON file with the results to any external application that can provide doctor’s credentials. Although this is not technically part of our requirements. We felt it imperative to have an endpoint in our system that allows the project to be expanded on in the future as requirements evolve, for not only us, but any outside teams that wanted to build using our system.
In order to accomplish all of this, our Web API exposes:

1. Location of the .fcs file of a specific analysis:
   - Any registered Virtual Machine that has been assigned to an analysis can have only read access to the location of the .fcs associated with that analysis. A call that attempts to get this resource looks like this:

   GET  http://3dcytoflow.azurewebsites.net/File//RequestAnalysis?vmId=e3da5c67-b855-4d4a-aa76-e9c077761027

   Where “e3da5c67-b855-4d4a-aa76-e9c077761027” is the unique id of the virtual machine computing the process, and “RequestAnalysis” is the function in the API that handles the request.

2. Location of the result of an specific analysis:
   - Any registered Virtual Machine that has been assigned to an analysis has only write access to the location of the results associated with that analysis. A call that attempts to update this resource is written like this:

   GET  http://3dcytoflow.azurewebsites.net/File//AnalysisFinished?vmId=e3da5c67-b855-4d4a-aa76-e9c077761027&location=test-test/patient-test/04-06-2016-09-07.json

   Where vmId and location represent the Virtual Machine that processed the analysis and the location of the finished JSON file ready to be visualized. And “AnalysisFinished” is the function in the API to handle it.

3. JSON string describing the result of an specific analysis:
   - Any authenticated user can access any previously computed results through an external application. The result is JSON string with a well-defined format that describes the entire set of three-dimensional representation of the analysis. Only read access is given. A call that attempts to retrieve this resource looks like this:

   GET  http://3dcytoflow.azurewebsites.net/File//DownloadResult?analysisId=fc8987a

   Where “fc8987a” is the id of the analysis that we want to get the result of. The header of this HTTP request includes a token that uniquely identifies the user for a specific session.
4. Clusters Information for visualization:

- Whenever a doctor creates, updates or deletes a cluster, the host asynchronously sends the updated set of clusters to the Web API so it updates the database with the changes made.

g. Database Specifications

1. Relationships between Tables - Referential Integrity

Referential integrity is the set of rules that are used to constrain relationships between tables to ensure validity and avoid any accidental deletion or modification of relational data. For referential integrity to be set, some specific conditions must be fulfilled. The conditions are that the primary table’s primary key must be the matching column, or there must at least be a unique constraint on it, and that the columns that make up the relationship must be of the same data type and of the same size. Without these conditions, referential integrity would be impossible.

Some rules must be observed while referential integrity is being upheld. A value in the foreign-key column of a table that is related cannot be entered if it does not also exist in the primary key of the associated table. That being said, a null value can be used in the foreign key column. A row in the primary key table cannot be deleted if it also exists in an associated table. Additionally, a primary key that is in the primary key table cannot be modified if the row has other rows related to it. Upon creation of a relationship, the referential integrity is enforced as a default. This enforcement assures that any value in the column of a foreign key matches any existing value the in primary key column.

- Creating relationships between tables
- Checking existing data when creating relationships
- Showing relationship properties

One of the simplest examples of referential integrity is the use of a primary key, every table has a primary key that must be unique and every instance of some entity has their own distinct value stored for this primary key, only one user will ever have a specific UserID, no doctor can have the same one.

Another more complex example has to do with what happens when a patient is deleted from the system. When this happens, all analyses and .fcs files that are associated with that patient’s PatientID get deleted as well because there is no way to reference them anymore. This is handled using trigger constraints that will
take action when a Patient is deleted, and will run through the FCS_File / Analysis / Cluster tables and delete all associated content.

Figure 15 displays the table relationships in 3DCytoFlow.

4. **General Overview of the system**

Each user has a unique User ID number (or UID) that is the primary key for the User table, additionally, users will have a name, login username, password, and email address. Additional information about the user such as full name, and address are included but not all details of this are reflected in this diagram for simplicity sake. The User table is then split into two categories, Admins and Doctors. There is a separate table called UserRoles that has two rows, one labeled “Doctor” and one “Admin” these are used to see what type of user is currently logged in and display the appropriate menus and content based on that realization.

The UserID is a foreign key of the analysis, allowing us to see what doctor uploaded the FCS file for analysis and when they were working with what patient. The relation between the Doctor and the analysis and patient essentially exists so that each Analysis and Patient pair can be connected and stored appropriately.

An analysis is stored with an AID(analysis ID) primary key, the UserID associated with whichever doctor uploaded the .fcs file, a PID to track what patient the analysis was performed for, and the date that the analysis was done. Additional attributes include the location of the analysis in the blob storage and the location of the .fcs file that was uploaded and produced the analysis. Also, the delta value produced from the analysis, and the location of the .fcs file in BLOB storage.
A patient does not have login credentials or personal access to the site as there was no reason in our scope for this to be necessary. The doctor working with them can instead log in and show the patient what they need to see. The patient has a PID (patient ID) as the primary key, the patient’s first and last name as well as middle initial stored as strings, their e-mail address, date of birth, phone number / street address / city / zipcode are also stored in the Patient table, not all of these are reflected on the current ER diagram, once again, for simplicity's sake.

When a doctor is observing the analysis of a .fcs file they are also able to select clusters of points that represent irregularities or other possible interesting things
to check into. If they select a cluster, the cluster is stored in the Cluster table along with the information about which analysis it was produced inside of. The cluster has coordinates x, y, and z as well as height, width, depth, and color values all of which can be used to reproduce the clusters when the analysis is opened up again. These clusters can also be modified or deleted.

Figure 16 Entity Relationship Diagram
h. Web Application specifications

1. MVC

As mentioned before, we used the ASP.NET and the MVC design pattern to create the 3DCytoFlow portal. These two different definitions together with Entity Framework were used to communicate between the server, the client, and the database.

The MVC (Model-View-Controller) architectural pattern separates an application in three main components: the model, the view, and the controller. The model is the blueprint used to allow data to be pulled from and inserted into the database, the controller handles all the requests made by the user as the communication with the database, and the view is the client-side or what the user actually interacts with.

The relationship between the three components can be seen in Figure 17.

```
Figure 17 Model View Controller pattern diagram
```

The MVC pattern in ASP.NET fit our needs perfectly due to several advantages:

- It made it relatively easy to manage complexity by dividing the application. For instance, we have controllers that manage all of the Clusters and Analyses, a controller to manage the results, and another one for the patients. Of course, these controllers are attached to their correspondent user without sharing information between accounts.

- It provides better support for test-driven development (TDD).

- It works well for web applications that are supported by teams of different sizes like in our case. We were able to split the work of the three of us into different aspects of the application at the same time.
1. It does not use server-based forms. All the requests are done to the controller and the view only handles the response from the server.

Having explained 3DCytoFlow foundation, we are now going to talk about the graphics that it handles.

2. Graphics

Ultimately, by definition, the higher the number of dimensions in the representation, the closer it gets to the original dataset. Our eyes catch images in two dimensions, however, the human brain is specialized in making sense of three-dimensional spaces using two-dimensional images. Anything bigger than that, and we cannot make sense of the information as accurately. Therefore, the best representation of the dataset had to be a set of three-dimensional points.

However, another calculation is carried out to finish the visualization. Because our screens are a set of two-dimensional colored points (just like our eyes), we needed to define the color of each one of those points based on the three-dimensional points given by the SANJAY algorithm. This process is called renderization.

In computer graphics, renderization is the process producing an image based on a three-dimensional or two-dimensional model. A screen is basically formed by a set of contiguous discrete points that emit light of a specific color (pixels). An image is a set of values that define the color of each one of those pixels. In other words, renderization is the reverse process of what our brains do when processing the two-dimensional images from our eyes.

A three-dimensional engine is a program or application programming interface (API) optimized to render three-dimensional models into images. There are many three-dimensional engines out there, both proprietary and open source, some for real-time and pre-rendered images. For our project an interactive real-time render was necessary so the users could make sense of the data by moving the data points around and understand it. One of them was the OpenGL environment, which “has become the industry’s most widely used and supported two-dimensional and three-dimensional graphics application programming interface” [9]. For our project we used specifically WebGL, a javascript API developed by the Kronos group based on OpenGL ES (OpenGL for Embedded Systems).

Our User Interface allows the user to:

1. Rotate around the center of the entire dataset
2. Zoom-in and out from the center of the entire dataset
3. Zoom-in and out from the center of a set of selected points
4. Select a set of points and create “clusters” of points
5. Make a selection of points, not visible in the scene
6. Make a selection of clusters, not visible in the scene
7. Save/Delete the current configuration of clusters

All of these features are meant to optimize the user experience and are as simple and intuitive as we found possible. Our project was based on several projects that attempt to tackle cluster visualization. In one of them a box is used around the data points as a guide when selecting points and, around it, a set of buttons and textboxes allow for making each cluster visible/invisible [10], however, the user interface didn’t look professional enough. Another approach looked professional enough, but lacked the functionalities of the first [11]. Our approach attempts to give the professional looking of the second one while giving the wide range of functionalities described before.

i. Twilio

Twilio is the API which we utilized to be able to send doctors sms messages upon successful upload and processing completion of .fcs files. It was also used to email Administrators and inform them that a doctor has requested to be added into the system. Twilio is currently used by over 560,000 users in various contexts. Twilio bills our account based on usage of its services. Near the end of our project we ran into an issue where Twilio would only send texts to numbers that are local to the number we bought to use, this ended up being just an issue with using Twilio’s free trial instead of a full account, once we upgraded, the issue was solved.

3. UI Technologies

Some of the technologies that we used for designing the UI of the website are:

a. Bootstrap

It is a free and open-source collection of tools for creating the front-end of websites and web applications. It was originally developed by designers and developers from Twitter back in 2010 to encourage consistency across internal tools.

Bootstrap is one of the most popular frameworks for designing and implementing websites and web-applications. It provided immediate access to useful HTML, CSS, and JavaScript code to ease up the process of beginning our application.

Additionally, Bootstrap supplied countless tools for creating an aesthetically pleasing front-end to our application. Some of these tools are typography related, with headings/labels etc. being heavily customizable with very simple commands
Bootstrap’s website is filled with code examples that made our lives just that much easier during the implementation phase.

The CSS features of Bootstrap are built using Less, a Javascript framework which is a CSS-preprocessor (it adds a lot of useful functionality to CSS). Less is used to create CSS code that is easily maintained/theme/extended.

b. jQuery

It is a cross-platform JavaScript library designed to simplify client-side scripting. Combined with Bootstrap, both of them are some of the most-used front-end technologies used today. It was useful while developing the asynchronous task of transferring data between the client-side and the blob storage in the project.

jQuery can be used to wrap complicated JavaScript functions into simple events for easy manipulation of CSS/HTML, this greatly reduced the overall complexity of our project and made the entire client-side of the application process much speedier and simpler to implement.

Because jQuery is a JavaScript library, it is infinitely more useful if you are already familiar with JavaScript syntax and the various constructs. jQuery can be used for CSS, AJAX and other things we knew we would have to work with in the production of the application. Additionally, jQuery has been pre-packaged with Visual Studio ever since Visual Studio 2008.

c.metisMenu

A jQuery menu plugin to implement responsive menus in web applications. It is just a jQuery wrapper that helped us easily implement the responsive menu that 3DCytoFlow needed.

d. Three.js:

It is a cross-browser JavaScript library/API that abstracts out the details of the WebGL API to render computer graphics on the client-side of the web browser. We used it to render the three-dimensional representations of the sample results generated by the SANJAY algorithm.

Benefits

- As mentioned before, it hides all the details of three-dimensional rendering representing everything as scenes, meshes, lights, and materials.
- It is object-oriented. We used JavaScript objects to render the graphics in the browser.
• It is fast. It has been designed to use best practices without sacrificing usability.

• It has many features that will help enrich the user experience such as animations and other pre-built functions used in game development.

• It does the math. It has pre-built functions to calculate the 3D math, such as projections, matrices, and vectors. This was extremely useful for different things, specifically to keep track of the clusters after they have been selected using the x-y-z coordinates of the 3-dimensional model.

• It supports interaction. Something that was particularly important to our project due to the fact that we wanted to give users the ability to interact with the 3-dimensional models processed by the **SANJAY** algorithm.

• It has built-in file format support. Three.js is able to load files in text format such as .json. This was helpful for displaying our responses coming from the Blob Storage.

**Limitations**

As popular as WebGL has become, it is not supported everywhere. Internet Explorer does not support it, and a minimum graphic cards of 256MB is recommended. Users without graphic cards and those using IE won’t be able to enjoy **3DCytoFlow**.

e. **Asp.NET Scaffolding**

ASP.NET Scaffolding is a framework used for generating code. This was useful for quickly adding code to a project and having it interact with data models. [12] These tools were also used to generate code allowing users to modify data in the database tables through the front-end of the website.

f. **ASP.NET Identity Framework**

ASP.NET Identity is modular and has high-compatibility between every other ASP.NET framework including but not limited to ASP.NET MVC, web forms and web pages. It can be used regardless of whether the application being built is a web application, a mobile application, or a hybrid of the two like ours. Identity was also useful for creating our login system and the profiles associated with it.

Plugging in data to the table was extremely simple through the identity framework, the system could easily modified to store the specific information that we required to be stored for our application. Identity also features many “persistence control” features. For example, the database schema was controlled by us, the developers, and common but tedious tasks like renaming
tables or using different data types for primary keys were done very easily and reflected across all areas of the application without breaking anything or causing worry about content not being matched up to the appropriate tables or fields or not being stored / called correctly. This is leagues better than the alternative which can lead to deprecated code throughout the application that is just taking up space or worse, not functioning properly and being the root of countless errors that would have to be hunted down and dealt with one at a time rather than the snowball-solving effect that Identity provided.

We utilized the role provider tool that Identity features. This tool allowed us to have one User table but a separate table UserRoles (described more in detail in the database overview on Figure 16). An example of a user role is Administrator, every user in our application should be either be a doctor or an administrator and the content they are able to access as well as the pages they have access to after login depend entirely on which of these roles the user is assigned.

4. User Interface

To give a general idea of how 3DCytoFlow works. Figures 18 - 25 display the different views that 3DCytoFlow has.

- Invitation Request View

![Invitation Request View](image)

Figure 18 Register View

The first view in 3DCytoFlow is the Invitation Request view. From here, health professionals can introduce their credentials to be registered into the main site. An administrator of the website receives this information, and if all the information provided is valid, he sends a password to the user so he is able to log in with his new account in the Login View.
- Login View

![Login View](image)

Figure 19 Login View

As expected, the Login View takes the new password received by the newly registered user and uses its email address to login into the main site.
- Add Patient View

Figure 20 Add Patient View

As seen in the image, the website features a toolbar at the top and a menu attached to the left side. These two features are common to the other views providing the expected set of possible actions by a user. These actions are: Patients, Analysis Results, Upload Data, Add Patient, Edit Patient, User Settings, and Search Patient. The Add Patient option is specifically to create new patients for this particular user. Client-side validation will be used to validate that the data entered is valid.
Similar to Add Patient View, Edit Patient View is used to edit the information of an already existing patient. The toolbar and the left-side bar contain the same information as before as the client-side validation for the inputs as well. Deleting a patient also means deleting all the information attached to him (including .fcs samples and results). So, the user has to be really careful at the moment of making that decision.
• Patients View

Patients View displays the basic information of every patient related to a user. It also displays the number of samples related to a patient, and it will let the user delete a patient if needed.
The Upload Data View contains a combo-box/drop-down menu that lets the user choose the patient for that particular sample, a button to select an .fcs file from the local directory of the user and a submit button. Client-side validation is also used to avoid any error from the user. A progress bar updates with the percentage and the estimated time remaining for uploading a particular file. When a file is successfully uploaded to the blob storage, an SMS message will be sent to the user to notify them that the data was successfully uploaded and the analysis has started. Another SMS message is sent eventually to the user when the analysis is completed by the SANJAY algorithm.
The Results View can be accessed once a result has been processed for a particular patient. A combo-box/drop-down menu displays all the results for every patient, so the user does not have to go back to the table to search for a particular result. Three.js helps translate the .json response from the storage to render the 3-dimensional model. The user is able to select particular clusters from the 3-dimensional models, rotate, zoom-in, zoom-out, and save all of this information in the database.
j. **Sequence Diagrams**

1) Below in Figure 25 is our first sequence diagram, this one shows the process of an Administrator logging into the web service. When the Admin attempts to log in with their credentials, a query will be run, checking to see if the credentials are correct. If they are incorrect, the admin receives an error dialog that explains the username/password combination is incorrect, and they may make another attempt at logging in with correct credentials. If they successfully log in, then they will be directed to a View that lists all doctors in the Doctor table.

![Sequence Diagram: Administrator Log in](image)

**Figure 25 Sequence Diagram: Administrator Log in**
2) Below is Figure 26, the sequence diagram which describes the process of an Administrator adding a Doctor to the system. We assume that the Administrator has already successfully logged in and is viewing the View that lists all doctors in the Doctor table. From this list, the admin can select “Add Doctor” which will open up another view where the admin can enter the information associated with the Doctor. After they submit this information it will be processed into a query which will be run on the database, this query will add a row to the Doctor table that has the information the Admin entered about the doctor. After the doctor is successfully added, the admin will receive a Token that the doctor can use to register their account.

![Sequence Diagram: Administrator adds a doctor.](image)
3) The Figure 27 below shows the sequence of events that take place when an Administrator deletes a doctor. We assume the admin has already logged in and is viewing the list of all doctors. If they select a doctor for deletion, a query is sent to the database which deletes any doctor that has a DoctorID matching the one that was selected by the admin. If successful, the admin will see a “Doctor successfully deleted” message, and the list will be refreshed with an updated list of Doctors.

![Sequence Diagram](image)

Figure 27 Sequence Diagram: Administrator deletes a doctor
4) The diagram below, Figure 28, shows the process of a doctor registering for their account after being given a token by the administrator. If the token exists, then the Doctor's account will be successfully created. However, if the token does not exist in the database, then the doctor will receive a message instructing them to try again with a valid token, or contact an administrator to have an account created.

Figure 28 Sequence Diagram: Doctor Registers using Token
5) Figure 29 below shows the sequence diagram of a Doctor logging into the web service. They will enter their username and password into the appropriate fields and the credentials will be checked in the database. If they are correct, the doctor will be logged in and displayed a list of patients associated with them. If the login is unsuccessful, then the doctor will see a dialog that informs them the combination of username/password was incorrect and they will have the opportunity to try again with correct credentials.

Figure 29 Sequence Diagram: Doctor logs in
6) Figure 30 below shows the process of a doctor going into the system to update their information. From here we assume the Doctor is already logged in and viewing the page that displays a list of their patients. There will be a clickable link to the View that will allow them to edit their information if they wish to. This View will have fields necessary to modify the doctor’s data, and will run a query on the database that updates the information associated with their specific doctorID.

Figure 30 Sequence Diagram: Doctor updates his/her information
7) The Figure 31 below shows the process of a doctor adding a patient to the system. We assume the doctor is already logged in and viewing the list of patients they are associated with. From here the doctor can add a patient to that list, should they do so, they will be directed to the Add Patient View and there will be fields for the necessary information that a Patient needs to be created. And a query will be run on the database that adds the information provided into a row in the Patient table. The doctor will then be redirected to the list of patients, which will now be updated to include the most recently added patient.

Figure 31 Sequence Diagram: Doctor adds a patient
8) The following sequence diagram (Figure 32) describes the case where a doctor deletes a patient from a list of patients. We assume the doctor is already logged in and authenticated and that the doctor has clicked the “Patients” button from the navigation panel at the left of the webpage (Figure X.X). The doctor will initially be prompted with a list of all patients, he then must choose one of them and click the “Delete” button (Figure X.X). The doctor will be asked to confirm the action, and the patient will be deleted.

Figure 32 Sequence Diagram: Doctor deletes a patient
9) The following sequence diagram (Figure 3) describes the case where a doctor edits a patient information from a list of patients. We assume the doctor is already logged in and authenticated and that the doctor has clicked the “Patients” button from the navigation panel at the left of the webpage (Figure 2). The doctor will initially be prompted with a list of all patients, he then must choose one of them and click the “Edit” button at the upper left corner (Figure 2). The doctor will input the new information and click the “Submit” button (Figure 2).

Figure 33 Sequence Diagram: Doctor edits patient information
10) The following sequence diagram (Figure 34) describes the case where a doctor requests an analysis to be performed by uploading the .fcs file. We assume the doctor is already logged in and authenticated and that the doctor chose to view all the analyses for a specific patient. The doctor will initially be prompted with the requested list of analyses, he must then click the “Upload Data” button from the left navigation pane (Figure 23). Another view will be shown, the doctor will then choose the path to the .fcs file and click the “Submit” button (Figure 23).

![Sequence Diagram: Doctor requests analysis](image)

Figure 34 Sequence Diagram: Doctor requests analysis
The following sequence diagram (Figure 35) describes the case where a doctor visualizes the result of an analysis. We assume the doctor is already logged in and authenticated and that the doctor chose to view all the analyses for a specific patient. The doctor will initially be prompted with the requested list of analyses, he must then select one of the analyses with available results. Another view will be shown that provides the three-dimensional visualization of the selected result.

Figure 35 Sequence Diagram: Doctor visualizes result
The following sequence diagram (Figure 36) describes the process where the .fcs file gets transmitted to the Virtual Machine for processing and the results get transmitted back. We assume that a doctor requested an analysis as described in Figure 34. Once the .fcs file is completely uploaded into the Blob Storage, it will notify the system, and the Virtual Machine will start downloading it. Once this process is finished, the Virtual Machine will start computing the three-dimensional representation of the data, and, once that finishes, the results will be transmitted as a JSON string to the Web API, who will then store it in the Blob Storage and update the database with the changes.

---

**Figure 36 Sequence Diagram: Virtual Machine processes the .fcs file**
13) The following sequence diagram (Figure 37) describes the case where a doctor creates a cluster during the visualization of the result of an analysis. We assume the doctor is already logged in and authenticated and that the doctor chose to visualize the result of an analysis. The doctor will have to select a set of points using the provided tools as described in Figure 25. Then, the doctor will click the “New Cluster” button, select a name and a color for the cluster and click “Add”. The changes will be immediately available. Meanwhile, in the background, the current configuration of clusters for that specific result will be updated asynchronously.

Figure 37 Sequence Diagram: Doctor adds a cluster
14) The following sequence diagram (Figure 38) describes the case where a doctor deletes one of clusters during the visualization of the result of an analysis. We assume the doctor is already logged in and authenticated and that the doctor chose to visualize the result of an analysis. The doctor will have to select one of the clusters listed in the panel to the right as described in Figure 25. Then, the doctor will click the “Delete Cluster” button. The changes will be immediately available. Meanwhile, in the background, the current configuration of clusters for that specific result will be updated asynchronously.

Figure 38 Sequence Diagram: Doctor deletes a cluster
j. Testing

As mentioned before, to test 3DCytoFlow, we used Test Driven Development.

Test Driven Development (TDD)

TDD is a process that relies on the creation of automated test cases that define a certain behavior in the application. Microsoft's ASP.NET was designed to enable testability without requiring dependencies on the IIS server, on a database, or external classes.

The TDD lifecycle can be seen in figure 40.

![Test Driven Development](image)

Figure 39 Test Driven Development

Test Cases:

a. Admin Login Right Authentication

When the administrator successfully logs in, they needed to be greeted with a view displaying all doctors that are currently in the Doctor table, which contains all doctors that have been created.

b. Admin Login Wrong Authentication
If the admin attempts to login with authentication records that do not match anything in the database then they must receive an error message and remain on the current view, allowing them to correct their login credentials and try a second attempt.

![Bad Login Text Box](image)

Assume the Admin's username is admin123 and their password is MyPassword!23, this was tested with several variations of that password including but not limited to: mypassword!23, Mypassword123, MyPassword123, and MyPassword12. Any of these that were not exactly “MyPassword!23” produced the same error message that the username/password combination was invalid.

c. **Doctor Login Correct Authentication**

If a doctor successfully logs in, they are greeted with a view that displays the flow of the usage of our site, they have immediate access to a list of all patients they are associated with, from a database point of view, that is all patients who are found in the Patient table that have a DID (Doctor ID) which matches that of the doctor that just logged in.

d. **Doctor Login Wrong Authentication**

If a doctor attempts to login with invalid credentials the same error message described for invalid login from an admin is produced. This can be seen above in Figure 41. The doctor will then have the opportunity to attempt to login again with correct credentials.
e. **Doctor tries to access other Doctor’s patient**

The only way this seems like a possibility is if the doctor alters the URL they are currently using in order to attempt to access a different patient record. In the event of this situation, an HTTP protocol Error showing lack of permission, is called and displayed in its own view. What this error implies is that the server has received and correctly understood the request it is receiving, but it has refused to fulfill the request. The error message is printed out along with an explanation for the error.

![Error](image)

*Figure 41 Bad Request Text Box*

f. **Doctor tries to access other doctor’s analysis**

Similar to test case (e), this error should only be encountered if the doctor has attempted to modify the URL in an attempt to access content that they do not have the access to. In the event that this error is encountered in other circumstances, it is important to contact an administrator who can resolve the issue.

g. **Doctor uploads a correct .fcs file**

This was tested by uploading a .fcs file that has the correct .fcs extension. Contradictory to this test, is a test case that uploads a file that does not meet the requirement. Some examples of this included attempting a .fc file, .fs file, .pdf, .jpg, etc. When this is successfully done, the doctor should receive a message that their file was uploaded successfully and that they will receive notification when the file has been processed and an analysis has been produced.

h. **Doctor sees the result**

After a file has been successfully uploaded and sent to virtual machines for processing, the doctor who uploaded the file to the server should receive notification that their file has been successfully uploaded and then another notification when it is processed and
the analysis has been completed. From this point they can view the analysis, the data from which is pulled from the Analysis table in the database.

I. Doctor selects a set of points

When the doctor views the analysis, the three-dimensional model has the capability for clusters to be selected. If the doctor selects a cluster, the coordinates that make up the cluster are stored in the Cluster table of the database and associated with the Analysis ID corresponding to whichever analysis the cluster was created inside of.

m. Doctor creates multiple clusters

The doctor is able to create more than one cluster at a time and every cluster that is created is added to the database as they are created.

n. Doctor creates no cluster on a blank analysis

This does not modify the Cluster table in any way.

o. Doctor deletes a single cluster

Every analysis is stored in the database, along with the analysis is a set of clusters that the doctor has selected for examination. These clusters can be deleted, if the doctor deletes a single cluster, it is removed from the Clusters table in the data and update the UI accordingly.

p. Doctor deletes a set of clusters

If a doctor selects several clusters to be deleted from the analysis, this takes effect on every cluster that is selected for deletion and they will all be removed from the Cluster table at once.

q. Doctor deletes all clusters

This removes all clusters from the Cluster table that match the Analysis ID of the analysis that is being viewed by the doctor when they delete the clusters.
3. Administrative Content

a. Budget

The following table we display the initial budget to implement 3DCytoFlow and the actual final costs. The Oculus Rift Virtual Reality headset was an optional requirement that we included in the first budget.

<table>
<thead>
<tr>
<th>Name</th>
<th>Initial price</th>
<th>Final price</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cloud Services with MS Azure:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Includes 3 instances of a Cloud Services with the following specifications:</td>
<td>$110.40/mo</td>
<td>$55.20/mo</td>
</tr>
<tr>
<td>● 4 Cores</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● 7 GB RAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● 999 GB SSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SQL Database with MS Azure:</strong></td>
<td>$4.98/mo</td>
<td>$2.49/mo</td>
</tr>
<tr>
<td>Instance size:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● 5 Data throughput units</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● 2 GB Database</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>App Service with MS Azure:</strong></td>
<td>$55.80/mo</td>
<td>$27.80/mo</td>
</tr>
<tr>
<td>Instance size:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● 1 Core</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● 1.75 GB RAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● 10 GB Storage</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Visual Studio Online:</strong></td>
<td>$0.00/mo</td>
<td>$0.00/mo</td>
</tr>
<tr>
<td>Includes code repository</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Free for the first 5 users</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Jira and Jira Agile by Atlassian:</strong></td>
<td>$0.00/mo</td>
<td>$0.00/mo</td>
</tr>
<tr>
<td>Free 1 yr license for students</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Oculus Rift Development Kit 2</strong></td>
<td>$350.00</td>
<td>Not bought</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>$1034.72</td>
<td>$342.36</td>
</tr>
</tbody>
</table>
b. OUR Grant

In October, we applied for a grant called the OUR Grant, this grant is issued by the Office of Undergraduate Research (OUR). A single student applying for the grant can be awarded up to $500. The proposal was written to request the full possible grant amount for a group project, which was a $1,000 maximum prize.

On December 7th, we received an email with the exciting news that our proposal was received well and that we will be rewarded with the full $1,000 maximum to put to use in our project.

This was both very humbling and a fantastic beginning to an incredible opportunity. Although our team was already excited and passionate about this project due to its relevance to society and the future of our species, the decision to award us with this grant came with a spike in the motivation to make this project the absolute best that it could be.

At that time we were all looking forward to progressing on the full implementation of the project and completion that was done over this past semester. Not only will this project allow us to make a serious difference in the world, it is a cornerstone project that will never stop looking impressive on resumes and also make for an interesting and very cool story to share with anyone for years to come.

An additional benefit that this grant provided us is that our successful proposal proved to Professor Jha that we were serious about the project and passionate about it enough to make it an absolute success, which it now is.
c. Milestones

The following chart is a list of tasks that describe the entire Senior Design project during the entire two semesters of the course.

<table>
<thead>
<tr>
<th>No.</th>
<th>Milestone Description</th>
<th>S Month</th>
<th>E Month</th>
<th>Team Member</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>First Semester (Research and Design)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Initial Meetings</td>
<td>9/2015</td>
<td>9/2015</td>
<td>M, R, L</td>
<td>Finished</td>
</tr>
<tr>
<td>2</td>
<td>Gather requirements</td>
<td>9/2015</td>
<td>12/2015</td>
<td>M, R, L</td>
<td>Finished</td>
</tr>
<tr>
<td>3</td>
<td>Initial Research</td>
<td>10/2015</td>
<td>10/2015</td>
<td>M, R, L</td>
<td>Finished</td>
</tr>
<tr>
<td>4</td>
<td>Technology Research</td>
<td>10/2015</td>
<td>11/2015</td>
<td>M, R, L</td>
<td>Finished</td>
</tr>
<tr>
<td>5</td>
<td>Initial Design</td>
<td>10/2015</td>
<td>10/2015</td>
<td>M, R, L</td>
<td>Finished</td>
</tr>
<tr>
<td>6</td>
<td>Project Proposal</td>
<td>8/2015</td>
<td>10/2015</td>
<td>M, R, L</td>
<td>Finished</td>
</tr>
<tr>
<td>7</td>
<td>OUR Grant Application</td>
<td>10/2015</td>
<td>11/2015</td>
<td>M, R, L</td>
<td>Finished</td>
</tr>
<tr>
<td>9</td>
<td>Set up Azure Environment</td>
<td>11/2015</td>
<td>12/2015</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td>9.b</td>
<td>Create Blob Storage</td>
<td>10/2015</td>
<td>11/2015</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td>9.c</td>
<td>Test Blob Storage</td>
<td>10/2015</td>
<td>11/2015</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Upload</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>9.d</strong></td>
<td>Create a simple Database</td>
<td>10/2015</td>
<td>11/2015</td>
<td>R</td>
<td>Finished</td>
</tr>
<tr>
<td><strong>10</strong></td>
<td>Create a simple View within the Web App that allows to upload .fcs files to the Blob Storage</td>
<td>11/2015</td>
<td>12/2015</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td><strong>11</strong></td>
<td>Set up Sample Virtual Machine (Computing Server). Download SANJAY source code. SANJAY test run.</td>
<td>11/2015</td>
<td>12/2015</td>
<td>M</td>
<td>Finished</td>
</tr>
<tr>
<td><strong>12</strong></td>
<td>Senior Design I Final Documentation</td>
<td>11/2015</td>
<td>12/2015</td>
<td>M, R, L</td>
<td>Finished</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Second Semester</strong> (Project Implementation, Testing and Release)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>13</strong></td>
<td>Design and Implement the system that handles inputs and outputs on every Computing Server</td>
<td>12/2015</td>
<td>2/2016</td>
<td>M</td>
<td>Finished</td>
</tr>
<tr>
<td><strong>14</strong></td>
<td>Implement the Web API controllers that communicate with the Computing Servers</td>
<td>12/2015</td>
<td>2/2016</td>
<td>M</td>
<td>Finished</td>
</tr>
<tr>
<td><strong>15</strong></td>
<td>Implement Database Design</td>
<td>10/2015</td>
<td>12/2015</td>
<td>R</td>
<td>Finished</td>
</tr>
<tr>
<td></td>
<td>Task Description</td>
<td>Start Date</td>
<td>End Date</td>
<td>Rating</td>
<td>Status</td>
</tr>
<tr>
<td>---</td>
<td>----------------------------------------------------------------------------------</td>
<td>------------</td>
<td>----------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>16</td>
<td>Implement MVC controllers that handle Administrator actions</td>
<td>12/2015</td>
<td>1/2016</td>
<td>R</td>
<td>Finished</td>
</tr>
<tr>
<td>17</td>
<td>Implement MVC controllers that handle Doctor registration and Log in</td>
<td>12/2015</td>
<td>1/2016</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td>18</td>
<td>Implement MVC controllers that handle analysis requests from Doctors</td>
<td>12/2015</td>
<td>2/2016</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td>19</td>
<td>Design and implement Views related to MVC controllers that handle Administrator actions</td>
<td>12/2015</td>
<td>1/2016</td>
<td>R</td>
<td>Finished</td>
</tr>
<tr>
<td>20</td>
<td>Design and implement Views related to MVC controllers that handle Doctor registration and Log in</td>
<td>12/2015</td>
<td>1/2016</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td>21</td>
<td>Design and implement Views related to MVC controllers that handle analysis requests from Doctors</td>
<td>12/2015</td>
<td>1/2016</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td>22</td>
<td>Implement Web API controller that handles the asynchronous calls from the three.js result visualizer</td>
<td>1/2016</td>
<td>3/2016</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td>23</td>
<td>Implement the three.js result visualizer</td>
<td>1/2016</td>
<td>3/2016</td>
<td>M</td>
<td>Finished</td>
</tr>
<tr>
<td></td>
<td>BACKEND TESTING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>--------------------------------------------------------------------------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Test Web API controllers that communicate with the Computing Servers</td>
<td>1/2016</td>
<td>3/2016</td>
<td>M</td>
<td>Finished</td>
</tr>
<tr>
<td>25</td>
<td>Test MVC controllers that handle Administrator actions</td>
<td>1/2016</td>
<td>3/2016</td>
<td>R</td>
<td>Finished</td>
</tr>
<tr>
<td>26</td>
<td>Test MVC controllers that handle Doctor registration and Log in</td>
<td>1/2016</td>
<td>3/2016</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td>27</td>
<td>Test MVC controllers that handle analysis requests from Doctors</td>
<td>1/2016</td>
<td>3/2016</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td>28</td>
<td>Test Web API controller that handles the asynchronous calls from the three.js result visualizer</td>
<td>1/2016</td>
<td>3/2016</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td></td>
<td>UI TESTING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Test Views related to MVC controllers that handle Administrator actions</td>
<td>1/2016</td>
<td>3/2016</td>
<td>R</td>
<td>Finished</td>
</tr>
<tr>
<td>30</td>
<td>Test Views related to MVC controllers that handle Doctor registration and Log in</td>
<td>1/2016</td>
<td>3/2016</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td>31</td>
<td>Test Views related to MVC controllers that handle analysis requests</td>
<td>1/2016</td>
<td>3/2016</td>
<td>L</td>
<td>Finished</td>
</tr>
<tr>
<td></td>
<td>from Doctors</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>32</strong></td>
<td>Test the three.js result visualizer</td>
<td>1/2016</td>
<td>3/2016</td>
<td>M</td>
<td>Finished</td>
</tr>
<tr>
<td><strong>33</strong></td>
<td>Senior Design Final Presentation</td>
<td>3/2016</td>
<td>4/2016</td>
<td>M, R, L</td>
<td>Finished</td>
</tr>
</tbody>
</table>

**OPTIONAL REQUIREMENTS MILESTONES**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>34</strong></td>
<td>Build an Oculus Rift &quot;Hello world&quot; stand alone application.</td>
<td>12/2015</td>
<td>1/2015</td>
<td>M, R</td>
<td>Stretch goal, unfortunately skipped</td>
</tr>
<tr>
<td><strong>35</strong></td>
<td>Design and develop a simple prototype application that retrieves information from our servers and renders a sample analysis result</td>
<td>12/2015</td>
<td>1/2015</td>
<td>M, R</td>
<td>Skipped as well</td>
</tr>
</tbody>
</table>
4. Conclusions

In conclusion, we are happy to be at the end of our long journey through the creation and accomplishment of 3DCytoFlow. Even though that the overall technical challenge might have sounded overwhelming at first, we had the support of an incredibly talented group of professionals who perpetually provided the constructive feedback that we needed to design, implement, and test the platform. And besides all this, we also received the award of the OUR Grant that covered all the technical expenses that came up while implementing the platform.

In addition, we feel very proud that we were part of a project that might bring such an impact to the health industry. In the truly plausible event that our project eventually is able to assist medical professionals strengthening their diagnosis in a way that might save lives in the future, we would know that our efforts definitely paid off, and all the long hours, disputes, fights, and tears between the group members were not the least bit in vain. And, this has been the kind of experience that we, highly technical professionals that are soon entering into the industry, needed to ensure success in our futures.
5. References


