CRANFIELD UNIVERSITY

GONZALO PEREZ BADA

Digital Virtual Wind Tunnel Workflow

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Computational and Software Techniques in Engineering

MSc
Academic Year: 2015 - 2016

Supervisor: Karl Jenkins
July 2016

CRANFIELD UNIVERSITY

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Computational and Software Techniques in Engineering

MSc

Academic Year 2015 - 2016

GONZALO PEREZ BADA

Digital Virtual Wind Tunnel Workflow

Supervisor: Karl Jenkins
July 2016

This thesis is submitted in partial fulfilment of the requirements for the degree of Computational and Software Techniques in Engineering

ABSTRACT

Nowadays most motorsport, automotive and aerospace elements require testing in wind tunnel facilities for checking their aerodynamic behaviour. This implies a need of countless tests in this kind of facility when developing elements with either new materials or structural composition, and an important part of this elements are proven useless during this testing stage.

Wind tunnel tests, however, are not available in every academic facility, and are not costless, hence, developing a digital and intuitive procedure for performing initial tests on new designs would provide small universities and independent researchers a way to discard unsuccessful experiments before having to make an economical commitment.

Actually some tools such as OpenFOAM[1] provide researchers a way to apply different flow solvers digitally, without the need of an actual Wind Tunnel facility. This means that skilful researchers can program their own experiments and advance in their projects before performing the physical tests. However this kind of tools usually require expertise in computing and the OpenFOAM operating command, and the learning process might be complicated for some students.

Cranfield University decided to coordinate multiple students for developing a Virtual Digital Wind tunnel that uses the high performance computing facilities available in campus for generating a fast and powerful simulation of Wind Tunnel experiments. The main objective is to provide the user a web-based simulation of basic Wind tunnel experiments.

This project is focused on the automation of all the necessary steps to take in a wind tunnel experiment workflow, as well as the structure of the webpage backend server, with Database managing, file uploading and parameter entry interface.

Keywords:

Wind Tunnel, Grid Computing, Windows Workflow Foundation, ASP.NET Core, OpenFOAM

ACKNOWLEDGEMENTS

In this section I would like to acknowledge my gratitude to my supervisor, Dr Karl Jenkins, for providing me guidance even before picking a thesis subject, and being helpful in every situation and issue that appeared due to some conflicts with my professional career.

I would also like to thank my fellow researcher Aljaz Jelen, for all the support in mechanical engineering related fields, and also the willingness to coordinate properly both our thesis to implement the same technologies in order to make the final integration easier.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

CU	Cranfield University
WWF	Windows Workflow Foundation
ASP	Active Server Page
PHP	Hypertext PreProcessor
JS	JavaScript

1 INTRODUCTION

Aerodynamic research is taking great importance nowadays in the development of any type of vehicle. One of the main tools for checking a vehicle model is the wind tunnel, a tubular structure with powerful fans, which allows an object to be mounted in its middle. The aforementioned object is a Wind Tunnel Model, and is usually instrumented with sensors to measure the pressure distribution, aerodynamic forces and other related characteristics.

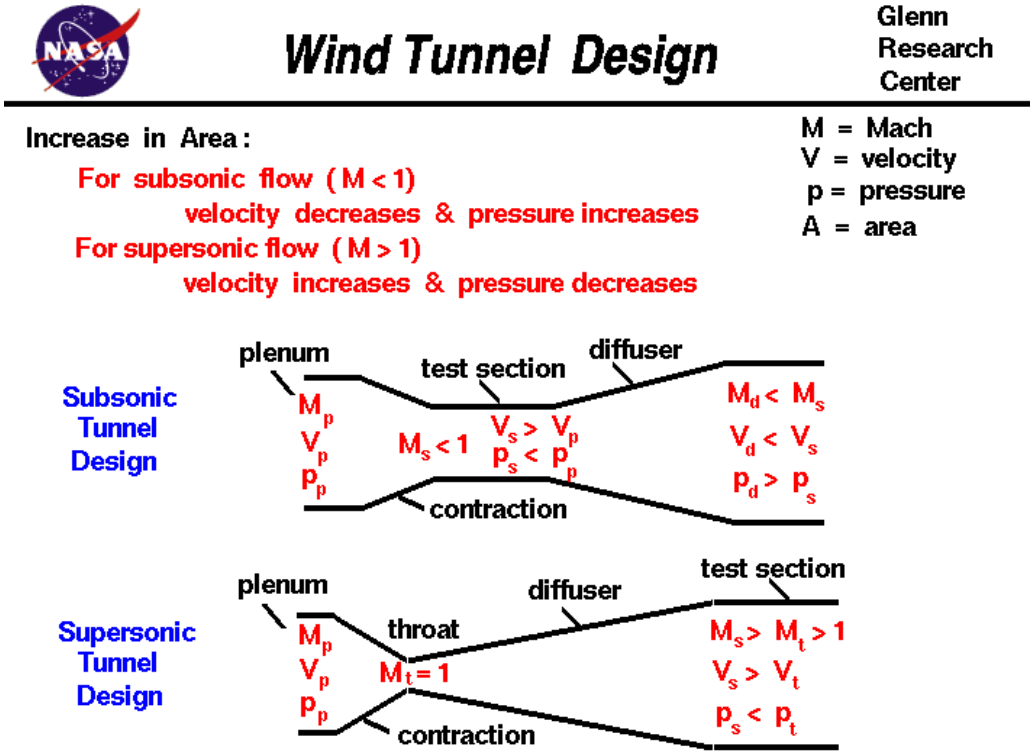


Figure 1 - Wind Tunnel Design

Wind tunnel design is done considering the specific experiments to be performed, and imply different areas depending on the flow nature to test: supersonic or subsonic. As explained in Figure 1 by NASA (1), there are multiple wind tunnel types, and each one requires certain design and instrumentation, depending mainly in the speed range to test, due to compressibility effects. This means a different distribution of the air density in the two main structures, implying a different speed gradient. Similarly, other parameters also affect the

wind speed, such as the cross sectional area and the test section area, both of which affect differently in subsonic and supersonic experiments.

All this considerations make the Wind Tunnel design not only complicated, but also expensive, and they imply limitations on the experiments based on the physical facilities.

But even in the case of having access to the perfect Wind Tunnel facilities for their experiments, researchers are forced to build expensive models that could end up harmed during the physical testing, generating additional costs for the investigation.

The aim of the project is to give researchers an alternate way to test their models virtually, based on an initial geometry, and being able to set up the simulation parameters beforehand, and just wait for the result.

For this purpose, an ASP.NET based webpage has been developed, which implements a windows workflow state machine, which sends the required OpenFOAM tasks to the server, monitors the progress and shows the user the results of each stage. This work will be also combined with a visualizing user interface developed by some of the fellow researchers, and also specific meshing and flow solving implementations researched by Cranfield University students.

The main alternative to Wind tunnel experiments for testing initial models are CFD software solutions, such as OpenFOAM, ANSYS Fluent, KRATOS Multiphysics and ABAQUS. These are really powerful tools that provide great detail on the experiment, but most of them are proprietary and require training on the platform for the researcher to get a solution to his problem.

There are also complete Wind Tunnel Simulators available Online such as Altair Hyperworks Virtual Wind Tunnel and FlowSimulator, but all of them have limitations, and the Altair Hyperworks one might not be affordable for non-industry related purposes.

This joint project consisted on developing a new take on Virtual Wind tunnels, by using Cranfield University's HPC facilities for high speed simulations, and also providing the user the outcome from each of the Workflow stages.

My thesis is focused on the workflow design of the process, as well as the main security and data management requirements for the webpage. In this report, the main implementation of this aspects is exposed, as well as alternative optios and recommended additions to the system.

The basic workflow for a Wind Tunnel experiment is composed by pre-processing (Meshing and Analysis Setup), flow solving process and post-processing (usually visualization and experiment reports). Nevertheless, in our project, some additional check stages where added for confirming the input parameters before actually submitting the experiment to the virtual wind tunnel.

This means that the workflow of the webpage has to be a bit different from the manual process for making automation possible, since the user should not simply restart the process in case of an input problem, but return to the creation state instead. This means the state machine for the workflow should be able to let the user change his input in case of failure in the early stages of the experiment.

This thesis is arranged in different parts:

Chapter 2 is the literature review, where several articles regarding scientific workflows, OpenFOAM and Wind Tunnel simulations are exposed and briefly explained.

Chapter 3 is the problem description, and tries to expose the aims of the project, as well as the main issues I faced during the development of the solution.

Chapter 4 is the implementation, the main section of the report, where I explain the steps taken for solving the problem and also the technologies used. Here multiple aspects of the project are highlighted and described in detail, such as the webpage layout, the database structure and the workflow state machine.

Chapter 5, Testing and Results, exposes the main procedures followed for testing the system and obtaining the final results. And it will also contain interpretation of the aforementioned results.

Chapter 6, the Conclusion, will consist of a review of the whole thesis, a summary of the work performed, and also includes a brief discussion on possible improvements and future work.

2 LITERATURE REVIEW

2.1 Introduction

This section aims to define the multiple approaches available for solving the main problems faced in my thesis. Firstly, the project requires to stablish the method for performing the actual Wind Tunnel simulation, and there are multiple options available for defining this workflow. Then, the web server has to be properly arranged, and there are also different technologies and versions available. Finally, there are also web security and data encryption considerations to take into account.

For each of the aspects previously mentioned, this literature review will try to describe how several actors approach the solution. The main differences and points in common will be highlighted for a better understanding of each problem and potential solutions.

This section is composed by 3 parts:

1) **Wind Tunnel Workflow:** Focus on the scientific approach to wind tunnel experiment simulations and Grid computing as a solution to the problem.
2) **OpenFOAM developing:** Main design guidelines for flow solve experiments using the OpenFOAM toolkit.
3) **Web design and security:** Design patters and technologies available for designing the web server, and al security considerations: Encryption, authentication and authorization.

2.2 Wind Tunnel Workflow

2.2.1 Workflow Technologies

The workflow word can be used to describe different things depending on the context, however, according to Y. Zhao [en la página 61], scientific workflows are usually applied to four aspects of computations: description of complex procedures, data derivation automation, high performance computing and provenance management. The author highlights the trend towards grid

computing systems as a scientific computation infrastructure, as opposed to supercomputer, which have fallen out of flavour. It also exposes the main strengths of the available workflow related technologies, such as WWF (workflow integration in an application), Swift (bridge between workflows and parallel computing) or Pegasus workflow system (long job scheduling). The author also states that the technologies available are sufficient for workflow deployment, but parallelism can be optimized for maximizing performance.

In [61], A. Pavethan highlights the need of a really automatic process for Wind Tunnel Experiments, since at the day, the coordination between software, hardware and data acquisition was not perfect, and most Wind Tunnel experiments required manual data movement operations. It also states the main states required for an automated solution: 1) Data verification 2) Experiment metadata annotation 3) Raw data movement 4) Processing defined by user. Pavethan also supports his statements by a real Laser Doppler Anemometry experiment, explaining each step in detail. The same paper also states the basics needed for developing a scientific workflow using WWF and Globus Middleware.

Also A.Pavethan, in [61], explains every step necessary to develop a Commodity Grid toolkit with the help of Globus Middleware. This is a toolkit to exploit the computing capabilities of a Grid Computing system for scientific experiment. The need for developing a new toolkit is due to every alternative being language-specific, and the MyCoG.NET toolkit developed runs under Microsoft .NET framework, providing the necessary API functions for grid management in FORTRAN, C++, C# and Java. In the paper, the author highlights the main considerations when developing a grid computing system, such as Security (Authentication, Authorization and Delegation), job submission scheme and queries, file transfer features and performance. It also supports .NET framework as a perfect framework for achieving good results in the aforementioned aspects. The paper backs up the toolkit with a case study of a Wind Tunnel application workflow, using Southampton University's Wind Tunnel facilities, using WWF as well as .NET for the workflow implementation of the experiment. The whole workflow architecture is defined and each activity is

composed by Experiment specific activities, MyCoG activities and WWF activities; all of the architecture running under Workflow runtime and .NET Common Language Runtime (CLR).

2.2.2 OpenFOAM principles

Thanks to Computational Fluid Dynamics (CFD), it is possible to perform a detailed numerical study on three dimensional flows, as well as an analysis of the numerical results. As stated in [61] by A. Gartmann, CFD has been used not only as a previous security check for scientific flow experiments, but also as a compliment for post-processing, since the measurement techniques have their own limitations. In this paper, a study in CFD modelling for characterizing a portable wind and rainfall tunnel is performed, describing numerical solving schemes, simulating and validating the wind speed patterns, and finding potential issues that might appear in this wind tunnel. The whole modelling is performed using OpenFoam 1.7.0 with the simpleFoam solver, and it is a great example on wind tunnel modelling.

OpenFOAM is proven a really powerful tool for scientific simulations, and this is not restrained to Wind Tunnel simulation, as S. Pashami highlights in [61], OpenFOAM based CFD is a really powerful modelling tool for Gas behaviour. This paper is focused mainly in compressible flows, but the design principles are similar, and with the proper theory basics adaptation, can be used as a base for non-compressible flow simulations too. The author clearly states the three main stages of any OpenFOAM design: geometry and environment definition as pre-processing, actual flow solving and finally post processing and result visualization. He also presents a list of multiple OpenFOAM solvers recommended for this specific task, highlighting their main characteristics, and also stating that all of them require the same initial condition set.

2.2.3 Web Design and Security

Web Design is one of the most critical of Software Development nowadays. Whether for designing web applications or just setting up APIs or web services, there are loads of different technologies and design patterns available to every

developer. Our approach is related to authentication, database entries and also file hosting, and also requires setting up a computation Server. This means that we will have to consider different frameworks for both Front end and Backend.

First off, for the Database possibilities, there are two main branches: Relational Databases (SQL) and Non-relational Databases (NoSQL). This distinction has been discussed several times since the early 80s, and as stated by E.F. Codd [7], we can call relational to Databases with the ability to process he date in a relational way. Meaning that these databases are the ones with capabilities to access the data by value, not only by ID, and then retrieve the values of the same able entry. This meant a huge performance increase for data management in computation, and made it easy for programmers to develop data management algorithms. The main advantages of the relational model are the simple structure of the Data Model are using data dependence as a boundary between logical an physical aspects of a database, simple model structure and providing high level language statements for retrieving big chunks of data. The most usual relational database technology is MySQL, and there are multiple possibilities for setup and deployment, but the choice is usually dependant on the technologies used for accessing them. A bit more recently, a new approach on databases appeared, and it quickly got loads of supporters. Michael Stonebreaker [8] distinguished two different types of NoSQL databases: the document databases, containing a key, a value and a payload, designed for storing documents, being MongoDB and CouchDB the most relevant examples. And the key-value stores, usually distributed hash tables that only include pairs of keys and values, being Dynamo the most used example. This approach's main advantages are performance and flexibility, since they allow the developer to deal with huge chunks of data, and perform map and reduce operations way faster than relational databases. They also allow the database schema to be more flexible, which is really useful when the data does not follow a rigid relational design. Nowadays this technologies are widely used in Big Data approaches and some databases with less focus on querying. However, it is complicated to deal with ownership in this kind of databases, and for some problems with intensive data querying and relational searches, NoSQL options are not really a good choice. Therefore, depending on

the approach of our project, we should consider both relational and non-relational databases and study which advantages could we exploit to further extent.

The database usually acts as link between the frontend and backend of a web application, and these two sides of the web also provide several valid options. Since the birth of Javascript [9], the way to design webs has changed wildly. No more static webpages with loads of links to more static content. Every webpage is developed taking dynamic interaction into account, and all of them adapt their content to its users' needs. In the mentioned book, the basics of Javascript are explained, and this helps to the development of dynamic content. However, this is no longer enough to keep web design up to date, and loads of frontend development frameworks and libraries have been created in the last decade. Javascript can be complicated while trying to find specific elements in the web document, and this was the main reason for the creation of jQuery [10] which provides querying tools for multiple element modification at the same time. jQuery is a javascript library that exploits the $ symbol for quering the HTML DOM, easy to learn and quite powerful, although it has some limitations over the complete Javascript language. AngularJS [11] is one of the newest frameworks available for web design, created in 2009 by M. Hevery and A. Abrons with the purpose of extending HTML in a declarative way. The result is a powerful language that allows easy creation of user interfaces. This language has been widely used by both big and small companies, and has been empowered by companies such as Google for their client side web design. There are plenty of other options for this Front end design, but not nearly as popular as AngularJS. Aside of the actual logic design, the styling is an important part of every webpage, and the base of this styling is Cascade Styling Sheets [12]. In the reference the basic concepts of cascade styling are defined. Originally, HTML styling was described in each component, and although that´s still done in certain cases, CSS changed the whole design concept. By using id and class tags, we can define the elements' style in separated files, and then apply it to the HTML DOM elements simply by adding this class or id tags to them.

But this design concept has been pushed even further lately, and new frameworks have emerged, due mostly to the need for consistent and responsive styling in webpages. Most big companies have decided to make their own styling libraries, being the two most popular Twitter Bootstrap and Google Material-ui. Both frameworks define the styling in such a way that the developer can easily setup a webpage that will adapt to the user's screen size and remain stable. However, the Google framework requires the use of Facebook React framework for modular design, and might be a bit more complicated to setup than Bootstrap. Twitter's Bootstrap [13] is simply the easiest to setup and use, and has become the de-facto standard for web styling. It allows the creation of navigation bars, dropdown menus, tabbed contend, element toggling, and much more; and does not rely on third party libraries to work.

The final part of the webpage is the backend, where the whole server-side logic should be held. There are also plenty of options for the server side, but I mainly considered three different options: NodeJS, Ruby on Rails and ASP.Net.

NodeJS is the most used out of the three options, and it provides almost limitless possibilities due to the plethora of available libraries to pair it with. Additionally, by pairing it with npm, a developer can easily extend the program using thousands of Javascript libraries. As stated in [10], node has become really powerful as a server-side development tool, and supports long-running server side programs based on events, as opposed to other programming environments, much more focused on multi-threading; but this does not imply a performance decrease, and it makes it way easier to design the application flow, since there is no need to define bit by bit the timing for each running task.

Ruby on Rails [14] is also an interesting alternative for the server-side design, and it provides a great advantage: an already implemented frontend architecture. This framework follows the Model-View-Controller system, which allows to easily follow separation of concerns and get modularity on a webpage. This frameworks provides a really easy way to get a webpage running, and supports prototype oriented design. This means that a starting draft of the web can be shippable really quickly, and all the development can be oriented for adding functional

increments to this initial prototype. This development schema has been widely used in software engineering, and supports agile methodologies, hence, Ruby on Rails shall be considered as a valid option. It has an embedded SQLite database, and can be easily connected to any kind of relational Database Server, by altering the drivers. The system is based on migrations, which can be generated on the command line, and this allows any change in the model to be reflected in the database, and also recovered by rolling back to the previous migration. Controllers hold the logic of the application, and retrieve data from the models to generate an updated View to the user, who can also call back the controller's functions by interacting with the view.

The final possibility for the backend is the Microsoft option, ASP.NET [15]. It stands for Active Server Pages, and is the main proprietary technology for server web design. It is also based on the .NET framework, and usually makes use of Visual Studio Tools for development. The .NET framework has two basic component, the Common Language Runtime (CLR) and the Framework Class Library (FCL). The CLR defines the execution environment, manages the .NET code execution, memory allocation and garbage collection. It runs for every programming language available in .NET (C# and Visual Basic, usually) and translates it into native code for the processor, so there's no need to set up a different runtime environment for each. The FCL defines a series of classes that implement the whole functionality of the ASP.NET system, and also classes that allow the developer to add functionalities to the system. The FCL does not only support web development, but also provides the same functionalities for console application design.

Hipertext PreProcessor or PHP [16] is another server-side language worth mentioning, since it had been widely used as the main language for server side design in the early 2000s, but has no longer such a big influence, and most developer lean towards other possibilities. This is mainly due to PHP not having a defined architectural structure. This means that it allows innovation in several ways, but is also complicated to approach a problem without a time consuming architectural design phase before even starting to solve it.

3 PROBLEM DESCRIPTION

The initial objective of this thesis is to coordinate all the necessary function calls for a Wind Tunnel Workflow, hence being able to run a Wind Tunnel experiment simulation based on web services and concurrent function calls.

A wind tunnel is the basic tool for any kind of aerodynamic research nowadays, about the physical experiments are quite expensive, and not every researcher can afford to run all the experiments there, especially in the early stages of the research, where discarding a design before submitting it to the wind tunnel can generate considerable savings in both time and money.

Digital solutions are already available, as stated in previous chapters, but these tools usually have drawbacks in the sense that they need extra investments even for basic usage, either be them monetary inversions or just time investments for learning how to use the technologies.

Hence, this thesis main goal is to provide researches a free to use and easy to configure tool, so that they can submit their experiment geometries to a webpage, setup the experiment parameters using a web form, and let the webpage go through the states of the simulation. He shall also be able to download any of the experiment related files to his computer, and save multiple experiments with different parameters in his own experiment database.

This goal is not only meant for my thesis, but to the whole subset of Virtual Digital Wind Tunnel thesis in which some of my colleagues have been working, and my main goal was to define an optimised workflow and coordinate each one of the stages of the process. This chapter is meant to briefly explain the stages involved in the Wind Tunnel process and also showcase the main difficulties I had to face during my work on the project, in order to properly comprehend the scope of my work and the whole virtual-digital wind tunnel project.

3.1 Wind Tunnel Stages

The physical wind tunnel is a duct mounted with a viewing port for being able to identify the geometry in detail. Air flow was originally generated by a series

of fans in one of the duct's ends, and the geometry of the element is mounted inside with sensors for being able to model the flow data across the object. For coping with the effects of physical viscosity, the cross section of the wind tunnel is usually circular rather than square, and this also helps a smoother air flow than a the square option. This last fact is also critical, since the turbulent flow is not useful for the experiments, and this also sets some limitations on the materials used for building the whole system.

As for the digital equivalent of the Wind Tunnel, the whole process is a bit different, but the main problem remains the same: applying a steady wind flow to an object, and measuring the flow field, pressure and other parameters, but this times those values are numerical rather than experimental. As previously stated, the main advantage of this option is the possibility to run the experiments in early development stages, and without requiring to build a physical geometry model of the object, or use wind tunnel specific facilities.

For being able to perform the digital simulation, the stage division is defined as pre-processing, flow solving and post-processing.

3.1.1 Pre-processing

This initial part of the whole process has to deal with the experiment setup. The mesh defines the environment in which the experiment will be ran, and an incorrect configuration would mean getting completely useless results, hence, this is a critical stage in every experiment.

This stage is tightly related to the experiment mesh, which defines the boundaries in which the flow solving simulation will take place. The geometry definition is usually composed by points in 3D space, and the format is pretty specific to the application used to deal with the experiment.

The mesh can be generated by a different tool than the solver, but it requires conversion to an understandable format, and it also usually has to pass a series of format checks before being accepted by the flow solving tool.

Pre-processing also includes additional parameter settings, such as the air flow velocity or experiment temperature, since these factors d not only affect the final results of the numerical analysis, but also might imply requiring a completely different numerical resolution method for obtaining an accurate result.

Boundary definition is also critical, and not only in the physical sense, but the mathematical implications of those. Boundary and inter-boundary conditions are an integral part of any computational method for solving a differential equation. For being able to apply this conditions to the different parts of the geometry, these conditions are usually divided, and the solving tool will treat them differently.

3.1.2 Flow solving

The critical part of the system is actually simulating the experiment, and this means solving multiple differential equations across the element's geometry. This is usually the most resource-intensive task of the whole process, and some experiments might require complex problem structure and long simulation periods.

Most of the flow solving tools have parallel computing capabilities, and having access to a super computer makes the simulations times much shorter. However, big experiments such as airplane or motorsport designs still require countless simulation hours, even when having access to supercomputing facilities.

The solving can usually be also configured for a specific area of the object, and this can also affect the way computations are performed. There are multiple solvers available, and each specific experiment might require using a different one, and hence configuring an optimal wind tunnel simulation is impossible without a previous setup for a specific experiment. Therefore, each experiment kind shall be predefined, and its parameters should be limited before setting up the whole experiment work flow, even applying combined limits on different input parameters that the user could introduce.

3.1.3 Post-processing

The final stage of the whole Wind Tunnel flow is the post-processing, where the output data from the flow solving is managed and treated in order for being understandable to the final user. This stage is often tightly bound to visualization, but can also be done in a more statistical sense, such as highlighting relevant simulation parameters that evolved across the whole simulation time.

There are multiple visualization tools, and once again the file format compatibility can be an issue between different tools at the different stages.

These tools usually offer a graphical user interface, and are not easy to embed in a web service structure, and due to time restrictions and complexity issues, this part of the project was set as out of scope when he project was under development.

However, once the result data is generated, being able to post-process it outside the web application is relatively simple, so the implementation inside the web application is not really critical for the Wind Tunnel Project right now.

3.2 Difficulties and issues

The initial problem definition and scope for the problem were constantly modified during the thesis, mainly due to the workflow being dependant on all the other stages and the specific wind tunnel experiment for the workflow implementation. This meant that a flexibility on scope was required in every stage, and due to the other related thesis being constantly evolving due to their own requirements and issues, in the end; the specific implementation for discarding them was discarded, and I had to focus on a more general implementation.

Some of the initial problems I had to deal with were technical debt issues and library dependencies being broken. Most of my initial research was focused on the Globus and Windows Workflow Foundation interaction for wind tunnel simulations on high performance computing approaches. There were plenty of papers on the subject, and although none of them got into implementation details,

I was able to subtract loads of ideas from them. However, I had to leave this approach aside, due to the Cranfield HPC facilities being updated and not using Globus infrastructure anymore. This meant that most of the advantages of using Windows Workflow Foundation were no longer relevant, and after doing some research on the latest webpage design technologies, I decided to park that idea too, since not even Microsoft's ASP.NET was supporting the WWF framework in the latest releases.

Instead I opted for an SSH connection approach on the problem, which would mean a way more general option for calling external functions, and could also enforce separation of concerns between the webpage design and the actual experiment solving, which was not part of my project's scope in the first place.

The most critical problem I had to face was the license expiry for Visual Studio Enterprise, my basic developing tool for the whole process. The downgrade to the Community Edition had quite a bad interaction with my project's inner dependencies, and the webpage code was rendered useless. I had to completely wipe out the program from the machine and reinstall the community edition, and then restart the webpage from scratch and port al the code sequentially. This mas a major problem for the thesis, and some of the planned features had to be cut out of the scope during the last weeks of the thesis.

4 IMPLEMENTATION

4.1 Selected Technologies

A great part of my work on the thesis has been related to the selection of the tools and technologies I was going to use for each of the components of the system, and this section is focused on justifying most of the elections and showcasing the process followed for taking this decisions.

As previously stated, my project is focused on the workflow, hence, the stage definition and coordination across the entire Wind tunnel simulation process. However, I cannot decide the way to implement each one of the stages, since those are carried by other colleagues from Cranfield University.

The initial idea was to coordinate all the students under Karl Jenkins supervision, and focus the workflow on coordinating those different functions that dealt with the Wind Tunnel simulation stages.

Unfortunately, this approach was discarded early on the project, since the timing for all the thesis was the same, and it was not possible to deal with the different technologies and scope changes of each thesis. Hence, I decided to restructure the thesis in a more general manner. And set a template for delivering wind tunnel experiments based on web services, which could be customized once each stage was properly defined. This will prepare the project for further evolution, and make it possible to coordinate the finished optimisations of pre-processing, flow solving, and potentially post-processing.

4.1.1 Wind Tunnel simulation

The technology selected for the actual wind tunnel simulation was OpenFoam, and as previously stated, is an Open Source tool with a wide user community. This means that there is a huge support on the Internet, and also plenty of tutorials and online resources. However, it is still quite a complicated technology, and I decided not to spend too much time researching the optimal implementation of the wind tunnel, and instead of that just focus on the basics of the program and design a way to customize an easy experiment through the web application.

OpenFoam is a really powerful tool, but it is not based on a Graphical User Interface, as the proprietary alternatives, but in a command line approach, that deals with several files at the same time. This files are quite complicated to properly deal with separately, and creating an experiment from scratch is really difficult, even for experienced aerodynamic experts.

For dealing with this issue, the OpenFoam releases a series of example files within each new release, and the quickest way to get an experiment to work is just to pick one of the releases and modify the inner file structure for meeting the needed criteria. This is indeed complicated, mainly because of having to deal with multiple values inside multiple files and folders, and that is almost impossible

to deal with for a non-expert user. The main goal of this whole project was to simplify this wind tunnel simulation task, and then the main goal as far as this technology is implied, is to create a way to customize an experiment according to the parameters introduced by the user through the web application; and for that purpose, it is imperative to comprehend the starting state of an experiment, and also the main differences between stages and the way to trigger them. However, this task changes depending on the experiment, and therefore I will explain it in one of the further sections of the chapter.

4.1.2 Web Design

As far as the web design goes, I already showcased loads of different options across the available technologies nowadays. Since I already had some knowledge on Javascript web design, this was initially my preferred options, but after an initial spike on the popular frameworks and libraries, I notices my backend knowledge was lacking, and this would require me to invest an outstanding amount of time trying to learn NodeJS on my own just for setting up a simple database. This did not force me to discard the technology, but it did indeed lead me to research into the rest of technologies with an open mind.

This led me to focus on ASP.NET, the Microsoft proprietary language, which provides great compatibility for Windows users, who are the vast majority of the computer users. So I did research on the latest available frameworks by Microsoft, and the latest release offered great capabilities for web design and additional features. ASP.NET Core unified all the previous webpage design templates into a single one: the ASP.NET Core Web Application. This made the architecture of the web really easy, but it also had three important new features. First off, the whole core had been redesigned for being able to run in any machine that holds the server, and then there is no longer a need to rely on Windows architectures for the server side. This might not sound like an important advantage, but might be really useful when the Wind Tunnel is released, since both the webpage server and OpenFoam could be run in the same machine, and therefore the server could potentially run the computer command line for running the simulation and avoid unnecessary inter-server communication. The second

relevant feature is the Model-View-Controller structure for the webpage. Previously, the framework offered this as an option, but it separated different templates for different webpage purposes, such as Windows Forms applications, web applications, static webpages… This new architecture layout means that a single starting prototype can evolve into whatever we need it to. The last new feature is the integration of NuGet, Bower and Gulp inside the framework, which makes library additions and debugging so much easier than before. This will also make extensions of the system easy to implement, and the library search can be dealt with within Visual Studio. Additionally, the MVC Structure made it easy to implement both the front and backend at the same time. All these reasons made me choose ASP.NET Core MVC as the web design technology

4.1.3 Database

In the literature review, I also stated different possibilities for holding the experiment data, and I picked a relational database as the technology to use for several reasons.

First and foremost, the Microsoft ASP.NET Core provides SQL database functionalities embedded into the MVC framework. This means that it is almost automatic to adapt the database to the desired data models, and this makes the database management and maintenance really easy. The MVC framework can also be reconfigured for using different database connections, but if possible, I would have liked to avoid this problem, and using SQL aligned with DNX core seemed like a perfect fit for my thesis goals.

The second main reason was the structure I decided to give to the database. A user based approach where each individual user had his own experiment history seemed perfect for our web based approach on the wind tunnel. This implied an ownership relationship between the user model and the experiment model, hence not being the optimal option for non-relational database performance.

The volume of experiment data was also worth considering, since non-relational databases reach their maximum performance on Big Data situations.

We can define Big Data as any quantity of data over one Terabyte, and it is usually linked to map and reduce kind of operations rather than regular queries. By map and reduce operations, we refer to a certain data process operation dealt on a subset of the whole data collection, or an operation that is incrementally applied across the whole data domain. This is not really the case for the Digital Wind Tunnel, where each experiments data is only used to run the simulations, and then is stored and queried, but not really modified. Taking all of this into account, most of non-relational databases advantages were nullified for the project, so sticking to the easiest to implement relational database was considered the best possible option.

4.2 System Architecture

After introducing my technology stack, the next logical step is to expose the structure of my solution to the problem stated in the previous chapter. I already talked about the need to redefine the project's scope to a more realistic target, mainly due to lacking the interface to the stages of the process.

This fact forced me to think of a way to model the wind tunnel experiment on my own, and it also had to be easy to adapt for adjusting to the rest of the functions once my colleagues finished their own thesis. This was quite a challenging task, but not having any previous knowledge on wind tunnel physics or technologies made it really complicated for me.

Therefore, I decided to stick to the simplest possible wind tunnel experiment and think of a solution that could connect with external servers in an easy way. Therefore, I decided to setup my own external server for the computations, and try to find a way to connect the webpage with the server. The final architecture is showcased in the following image:

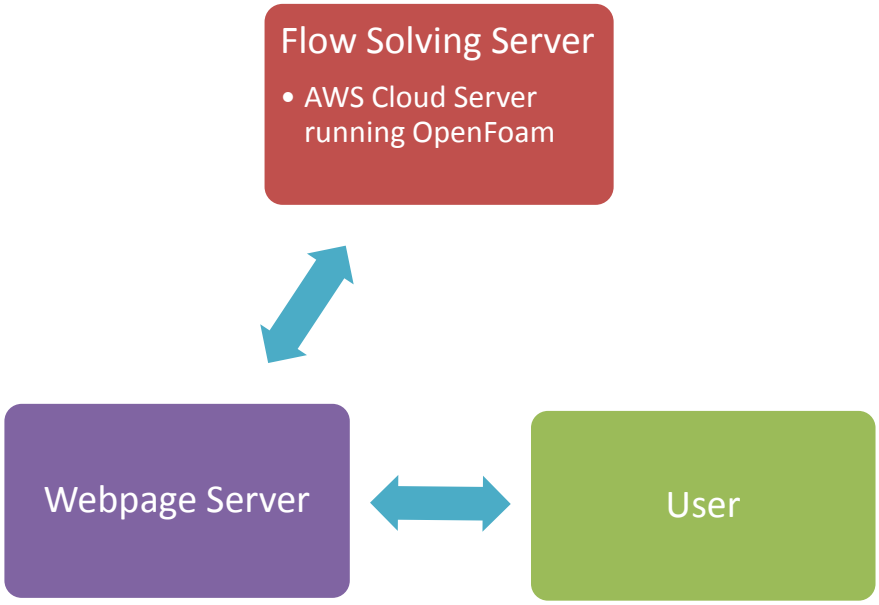


Figure 2 – System architecture

The three main components of the system are the two servers and the User. The webpage server hosts most of the project, and serves as a link between the user and the actual solver for his Wind Tunnel experiment.

The user interacts with the webpage using his web browser, and can easily setup the experiment parameters, and upload his own files to the webpage. This means that he no longer cares about the way to run the simulations, and can just wait for his experiment to be handled by the web server.

In the webpage, the user will be able to generate new experiments, move their experiments to the next stages, or delete the experiments. In the experiments tab, a history of previously generated experiments will be showed to him, and he will be able to view its details and download each of the experiments files, for both the setup and results.

Then the webpage will generate an ssh connection and an scp connection with the AWS server, and copy al the needed files to run the simulation to a new folder in the flow solve server. Then, in each one of the following stages, the webpage wlll use the ssh and scp connections to run the OpenFoam commands and take the files back to the webpage, so that the user can download all the results.

This architectural design was designed for maximizing compatibility and expansion capabilities. SSH and SCP are really powerful functions that allow for secure connections with Ubuntu servers, and this allows to easily adapt the system to interact with new implementations of the flow solving system. Modifying the ssh calls is as easy as reconfiguring the commands used in the ssh call. Hence, getting the system to work would mean that we could reconfigure it for implementing the subsequent calls to the optimised functions once my colleagues' thesis were also completed.

Therefore, the architecture was designed in such a way that completing the project would guarantee a starting point for the whole Wind Tunnel Process, which could be further developed by future students and generate a complete solution to the problem without having to redefine the whole workflow for each new improvement.

4.3 Flow Solving Server

As just stated, setting up a server to deal with the computations mean that the whole solution would be easy to adapt to different implementations of the functions, since the connection would only need a reconfiguration to target a different server.

The server had to be able to run the OpenFoam simulations, but it would also be interesting to set up ParaView in the server, for being able to process the results and customize visualization in further stages of the Virtual Digital Wind Tunnel.

It was also in the projects best interest to set up the server in most widely used setup possible. And after some research, I decided to set the server using Amazon Web Services. Amazon provides cloud computing capabilities to loads of huge companies and projects worldwide, and allows for easy setup of Unix and Windows servers; while also providing a wide variety of pre-loaded server images.

In fact, the designers of OpenFoam provide huge amounts of content and tutorials on setting up and using their tool, and after researching through the

portal, I found an already preset AMI for a server running both OpenFoam and Paraview at [17].

The server was setup by launching a new EC2 Instance from the AWS Console, and I was configured for using the minimal resources, in order to minimize server costs. A micro instance of the server was launched and could easily be accessed using the .pem keyp provided for ssh connection with AWS.

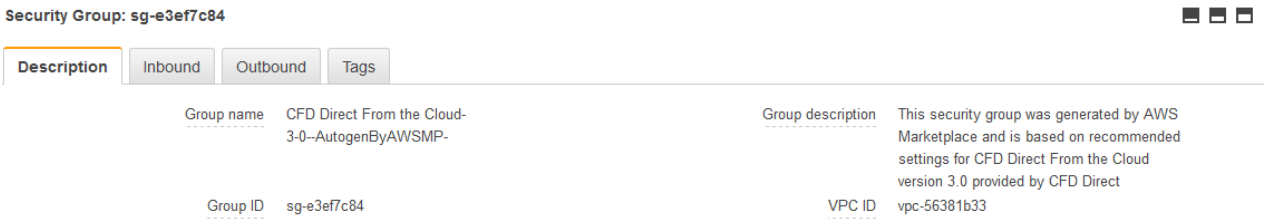


Figure 3 - CFD Direct from the Cloud Instance

In Figure 3 we can see the description of the instance in the AWS EC2 Console, and in the same AWS Console we can get the servers IP address, and set up the security restrictions. AWS is a really complete service that allows for great customisation, and we could even setup different user roles, with various permissions, and hence we could configure these permissions to adapt to our connection's needs.

However, for now I decided not to focus on the server configuration, since the ultimate goal of the collective Digital Wind Tunnel is to build a joint solution, and these server might have to be set aside once the rest of the stages have been properly set up in a single Flow Solving server.

For the initial testing stages of the process, MobaXTerm [18] was the selected tool for running the AWS Server. This is a tool for remote computing in Windows, which allows Windows user to run bash command line interfacing. This allowed for easy testing in the same machine I was running Visual Studio, without having to restart the computer and run the Linux partition.

4.4 File Structure

For supporting the experiments from multiple users, I also had to think of a way to save all the files from the experiments, and I decided to arrange the structure in such a way that each user would have his personal folder inside the Files folder of the web server, and then inside that folder, each experiment will have its own subfolder to hold all the OpenFoam files, as we can see in Figure 4.

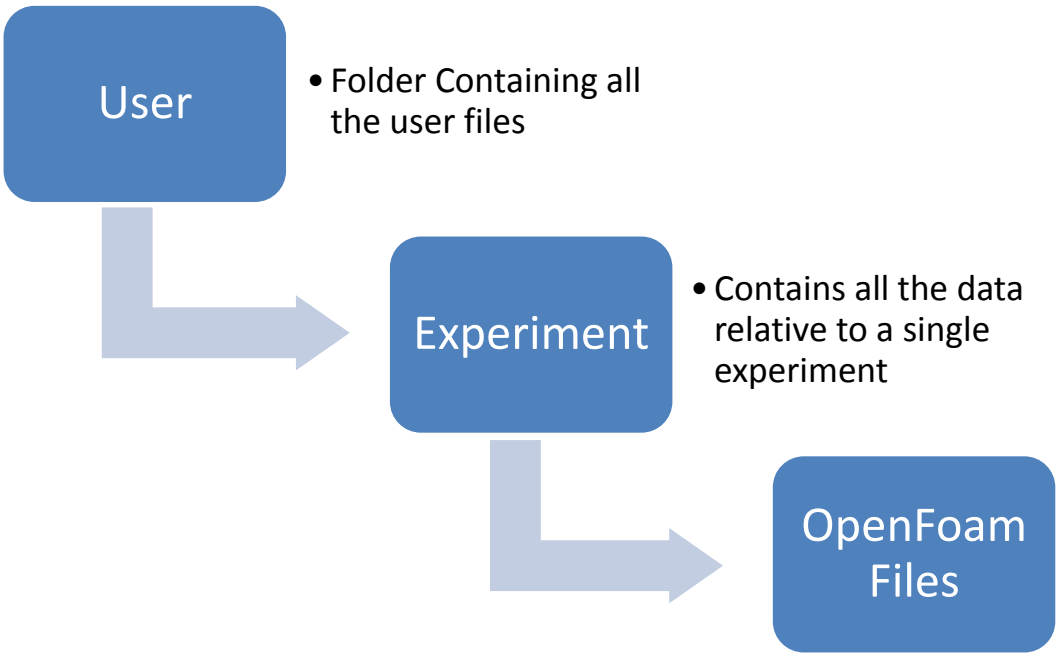


Figure 4 - File System Structure

This file structure means that there will be no filename duplication system between the users, and as far as the experiments are not named the same for a single user, creating new experiments or modifying old ones will never affect the rest of the experiment files. This means that users can play with the parameters of experiments without having to worry about their already finished ones.

This structure also allows for easy tracking of each experiment's files, and thus the webpage can easily create, remove or modify files inside each experiments folder with simple logic.

4.5 Webpage Structure

The main workload of the thesis was designing the whole webpage for serving as a link between the user and the Wind Tunnel experiment resolution, and as stated in previous chapters, the development tool of my choice was ASP.NET Core MVC. For being able to develop in the ASP.NET environment and take maximum profit out of the MVC Web application template, I used Visual Studio 2015. Initially, I used the Enterprise version of the software, but during the last stages of the project, the Cranfield license for the software expired and I had to downgrade to the community version. This meant some losses on the testing department, but the main issue was the effect the downgrade had on the code.

Apparently, some of the installed dependencies were configured for the Enterprise version, and while moving back to the Community version, these dependencies remained incompatible, and fixing them was impossible. After some trial and error, repairing the thesis project resulted almost impossible, so for being able to advance with the project, I had to start a new project and code the whole solution again. This ended up being the main difficulty of the whole project, but I was able to cope with it relatively easy and keep working on the project.

Figure 5 - Model View Controller schema

I decided to use the MVC structure for the Web application, which stands for the Model-View-Controller design pattern that can be seen in the figure 4. This design facilitates encapsulation and separation of concerns.

4.5.1 Models

The Model is holds the database structure, and defines how the information on the data entities will be structured. For this solution, I defined two basic models, the user model and the experiment model. The user model is an extension of the ASP.NET Identity system, designed for holding user account information in a secure way. This system was redefined for holding some university related information, and using a username rather than an email for logging in.

```
CREATE TABLE [dbo].[AspNetUsers] (
    [Id]                   NVARCHAR (450)     NOT NULL,
    [AccessFailedCount]    INT                NOT NULL,
    [ConcurrencyStamp]     NVARCHAR (MAX)     NULL,
    [Email]                NVARCHAR (256)     NULL,
    [EmailConfirmed]       BIT                NOT NULL,
    [LockoutEnabled]       BIT                NOT NULL,
    [LockoutEnd]           DATETIMEOFFSET (7) NULL,
    [NormalizedEmail]      NVARCHAR (256)     NULL,
    [NormalizedUserName]   NVARCHAR (256)     NULL,
    [PasswordHash]         NVARCHAR (MAX)     NULL,
    [PhoneNumber]          NVARCHAR (MAX)     NULL,
    [PhoneNumberConfirmed] BIT                NOT NULL,
    [SecurityStamp]        NVARCHAR (MAX)     NULL,
    [TwoFactorEnabled]     BIT                NOT NULL,
    [UserName]             NVARCHAR (256)     NULL,
    [Compname]             NVARCHAR (MAX)     NULL,
    [Fullname]             NVARCHAR (MAX)     NULL,
    CONSTRAINT [PK_ApplicationUser] PRIMARY KEY CLUSTERED ([Id] ASC)
);

```

Figure 6 - Identity Structure

In the Figure 6 the structure of the SQL table for the user model can be seen in detail, and most of this was provided by the Identity model, but it was extended with Company Name and Full Name in our model. It is easy to recognise the password hash being stored instead of the actual password. This allows for secure login, and we don't have to focus on implementing the encryption ourselves. This was taken into account while picking the technology

stack, and was a great advantage of ASP.NET, which allowed class extension for developing webpages in an Object Oriented Approach.

```
7   namespace FixedThesis.Models
8   {
        19 references
9       public class Experiment
10      {
11          [Display(Name = "Experiment ID")]
            5 references
12          public int ID { get; set; }
13
14          [Display(Name = "Creating user")]
            2 references
15          public string username { get; set; }
16          [Required]
17          [DefaultValue("Experiment 0")]
18          [Display(Name = "Experiment Name")]
19          [Remote("validExpName", "Experiments",
20              ErrorMessage = "You have already an experiment with that name registered"
21              )]
            9 references
22          public string expName { get; set; }
23
24          [Display(Name = "Experiment Status")]
            1 reference
25          public string status { get; set; }
26
27          [Display(Name = "Submission Date")]
28          [DataType(DataType.Date)]
```

Figure 7 - Experiment Model

In the Figure 7, some of the fields of the experiment model are showcased. Inspecting the code, the model can easily be identified as a class that is only composed by attributes, with no methods at all. However, some differences between regular C-like classes can be spotted. First of, the definition syntax includes get and set methods, right after the definition. This is for the controllers and views for being able to access and overwrite the value of the specific field during webpage runtime, and this is needed because of binding them to database models. Additionally, in some of the methods, there are some rules wrote between square brackets. These labels indicate the ASP.NET framework what to expect out of each attribute. Some of them are meant for validation, such as the *Required* tag or the *Remote* tag. Required means that the field has to be filled for being able to push the item to the database, and the Remote tag allows for runtime validation using algorithms defined in the code.

```
public JsonResult validExpName(string expName)
{

    foreach (var item in _context.Experiment.ToList())
    {
        if (User.Identity.Name == item.username)
        {
            if (item.expName == expName)
                return Json(false);
        }
    }
    return Json(true);
}
```

Figure 8 - Experiment Name Validation

In the Figure 8, the code used for validating an experiments name is showcased. The only condition for these remote validation scripts is the return value, which has to be a JsonResult. In this algorithm, we iterate through all the user's experiments and check if the name for the new experiment is already in use by the user. The *_context* variable allows us to check the experiment that is being introduced by the user, even before submitting the form, and this allows for dynamic validation on the client side. In this specific case, the system is not backed up for uniqueness in the database, mainly because of the uniqueness being only restricted to a user, and not to the whole experiment database. I decided not to structure the database in order to generate a different experiment table for each user, so this validation cannot be performed on the server end. This might be troublesome in some spots, but the main issue with name duplication is the user mixing his own experiments, so the client side validation should be enough, since the file system structure is arranged to have a folder for each user, as exposed in Figure 4.

4.5.2 Views

The Views of the system are in charge of the interaction with the user, they define what the user can see, all the styling and the forms for the user to actually send information to the server. This is a critical part of every webpage, and defines important qualities of the final product, especially usability and attractiveness.

Fortunately, the ASP.NET framework implements Bootstrap, and this allows for easy and quick styling without having to expend the time planning the responsive adaptation to the browser of each user, and also designing each of the webpages small components. Instead, it is possible to use Bootstrap's predefined components, such as navigation bars, input forms and buttons, to quickly generate a good looking webpage that adapts to the user's screen size.

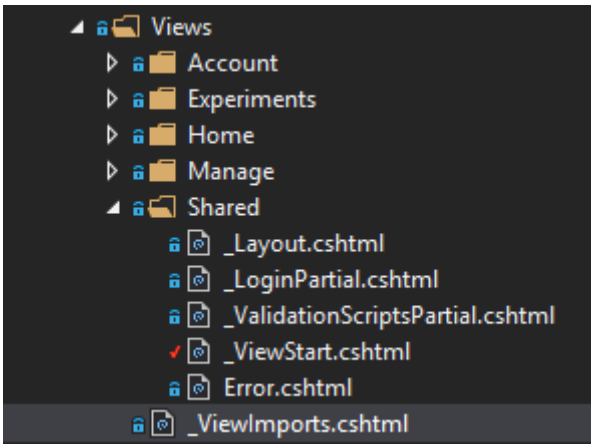
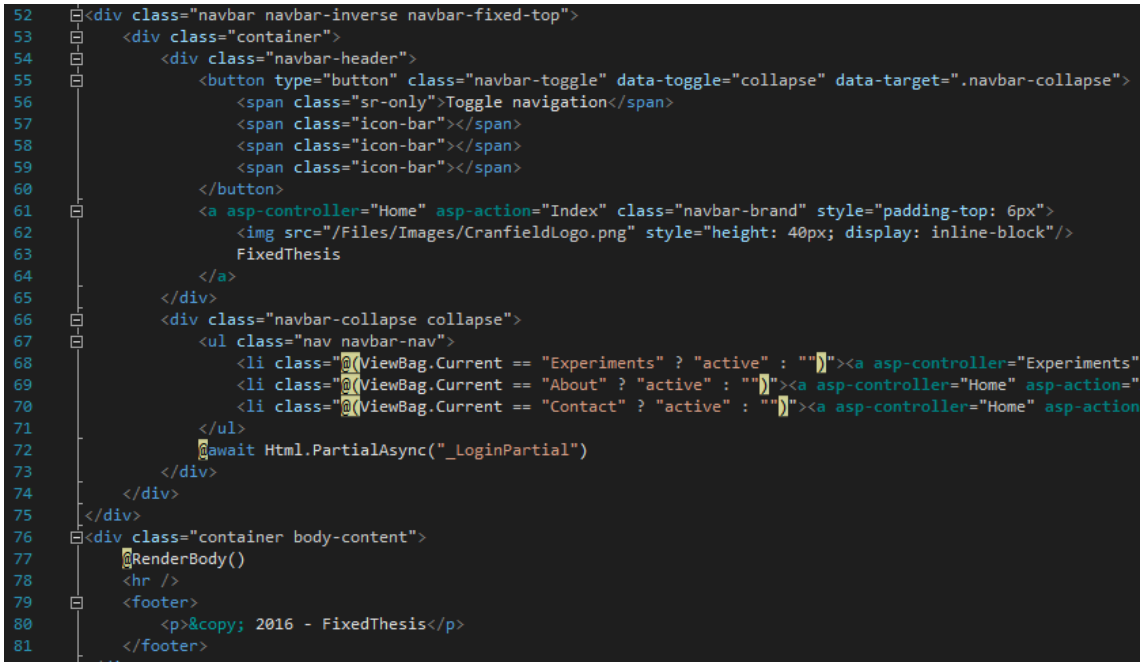


Figure 9 - Web page's shared views

The design of the whole webpages is a bit complex, due to ASP.NET's way to arrange these views. The user interface is not defined in a single cshtml file, but rather is divided in multiple subpages that are arrange in the _Layout.cshtml. In the Views folder, we have one views folder for each of the models, as well as a Manage and a Home folder, which define the views for the admin view of the webpage and the homepage.



```
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a asp-controller="Home" asp-action="Index" class="navbar-brand" style="padding-top: 6px">
                <img src="/Files/Images/CranfieldLogo.png" style="height: 40px; display: inline-block"/>
                FixedThesis
            </a>
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li class="@(ViewBag.Current == "Experiments" ? "active" : "")"><a asp-controller="Experiments"
                <li class="@(ViewBag.Current == "About" ? "active" : "")"><a asp-controller="Home" asp-action="
                <li class="@(ViewBag.Current == "Contact" ? "active" : "")"><a asp-controller="Home" asp-action
            </ul>
            @await Html.PartialAsync("_LoginPartial")
        </div>
    </div>
</div>
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; 2016 - FixedThesis</p>
    </footer>
```

Figure 10 - Webpage Layout file

The most interesting folder, however is the Shared one, in which the whole structure of the page relies. The *_Layout.cshtml* file is the one holding the component arrangement, and if we take a closer look, in Figure 10 we can appreciate the navigation bar division followed by the rest of the body. Nevertheless, two tags stand out as unusual:

First off, the line *@await Html.PartialAsync("_LoginPartial")* is in charge of showing a different end of the navigation bar depending on the user being anonymous or logged in. Therefore, the *_Loginpartial.cshtml* view file is the one in charge of defining this specific end of the bar.

The second important statement to notice is the *@RenderBody()* statement inside the container body division. This is indeed what renders the rest of the webpage, depending on which view are we looking at the moment.

If we consider the rest of the _shared views folder, we only have three: the *_ViewStart*, which defines the Layout to use as the main view. The *_LoginScriptsPartial*, which makes it possible to include all the model validation skills. Finally, the *Error.cshtml* file defines the view to show the user in case anything fails while processing a user request.

Let’s take a closer look to one of the views, in order to fully comprehend the possibilities offered by asp.net. For this purpose, the most complete view developed for my thesis was the Create new Experiment view, which dealt with multiple validation algorithms, as well as file binding.

```
<div class="form-group">
    <label asp-for="temperature" class="col-md-2 control-label"></label>
    <div class="col-md-10">
        <input asp-for="temperature" class="form-control" placeholder="Enter desired experiment temperature" />
        <span asp-validation-for="temperature" class="text-danger" />
    </div>
</div>
<div class="form-group">
    <label asp-for="inputFilename" class="col-md-2 control-label"></label>
    <div class="col-md-8">
        <label class="btn btn-info" style="width:76%">
            Browse file...
            <span class="glyphicon glyphicon-folder-open"></span>
            <input name="uploaded" type="file" style="display:none" formenctype="multipart/form-data"
                   onchange="$('#upload-file-info').html($(this).val()).addClass('alert').removeClass('label')" />
        </label>
        <br />
        <div class="label alert-success" style="margin-top:5px; width:80%" id="upload-file-info"></div>
    </div>
</div>
```

Figure 11 - Create experiment View

In the code, we have a special attribute in the inputs called *asp-for* that defines the field of the model to which we are linking the specific input form. It is also easy to spot the span with the *asp-validation-for* tag right beneath each input. This span is empty in regular conditions, but is filled with an error message whenever a user tries to feed the system an incorrect value.

Then we have the file binding input. We can see this input mapped as a file type, and then we have some Javascript code for the *onchage* tag. This code is bound to show the user some information on the file he chose to upload, and thus he will be able to overwrite it if he accidentally chose an incorrect file. The file validation is not done dynamically, and in case of it not fulfilling the file check, the create view will reload with no file selected while maintaining all the remaining information introduced by the user.

4.5.3 Controllers

The most important part of the webpage are the controllers, the entities that hold all the logic behind a web application, and is here is where most of the developing time was spent.

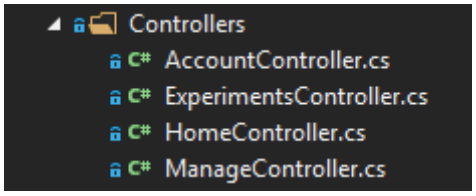


Figure 12 - Webpage Controllers

There is one controller for each one of the models, plus once again the Home controller that defines the logic for the homepage, and also the manage controller for allowing administration tasks to be performed.

The main part of my work in this section was once again while focusing on the experiments section, and ExperimentsController holds the majority of the web applications Logic.

```
namespace FixedThesis.Controllers
{
    [Authorize]
    1 reference
    public class ExperimentsController : Controller
    {
        private ApplicationDbContext _context;
        private IHostingEnvironment _hostingEnv;
        private IFormFile uploadedFile;
        0 references
        public ExperimentsController(ApplicationDbContext context, IHostingEnvironment env)...
        [HttpPost]
        0 references
        public ActionResult uploadFile(IFormFile fileInput)...
        0 references
        public JsonResult validExpName(string expName)...
        0 references
        public FileResult DownloadFile(string filepath)...
        0 references
        public FileResult DownloadInputFile(int? id)...
        1 reference
        public void generateFiles(Experiment exp)...
        // GET: Experiments
        0 references
        public IActionResult Index()...
        // GET: Experiments/Details/5
        0 references
        public IActionResult Details(int? id)...
        1 reference
        private string[] getFileExtensions(string[] fileNames)...
        1 reference
        private string[] getFiles(Experiment experiment)...
        1 reference
        private string [] getFileNames(string [] files)...
```

Figure 13 - Experiments controller overview

In the Figure 13, I exposed a brief overview of the class structure, where some key features can be appreciated. First, the *[Authorize]* tag, that indicates the framework that this controller can only be accessed by an authorized user.

This means that the Controller remains invisible for guest and anonymous users, guaranteeing safety and traceability in case of any error. Also, there are three private variables, the *_context*, which holds all the temporary data of the experiment (all the runtime data not saved in the database), the *_hostingEnvironment*, variable that holds the information needed for dealing with user file uploads, and the *uploadedFile*, which holds the actual file uploaded by the user until is copied to the proper folder and its name is saved in the Database.

Then, some differences can be spotted in the declarations of the different methods. Some private methods are used for intermediate functions used multiple times across the controller. Most of them are tightly bonded to the file handling, which requires directory and extension control, for both tracking and validating the files. These methods are only used inside the controller, and extracting them to an external logic file makes no sense. The methods with a JsonResult return value are used for validation purposes, and usually perform remote checks that cannot be dealt with straight from the experiment model. The rest of the methods are the way the Controller answers the GET and POST requests from the Views previously mentioned. The answer to the GET is just redirecting the user to the appropriate view, sometimes parsing the experiment id received into the actual experiment object.

```
// POST: Experiments/Create
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public IActionResult Create(Experiment experiment, IFormFile uploaded)
{
    if (ModelState.IsValid)
    {
        FormUpload upload = new FormUpload(_hostingEnv, getPath(experiment));
        string filename = upload.SaveFile(uploaded);
        if (filename == "InvalidExtension")
            return View(experiment);
        else if (filename == "InvalidSize")
        {
            return View(experiment);
        }
        else if (filename != "")
        {
            experiment.inputFilename = filename;
            experiment.SubmitDate = DateTime.Now;
            //DEPENDS ON WORKFLOW
            experiment.dateMeshed = DateTime.Now.ToString();
            experiment.status = "Meshing";
            experiment.username = User.Identity.Name;
            experiment.foamMap = getMap(experiment);
            generateFiles(experiment);
            _context.Experiment.Add(experiment);
            _context.SaveChanges();
            return RedirectToAction("Index");
        }
    }
    return View(experiment);
}
```

Figure 14 - Experiment creation POST

In the Figure 14 the logic behind a post for creating a new experiment, which happens after the user submits the information in the Experiment creation form hold by the View. The inputs for the method are an experiment object and a IFormFile object, both bond directly from the view that called the method. After checking the validation through the ModelState, the first action is creating a new instance of FormUpload. This class will be explained later in the project, but is its main goal is handling these file uploading process and creating the appropriate folders or the experiment. Afterwards, we will try to upload the file using the instance, and some checks will be performed based on the result. If everything went right, the Controller will fill some of the database row values and then create the files based on the used template. Finally, the experiment is added to the

context, and the context is updated; before redirecting the user to the index page, where he will be able to check his experiment set.

```
//Method for checking the kind of solving map needed
1 reference
private string getMap(Experiment experiment)
{
    //8314.472 is the gas constant in ideal conditions
    float temp = experiment.expSpeed / (float)Math.Sqrt(1.4 / 29.0 * experiment.temperature * 8314.472);
    if (temp > 1.0f)
    {
        return "Compressible";
    }
    else
    {
        return "Uncompressible";
    }
}
```

Figure 15 - Experiment solving map check

Some of the functions developed for the experiment are related to the nature of each experiment. For example, the getMap function from Figure 15 calculates the kind of map needed to use for an experiment, depending on this experiments velocity and temperature. A different map, would mean having to apply a different flow solve schema, and this type of checks are necessary for the system to be properly implemented. Right now, this fact is not considered for the basic wind tunnel experiment implemented, but it would be really easy to check this value during the file generation, and using different file templates and solving commands.

```
// GET: Experiments/Edit/5
0 references
public IActionResult Edit(int? id)
{
    if (id == null)
    {
        return HttpNotFound();
    }

    Experiment experiment = _context.Experiment.Single(m => m.ID == id);
    if (experiment == null)
    {
        return HttpNotFound();
    }
    return View(experiment);
}
```

Figure 16 - Experiment Details GET Request

As previously stated, GET requests are way simpler than POST requests, and they only need to return the proper view. In Figure 16, we can appreciate the simplicity of method, although it also explains the way to access a certain item from a view. The input parameter of the method is a non-null id, and this id is used to identify the experiment the user is trying to access. After some basics checks, the user is either answered with a no found response or the detailed view of the object. This schema is also followed in other experiment related methods, such as calling the meshing or removing an experiment.

4.5.4 ViewModels

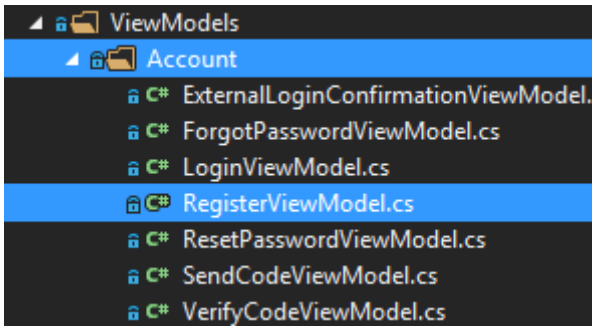


Figure 17 - Webpage ViewModels

Additionally, there are some complicated forms that require an intermediate model before updating the database one. This models do not need to be stored in the database, and are usually meant for making validation easier in complex views. In the thesis, the only important one is the ViewModel related to the User register system. I had to modify the model, for holding additional information and also stop focusing on the email as login or register key.

4.6 Additional Classes

The main webpage structure is really complete and useful, but in order to guarantee the proper workflow and calls to external servers and functions, I had to develop several additional classes.

4.6.1 FormUpload Class

The first class that I created to guarantee the webpages functionality is the FormUpload class, on charge of making it possible for the user to upload his own input files to the web server, and arrange them in the proper folders.

```
public class FormUpload
{

    //Max file upload size
    public const int MAXFILESIZE = 16 * 1024;
    3 references
    private static string UploadDestination { get; set; }
    2 references
    private static string[] AllowedExtensions { get; set; }

    private IHostingEnvironment _env;

    1 reference
    public FormUpload(IHostingEnvironment envir, string DocumentPath)
    {
        //Configuration
        _env = envir;
        AllowedExtensions = new string[] { ".exe", ".diz", ".txt" };
        UploadDestination = envir.WebRootPath + "\\Files\\" + DocumentPath;
        Directory.CreateDirectory(UploadDestination);
    }
    1 reference
    private bool VerifyFileSize(IFormFile file)...

    1 reference
    private bool VerifyFileExtension(string path)...

    1 reference
    public string SaveFile(IFormFile uploadedFile)...
}
```

Figure 18 - FormUpload class structure

As exposed in Figure 18 - FormUpload class structure, the class has four attributes, a constant *MAXFILESIZE* for setting the file size boundaries of the upload, the *UploadDestination* string, which sets the path where the uploaded file will be copied, the hosting environment, and an array of strings that hold all the allowed file extensions for the user. These were set as attributes in order for the developer extending the project to easily adapt it to new necessities, where different file extensions are allowed or bigger files are required, but for now, these values were set low for ensuring the prototype was working. The constructor

requires a string pointing to the destiny path and a hosting environment, both of which will be passed by the controller that calls the function.

The class has a single public method, guaranteeing the proper use of it and minimizing complexity. This method is called after the object is instantiated, and takes care of actually copying the file to the desired folder. This function makes use of the other two private methods for checking that the file extension are size are not out of the set limitations. In case any of those returns a false value, an error string will be sent to the controller, and no file will be uploaded to the folder. If everything goes right, the file will be copied and the controller will get the Filename as a result.

4.6.2 FileHandle Class

The second extension class that I developed is in charge of generating all the files needed for running an experiment in the OpenFoam simulation environment. As stated in previous chapters, OpenFoam requires a specific folder arrangement system, and the best way to generate these files is relying on templates.

```
public class FileHandle
{
    private string inputPath = null;
    private string fileText = null;
    private string outputPath = null;
    1 reference
    public FileHandle(string input, string output)
    {
        inputPath = input;
        outputPath = output;
    }

    1 reference
    public void copyTemplate(Experiment exp)...

    1 reference
    public void updateFile(Experiment exp, string outputPath)...

    1 reference
    private void saveFile(string text, string title)...

    1 reference
    private string patternReplace(string text, string pattern, string newValue)...

}
```

Figure 19 - FileHandle Class Structure

In Figure 19, the structure of the FileHandle class can be appreciated, where the private attributes are strings set up in the constructor call. These strings hold the input path that points to the template file, and the output path for saving the template once the parameters have been changed. The last string holds each files text, and will be used together with regular expression for customizing the template to the user's needs.

```
public void copyTemplate(Experiment exp)
{
    System.Console.Write(Directory.GetCurrentDirectory());
    var files = Directory.EnumerateFiles(inputPath, "*", SearchOption.AllDirectories);
    foreach (var item in files)
    {
        string output = item.Replace(inputPath, outputPath);
        fileText = File.ReadAllText(item);
        updateFile(exp, output);
    }
}

1 reference
public void updateFile(Experiment exp, string outputPath)
{
    string text = fileText;
    var properties = typeof(Experiment).GetProperties();
    foreach (var item in properties)
    {
        string fieldValue = typeof(Experiment).GetProperty(item.Name).GetValue(exp).ToString();
        if (fieldValue != null && item.Name != "ID")
        {
            text = patternReplace(text, item.Name, fieldValue);
        }
    }
    saveFile(text, outputPath);
}
```

Figure 20 - FileHandle methods

In Figure 20, the methods from the class can be seen in depth. The first function basically reads the template files, and for each file and subfolder in the specified path, generates another one inside the experiment folder. The second function is run right after the generation, and updates these files one by one for replacing the references to parameters for their actual value. For this to work, the parameter names in the template are defined with the same name as the ones in the Experiment model. The second function iterates through all the files in the experiment folder and saves them after applying regular expressions.

The class also has two private functions, one for replacing a pattern inside the file and the other one for saving the modified one .Both are called inside the updateFile method.

4.6.3 SSHConnect Class

The final class created for supporting the webpage is the one in charge of the connection to the AWS Flow Solving server. As explained in previous chapters, this connection is easily modifiable, and the class is built in an easy to extend way, so that new experiment schemas can be implemented without further issues.

```
public class SSHConnect
{
    private static string hostDir = "52.31.139.254";
    private static int portNumber = 22;
    private ConnectionInfo conInfo;
    private static string keyFileName = "..\\Assignment.pem";
    private static string username = "ubuntu";
    private SshClient myClient;
    private ScpClient myScpClient;

    1 reference
    public SSHConnect()...

    0 references
    public string connectToServer()...

    1 reference
    public string createDirectory(Experiment exp, string Username)...

    1 reference
    public string copyFiles(Experiment exp, string Username)...

    1 reference
    public string makeMesh(Experiment exp, string Username)...

    0 references
    public string solveMesh(Experiment exp, string Username)...
}
```

Figure 21 - SSH Connect class structure

As can be appreciated in Figure 21, the structure of this file needs a bigger setup. The attributes are private and some of them static. Those are the ones related to the connection to the specific server. There are also two additional

Client attributes, that are needed for opening the connection to the server, either for a command line interface (ssh) or a file transfer (scp).

Then the methods implemented make use of these private attributes for the connection handling, and I designed different functions for each one of the stages needed for fulfilling the OpenFoam workflow.

The whole class was developed using the Renci SSH.NET library [18], which is not implemented in DNX5 yet, but could be used in the webpage combined with DNX core 4.5.

```
public string createDirectory(Experiment exp, string Username)
{
    myClient.Connect();
    string makeCommand= String.Concat("mkdir -p ",Username, "/",  exp.expName);
    var makedir = myClient.CreateCommand(makeCommand);
    makedir.Execute();
    var getdirs = myClient.CreateCommand("ls");
    getdirs.Execute();
    myClient.Disconnect();
    return getdirs.Result;
}
```

Figure 22 - SSH Create directory

The method shown in Figure 22 is in charge of creating the needed folders for the experiment in the OpenFoam enabled server, and for that purpose takes the username and experiment as inputs. It has two different commands, the first one is the actual command line action, and the second one is for retrieving the file structure of the folder once it was created. The commands are created as variables using the *client.CreateCommand()* function, and then run using *client.Execute()*. For being able to run the commands, we have to open and close the ssh connection using the myClient attribute's *Connect and Disconnect* methods.

5 RESULTS

5.1 Overview

This section of the thesis is focused on highlighting all the accomplishments met during the extension of the project. Before showing the actual webpage and results, I will outline the main accomplishments that were fulfilled.

- ✓ Generated an experiment database for storing all the details of each experiment
- ✓ Implemented a user login system with Userbase and Password seeded to the system
- ✓ Linked each system user with his own experiments and displayed the state of each one of his experiments
- ✓ Allowed each user to create a new flow solving experiments
- ✓ Allowed users to upload their own files to send for simulation
- ✓ Generate an experiment template for calling the OpenFoam Solver
- ✓ Setup a OpenFoam and Paraview enabled Web Server using Amazon Web Services
- ✓ Created an SSH Connection Library for interaction with the OpenFoam Server
- ✓ Generated a FileHandling class which used the OpenFoam template and changed the required parameters for each experiment
- ✓ Implement a simple workflow solution, using the cavity example from OpenFoam

5.2 Webpage

Now I will showcase the main parts of the webpage, which acts as the user interface with the flow solving system.

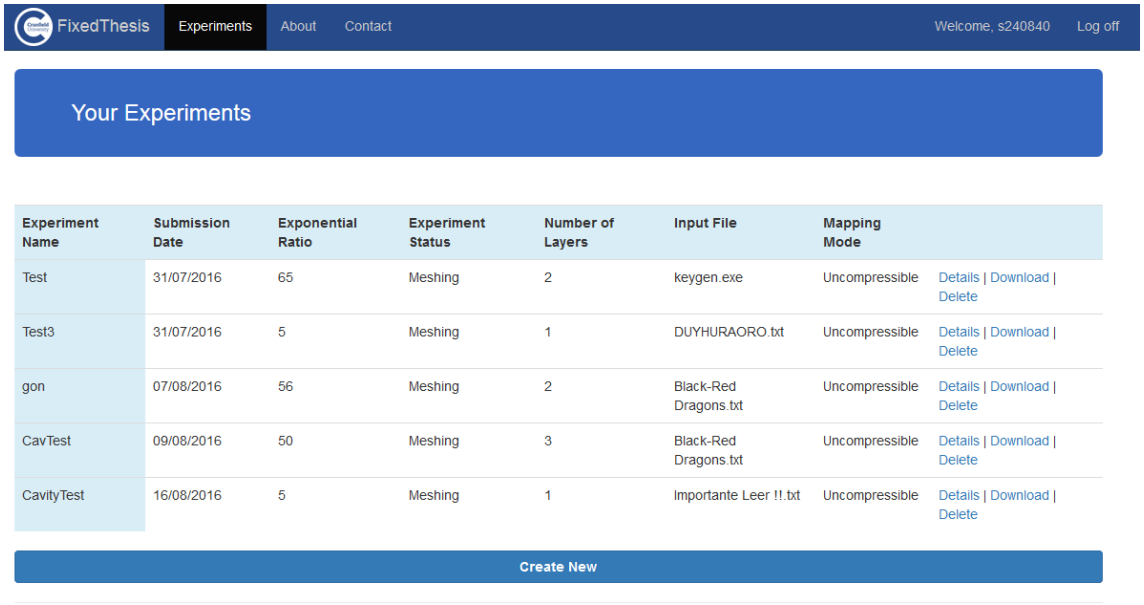


Figure 23 - User experiments summary

In the Figure 23, the summary of all of the user's experiments is showcased, with basic information on each one of the experiments. From this view, we can create a new experiment, get the details of a current experiment and delete them. The basic details from the experiment are name, the submission date, the Experiment status, some experiment fields that can be modified, the mapping mode and the actions that can be performed in the event. The mapping mode is showed for showcasing that this value can be calculated using the controller method *getMap()* that was explained in the previous chapter.

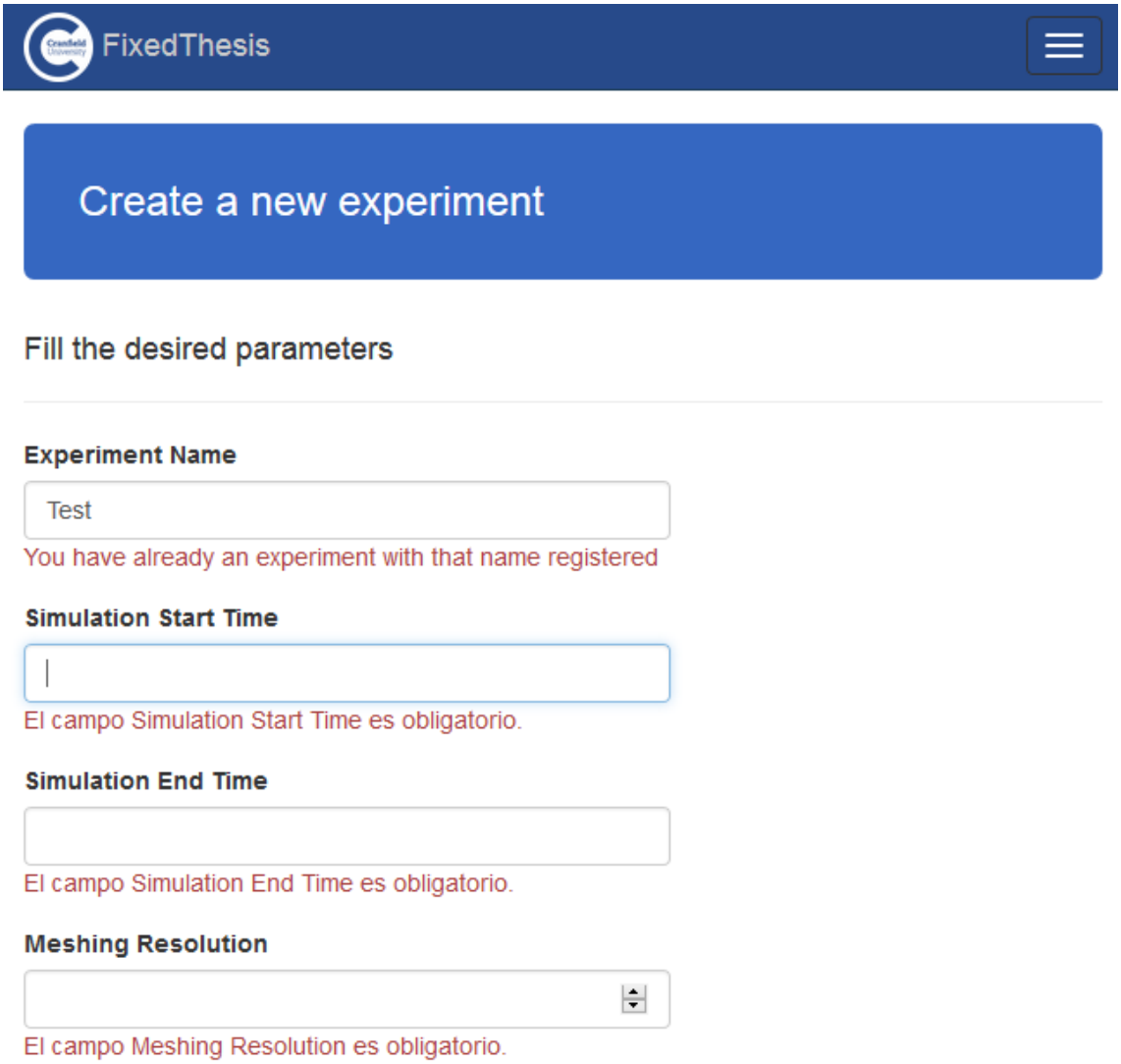


Figure 24 - Failed experiment creation

In the Figure 24, we tried to create a new experiment, but we did not fill the proper values, and the validation script is failing. As we can see, web will not allow us for the introduction of a new experiment with the name of another one. We will fill all the values properly, using the username CavityTestingExperiment.

In the, Figure 25 show how the webpage deals with removing an experiment. This process does not only remove the entry for the database, but also the files from the wwwroot Files folder, so that the user cannot see that experiment anymore. Before removing the item, the user is showed some details on the experiment, and given the option to confirm the file deletion or just go back to the list of experiments.

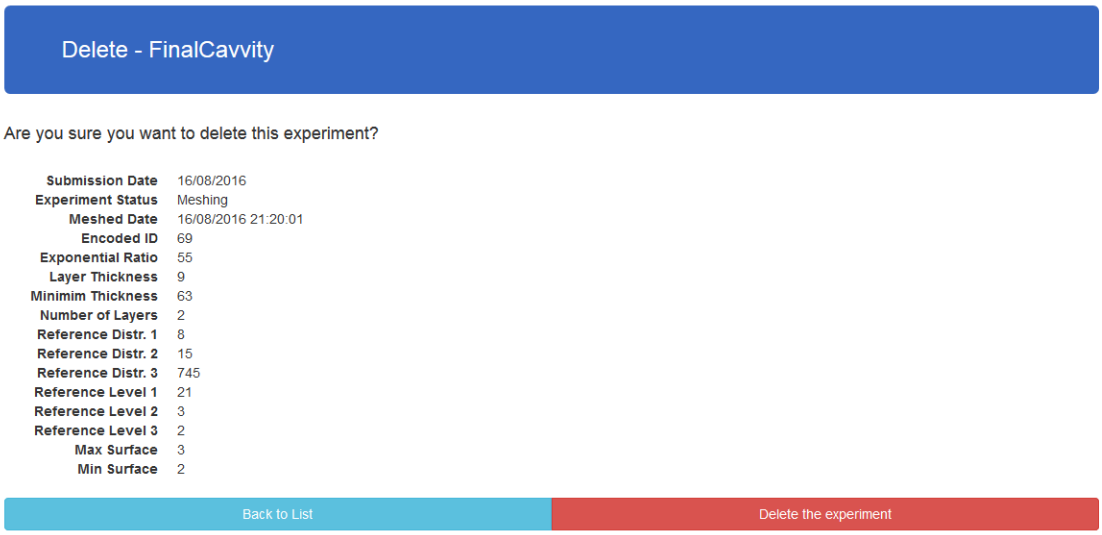


Figure 25 - Experiment removal

Finally, the detailed view of an object not only shows all the fields filled for the object, but also has a file list, where the user can download all the files related to the experiment, as can be seen in Figure 26. The experiment was just created, and we can already download all the files from the adapted template. After running through some of the other stages, the result file swill be added to this table, and the user shall be able to retrieve all the results.

Figure 26 - Experiment files

The next figure shows the user register system, where the user has to fill the following fields: username, email, full name, company name, password and confirmation. The username has to follow the pattern used for the cranfiield university usernames, hence a letter followed by a six digit number, as can be seen in the

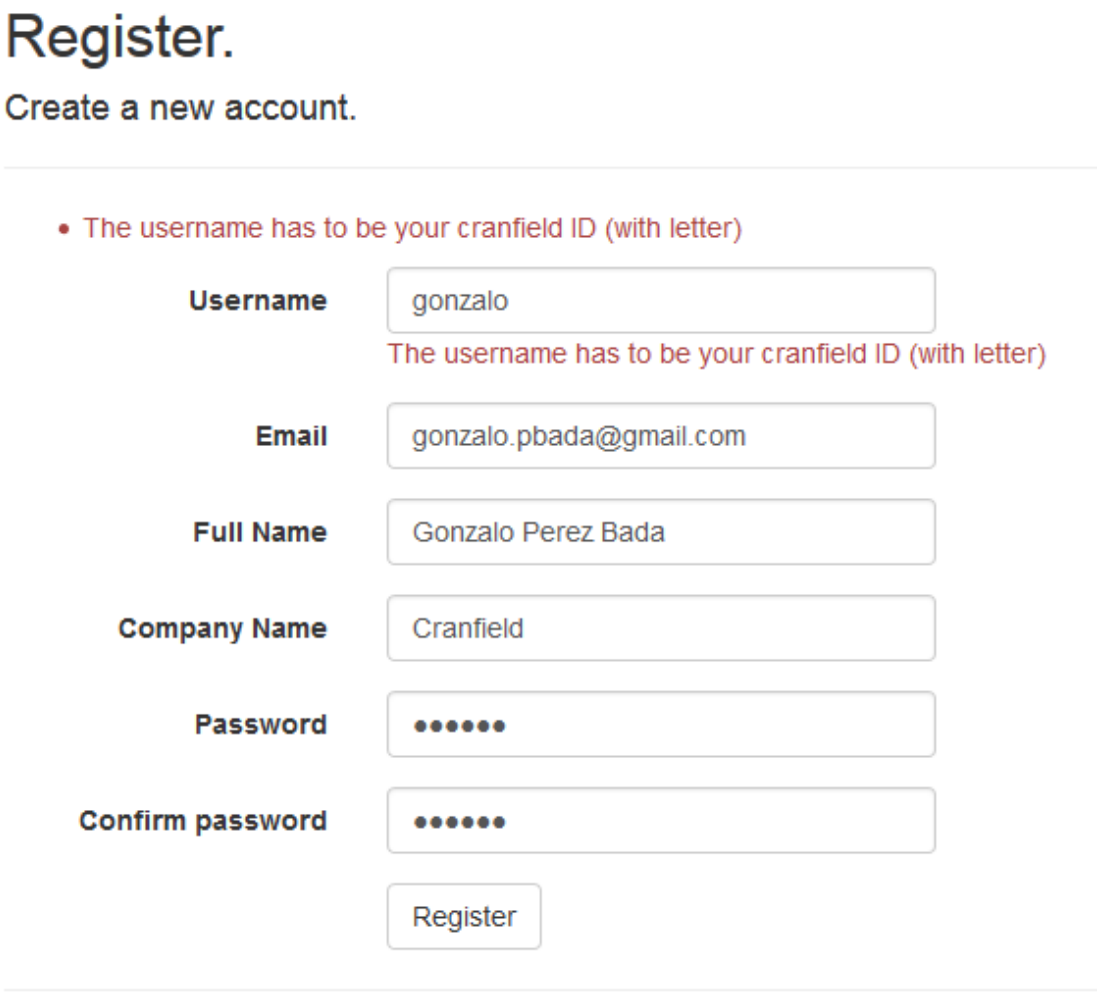


Figure 27 - Register view

This register script is designed to only allow Cranfield user IDs for registration, but could be worked a bit, with confirmation emails or even setting up admins to confirm user's identities. Additionally, it shouldn't be difficult to perform a check on Cranfield University's database to confirm the username and email or full name are linked.

The webpage is fully functional, although it has plenty of room for upgrades and additionall functionalities. However, the result obtained offers a really easy to understand interface that can be easily extended.

5.3 Cavity Example

For demonstrating that the webpage can deal with OpenFoam experiments, I decided to use the cavity tutorial as a template. This experiment

is a really simple one, but it's good enough for demonstrating the workflow through the webpage, and serves as a demonstration of the whole process. This is the proof that the workflow can be handled by the webpage in a seamless way for the user, while not having to worry about performance issues of each stage, since those are my colleague's thesis' goals.

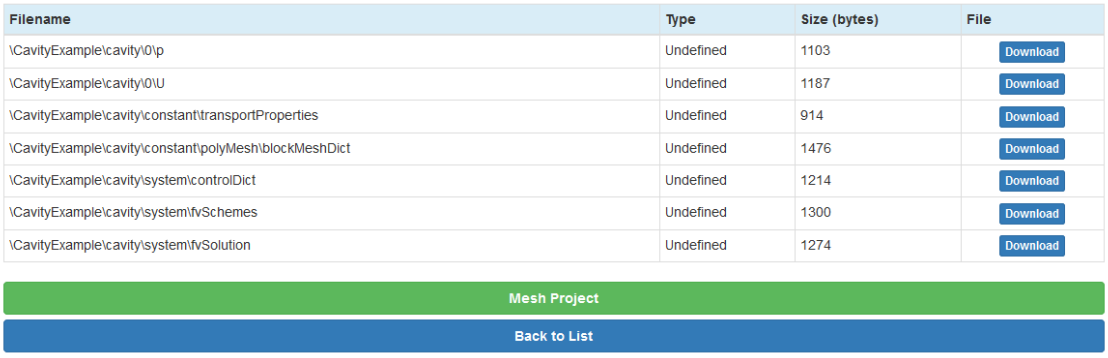
The cavity example is one of the OpenFoam tutorials, and a computational walkthrough is available at [19], where the whole experiment basics are explained.

The webpage was setup to use the tutorial file as an example, and the related parameters were introduced into the input files, for the FileHandle class to modify them. In this section, we will cover how a user can perform this experiment using the Wind Tunnel Webpage. In the next figure, we can see the needed inputs for the user to fill in order for the experiment to work.

Figure 28 - Cavity experiment creation

The only parameters that will be taken into account for the experiment are the ones noted above: the simulation start and end times, the meshing resolution, that affects the way that blockMesh will work, and the Kinetic viscosity, which affects the final solution. This way, we can check that the results are actually changing due to each stage, since we have parameters related to both meshing (pre-processing) and the flow solving. I created an experiment that will run simulations from 0.1 to 0.6, with a meshing resolution of 40 and a kinetic viscosity of 0.01. As a file input, the user has to upload the p file, which holds the pressure field initial values, and has to be stored in the 0 folder, where all the initial

conditions are saved. That folder is generated by the webpage, and the only missing file was this. The design could be however adapted to take a different input, but the FileHandle class should be adapted in consequence. The p file is available at the first appendix [6.1.7Appendix A].



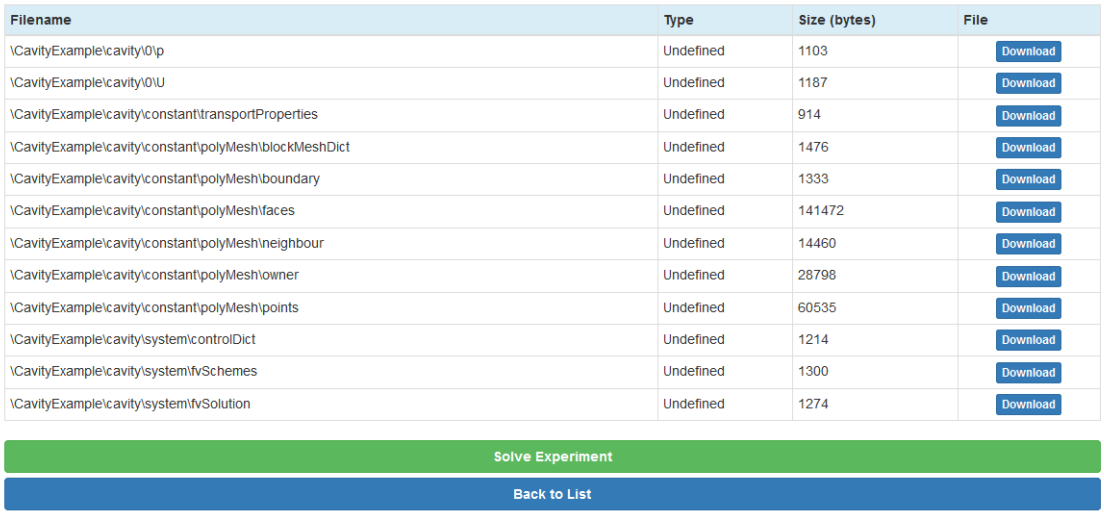
Filename	Type	Size (bytes)	File
\CavityExample\cavity\0\p	Undefined	1103	Download
\CavityExample\cavity\0\U	Undefined	1187	Download
\CavityExample\cavity\constant\transportProperties	Undefined	914	Download
\CavityExample\cavity\constant\polyMesh\blockMeshDict	Undefined	1476	Download
\CavityExample\cavity\system\controlDict	Undefined	1214	Download
\CavityExample\cavity\system\fvSchemes	Undefined	1300	Download
\CavityExample\cavity\system\fvSolution	Undefined	1274	Download

Mesh Project

Back to List

Figure 29 - Cavity initial state

In Figure 29, the initial state of the experiment is showed, where we can not only see and download the initial template files copied to the experiment folder (with their file types), but also a button at the end of the page, that runs the meshing part of the experiment.



Filename	Type	Size (bytes)	File
\CavityExample\cavity\0\p	Undefined	1103	Download
\CavityExample\cavity\0\U	Undefined	1187	Download
\CavityExample\cavity\constant\transportProperties	Undefined	914	Download
\CavityExample\cavity\constant\polyMesh\blockMeshDict	Undefined	1476	Download
\CavityExample\cavity\constant\polyMesh\boundary	Undefined	1333	Download
\CavityExample\cavity\constant\polyMesh\faces	Undefined	141472	Download
\CavityExample\cavity\constant\polyMesh\neighbour	Undefined	14460	Download
\CavityExample\cavity\constant\polyMesh\owner	Undefined	28798	Download
\CavityExample\cavity\constant\polyMesh\points	Undefined	60535	Download
\CavityExample\cavity\system\controlDict	Undefined	1214	Download
\CavityExample\cavity\system\fvSchemes	Undefined	1300	Download
\CavityExample\cavity\system\fvSolution	Undefined	1274	Download

Solve Experiment

Back to List

Figure 30 - Cavity meshed files

Once clicked in the button, we can go back to the experiment details and check that not only the experiment state was generated, but also new files have been added to our experiment folder. These files are inside the

constant/polymesh/ folder, and hold the mesh data generated in the ASP.NET server using the blockMesh command. This generated mesh used the meshResolution value we set in the previous steps.

Figure 31 - Cavity timestep results

After running the solver, which calls the server again for running the icoFoam command, with the kinetic viscosity that was set on the first stage, running in the time interval that we set. Now for each one of the timesteps set, we get four files that contain the simulation results.

U and p are still the velocity and pressure fields, but the values we stablished as initial values are modified in each time step. In this specific simulation, the pressure does not change that much, but in more complex situation, these changes are more obvious.

The result files also contain a phi file that holds the values of the air flow going through the cells defined in the experiment, and also a uniform/time file that holds time information, and although not useful in this case, generates information needed for running parallel simulations.

After this stage is simulated, the experiment has no additional stages to go through, so its status is changed to done and the meshing and simulating actions are no longer accessible.

6 CONCLUSION

The final result of the thesis is a functional and well styled webpage that allows the user to configure his own way the cavity experiment. This experiment could be configured in a more advanced way, and additional experiments could be setup with minimal effort, but due to time constraints, I decided not to focus on

these alternative implementations, since that would require a deep understanding of OpenFoam, and I had no previous background on it.

The process that I went through during the thesis was not properly defined due to the functions about to use in the experiment not being defined yet, and also the lack of definition of the wind tunnel experiments to run or not, so I spend a great part of my worktime trying to figure out the best way to approach a general solution to the workflow issue. Some problems of dependencies also arouse during the project and caused changes to the technology stack and library calls.

Considering the complex nature of the project and the need to coordinate with thesis that were still under development, I am indeed happy with my approach to the thesis, and I consider that this will allow future expansions to implement a robust and optimised wind tunnel experiment.

I am also proud of the final solution I managed to get, since both the webpage, file system and additional libraries provide a great structure for any wind tunnel experiment, and duplicating the controller would allow experiment specific customization, requiring minimal time refactoring the code, and adding additional fields to the navigation bar.

However, I also think that this thesis had some issues, and probably the whole development process would have been smoother if the tools for each computing stage had been available at the start of the thesis. This made the solution generated a bit too general, and not really useful for real world simulations. That was a scope redefinition I had to take, and I am confident on this being the best available option giving the situation.

6.1 Further Work

This thesis was focused on potential improvements, hence, there are plenty of features to improve and upgrading to do before achieving a completely functional online web tunnel simulation. In this section, I will introduce the main goals that were left out of scope during the work period, but were taking into account during design. I will also comment some options that I considered to add

during the latest stages, but decided not to due to time constraints or lack of resources.

6.1.1 Connection to a HPC Facility

The first thing to consider is using OpenFoam commands to exploit the Astral computers parallel capabilities. This was the initial idea during the early stages of the project, but when I discovered that Globus was no longer available for use and moved away from WWF, I had to forget about this possibility.

Nevertheless, the ssh connection approach used is really powerful for implementing this parallel approach, since it can be restructured for generating a run script and submitting it to a super computer. This whole process can be done only by modifying the methods inside the ssh connection class, and adding a scripting function inside the FileHandle class.

6.1.2 User groups

Another important feature to add to the webpage would be shared experiment folders, where researches could share their experiments' details and results. Most of the research nowadays is performed by groups rather than individuals, and the webpage should support the research groups and university departments in the Wind Tunnel framework.

For implementing this, two main actions would be needed. First, set a Groups folder should be arranged, and a way to save the experiment files and details shall be designed. Additionally, the database models should be changed, in order to have a link between users and groups, and also a link between an experiment and the related group (if this experiment is shared)

6.1.3 Error Handling

One of the main features I did not consider during development was Error handling inside the webpage, and sometimes this can generate some troublesome spots in the webpage, where the user ends up in empty pages.

This should be one of the main priorities to handle in the short term, and it mainly infolves exception throwing and handling inside the C# code. However, I was not sure about the interaction between C# and .NET framework in exception spots, and I decided not to spend too much time in this kind of features.

6.1.4 Complex Wind Tunnel experiment

The whole webpage functionality was designed taking a really simple tutorial in mind. This made the webpage easier to define, and was probably the only way to get a suitable product at the end of the timeline, but also means that some function adaptation to other experiments can be proven difficult.

However, by coordinating with some people who are more experienced with OpenFoam files and commands, it can probably be dealt with in a short time lapse.

6.1.5 File validation

The user can upload any file to the server, aside of some simple extension checks, and this can be a hazard to the webpage and the flow solve server.

Another one of the first stages to take in the further development of the project would be dealing with this validation, probably by performing a format check inside the OpenFoam file, so that the webpage can reject files that OpenFoam will not be able to simulate, and at the same time get rid of some security issues.

6.1.6 Asynchronous handling

ASP.NET allows for asynchronous tasks inside the controllers, and this would be the best way to run the connection to the Flow solve server. The final version of the thesis runs this calls synchronously, meaning that the user will have to wait for his browser to finish his simulation.

This can be quickly fixed using additional processes running asynchronous tasks, but this was not implemented due to two reasons. First, the task handling is not straightforward, and could mean a complicated task handle

scheme. Second, one of the first improvements to add is parallelisation, and this meant dealing with HPC queuing, and hence, the way to call the functions and retrieve the data might have been redefined, so the work spent on this task could be useless. This is the reason that this upgrade is explained after the parallelisation.

6.1.7 User Register and Login System

Although ASP.NET provides a great system for commercial users, where anyone can register and use the webpage, this project might require additional security checks, and not everyone should be able to register.

Therefore, a user validation system should be implemented, either by checking an internal database that holds the allowed user details, so that these can be check upon registry or by requiring an authorized user to validate someone else's registry request. The first method's main advantage is automatization of the register process, while the second one means that the user base can be controlled more easily, and could improve some security constraints.

REFERENCES

(1) National Aeronautics and Space Administration (2016) Wind Tunnel Design. Available at https://www.grc.nasa.gov/www/k-12/airplane/tunnozd.html

(2) Y. Zhao, I. Raicu and I. Foster, Scientific Workflow Systems for 21st Century, New Bottle or New Wine?, 2008 IEEE Congress on Services - Part I, Honolulu, HI, 2008, pp. 467-471.

(3) A. Pavethan, K. Takeda, S.J. Fox, D.A. Nicole, Workflows for Wind Tunnel Grid Applications (2006). Lecture Notes in Computer Science Computational Science – ICCS 2006, 3993, 928-935.

(4) A. Pavethan, K. Takeda, S.J. Cox, D.A. Nicole, MyCoG.NET: a multi-language CoG toolkit (2006). Concurrency and Computation: Practice and Experience – Wiley InterScience 2006,

(5) A. Gartman, W. Fister, W.Schwanghart, M.D. Muller, CFD Modelling and validation of measured wind field data in a portable wind tunnel. Aeolian Research, 2011, 315-325, University of Basel, Basel, Switzerland

(6) A. Pashami, S. Asadi, A. Lilienthal, Filament-Based Gas Propagation Models for Gas Dispersion Simulation (2010), Proceedings of the Open Source CFD International Conference, Örebro University, Sweden

(7) Codd, E. F. (1982). Relational database: a practical foundation for productivity. Communications of the ACM, 25(2), 109-117.

(8) Michael Stonebreaker, Communications of the ACM (2009) The NoSQL Discussion has nothing to do with SQL. Available at: http://cacm.acm.org/blogs/blog-cacm/50678-the-nosql-discussion-has-nothing-to-do-with-sql/fulltext (Accessed: 3 June 2016).

(9) Flanagan, D., (2006). JavaScript: the definitive guide. "O'Reilly Media, Inc.".

(10) De Volder, K. (2006), January. JQuery: A generic code browser with a declarative configuration language. In *International Symposium on Practical Aspects of Declarative Languages* (pp. 88-102). Springer Berlin Heidelberg.

(11) Branas, R. (2014). *AngularJS Essentials*. Packt Publishing Ltd.

(12) Lie, H.W. and Bos, B. (2005). *Cascading style sheets: Designing for the web, Portable Documents*. Addison-Wesley Professional.

(13) Cochran, D. (2012). Twitter Bootstrap Web Development How-To. Packt Publishing Ltd.

(14) Hibbs, C. (2005). Ruby on Rails. *OnLamp. com. O'Reilly Media*, *13*.

(15) Duthie, G.A. and MacDonald, M., 2003. *ASP. NET in a Nutshell*. " O'Reilly Media, Inc.".

(16) Cui, W., Huang, L., Liang, L. and Li, J., 2009, November. The research of PHP development framework based on MVC pattern. In *Computer Sciences and Convergence Information Technology, 2009. ICCIT'09. Fourth International Conference on* (pp. 947-949). IEEE.

(17) CFD Direct, The Architects of OpenFoam (2016). CFD Direct From the Cloud: OpenFOAM on AWS EC2. Available at http://cfd.direct/cloud/

(18) Secure Shell Library for .NET, optimized for parallelism. Available at https://github.com/sshnet/SSH.NET/

(19) CFD Direct, The Architects of OpenFoam (2016). OpenFOAM user guide: 2.1 Lid-driven cavity flow. Available at . http://cfd.direct/openfoam/user-guide/cavity/#x5-40002.1

APPENDICES

Appendix A Cavity Pressure initial conditions

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }

    fixedWalls
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
}

// ************************************************************************* //
```