The Development of a Web-based Multidimensional Data Plotter

AC40001 Honours Project

Sameer Al Harbi
ID: 190007048/1
BSc (Hons) Computing Science
Supervisor: Dr. Iain Martin
University of Dundee
Dundee, UK

ABSTRACT

This **should** be a *summary* of 250 words that summarises your project, including your aims and contributions that will arise from it. Think of it as a summary of your introduction.

1 INTRODUCTION

Data visualisation is an important step in the data analysis process. An importance which is only further compounded by the rise of Big Data and the subsequent need for extracting insight from exponentially larger and more complex data sets. This insight, whether it's used to highlight results in an easy to digest manner or to identify patterns otherwise hidden by the complexity of a dataset- can bring value across a project's team, with each member being able to extract value specific to their own tasks.

For all members working on a project (With the word project being used in the context of any work done that uses data within an industrial or academic sector) visualisation is the main "window" through which insight and thus value is accessed. Analysis is done towards some end- and that end must result in the return of something- which often must be visually consumed (See Big Data above). In many cases visualisation in itself can act as a form of analysis, both creating insight and communicating it when applied to a dataset.

To be more specific, it has been found that there is a definite market among both technical and non-technical professionals for tools that can unlock the results of analysis.

The aim of this project was to then create an accessible, web-based application that can be used by anyone to quickly plot data in multiple dimensions, regardless of programming skill. This paper highlights the decisions that lead to this final

2 BACKGROUND

The result of background research done for this project can be subdivided into two distinct sections being Related Work and Application.

2.1 Related Work

This section highlights the wider context that affected the structure and what the application was planned to do. The research identifed here acted to directly influence what kind of application was developed, for what exact purpose and for what users.

2.1.1 Project Context. One of the first steps for this project was to place and understand the needs of this project through the lens of some wider context. By what metric should potential requirements be prioritised? and what those requirements should even be? Those are some of the questions whose answers will depend on the context that they are looked at. After some consideration, it was decided by the student to focus on the current business industry, and it's needs as this context.

> This context (the business market) sets the aim of this project to then be an application that would be considered a valuable market entry compared to its competitors. It should also preferably, address some specific pain points of the market that are not fully covered by some other solution on the market.

The alternative would also have been to look at this project from a more academic, literature driven view- where a project's success would depend on more research oriented exploratory work. This alternative though, had some drawbacks that ultimately resulted in the selection of a Business Context. Mainly, a focus on existing market solutions allowed a more logical selection of features to develop for a practical development project. Although it should be stressed that this doesn't mean that the academic background for this project was disregarded (See Section 2.3)- It's just that the focus was on the business-based requirements first and foremost.

2.1.2 Market Research. With the context set- It was then considered that an analysis of the current options on the market would be a prudent next step to start understanding what a successful project should incorporate. This analysis was open-ended and focused on identifying key points that differentiated each product apart.

Detailed notes on each competitor can be found in the Appendix [A], which are although not exhaustive, nevertheless highlight some important aspect of the overall market- But the main important trends and highlights identified are the following:

Firstly, it was found that most solutions are highly technical and need at least some degree of programming knowledge to use. A rough relationship would be that the complexity of using the solution and its capability is inversely proportional. Feature rich solutions need programming experience while the ones that don't, have more limited features, and are more likely to be not free to use. Obviously, there were outliers such as MATLAB which has both a UI interface and a programming interface, but this relationship still stands.

Another interesting point identified is that The Stack Overflow 2021 Developer Survey [1] shows that a large majority of the most used libraires and frameworks were for data analysis or data-based projects, specifically Python. Which is also the third most used

language identified by the Survey [2]. Although these statistics don't directly relate to visualization solutions, they help form an important background to the context in which visualization may be needed in. Which is that Python and its libraries are a highly prevalent option for undertaking data analysis projects, meaning that any visualization solution is likely to be part of a workflow that contains these tools.

But Solutions focusing on Visualization first and foremost seemed to be much fewer when compared to analysis solutions with visualization features. Those that were, were also much more likely to be highly technical. This point was found particularly important because data analysis and by extension visualization has been found to be needed in a wide field of industries, where programming expertise is not as widely common. And this need is expected to rise in the future with the advent of Industry 4.0- Which is a widely believed idea of increasing business productivity fueled by disruptive emergent technology in the near future [3].

Taking all this into consideration, several key points were identified on how to structure a new development project to create a valuable solution from the market's point of view:

- Focus on accessibility first and foremost, including an easy-to-use interface that does not need programming experience.
- Focus on the visualisation aspect, but either ensure that adequate analysis tools are available or importing or exporting data is easy and straight forward. Visualisation is often only one step of a multi-step analysis project.

On the other hand, some thing's to avoid are:

- Focusing on the features but not accessibility. It is unreasonable to try to beat the current market players on features alone You can't get more feature-rich than a programming language such as Python, which is arguably one of the most used solutions on the market, See Appendix A.3 for more information. It is important instead to create an alternative that is powerful enough but makes the process of data visualisation much easier.

2.1.3 Academic Background. A General study of research in the field was conducted with a focus on identifying any other similar projects such as this one undertaken by the student, and more importantly the wider context from an academic point of view. Which was followed by a comparison to previous market research in section 2.2. This was done to give the student a better understanding of how visualization works in different settings and if there are any trends and patterns that can help prioritize functionality or uncover new opportunities.

This was done by analyzing a small subset of important research papers talking about data analytics and visualization. Some of these papers focus on market research which were particularly important to expand upon the market research done by the student themselves. The most important points identified are mentioned below (Consider this a continuation of the market research / section 2.2 points above):

- Current visualization techniques are not capable of handling big data. New solutions are being created and investigated by both industry and academia []. This is a big point that this project could contribute to fixing.
- No matter how good a visualization is- if it cannot be understood its value is nullified. There currently exists a gap between what computation is capable of creating and what can be easily grasped by a user. Naturally, there exists opportunity in creating solutions to close this gap.

2.1.4 Other Research. Another important aspect of research that was done included a more technical look into general design points that could help drive the design of the application itself and how features are developed.

One specific aspect researched has been software planning methodologies. The student was aware of agile and waterfall techniques from their study but those techniques were usually applied in and made for multi-person teams, which naturally conflicted with the single-person structure of this project. Through this research, a technique called PSP [] (Personal Software Process) was discovered. This is a methodology that allows a singular developer to apply a structured, continuously improving process to how they develop software. A further inspired process was also identified called PXP [], which was a fusion of Agile XP principles adapted to a single person team inspired by PSP.

Another notable finding in this phase, was an analysis of graphing and visualization techniques. In the 2.2 Market Research phase a lot of different solutions with a large variety of graphing types were seen. An analysis of the most common ones was undertaken with the hopes of identifying the most suitable graphs in terms of flexibility at higher dimensions (3+). The full analysis document can be seen here [] but in summary- It was found that Scatterplots fit these needs the most.

2.2 Application

With the Application Context and Background set, the next step for this section is to highlight what technologies and design paradigms would be available for the development of this project followed by a comparison between them. No decisions were made during this stage of the project- The research and analysis recorded here act to justify and shortlist the final design in Section 4.

It should also be noted that obviously not all combinations of technologies mentioned below are viable. Usually deciding on one aspect would limit what can be made for another. But nevertheless, each section below was looked at separately to fully understand the opportunity cost for each decision when going one route as opposed to another.

2.2.1 Application Infastructure. This section specifically focuses on the structural decisions in the design of an application that directly affect how it is created, run and in some cases what features are possible to implement. In general, the design options for this application in particular could be split into two groups.

Client-Side Run Application. This is the simplest design. All code that makes up the application is run on the user's device locally. No need for any server resources (other than for serving the initial code which even then may not need a server, Disc etc..) thus can

be ran offline. But resources are limited to only what the user has and no way to synchronize data among multiple users- such as for user accounts, etc.

Server-Side or Full Stack Application. This is a solution to some of the limitations mentioned for Client-side. Either have the client application be able to connect to a central (or server-less functions based) server and offload some tasks to it, or have the application be fully computed on-server with only static content responses being sent to the user. This design is naturally more complicated and often results in a split codebase among backend and frontend components. There is a need to provision computing resources, and more security considerations need to be made.

2.2.2 Run Environment. This can be defined as the environment where the application will run. The main options identified were the following:

.NET Environment. .NET, which is an open-source developer platform for building Software mainly developed by Microsoft [4]. It is cross platform among Windows, MacOS and Linux (With support for iOS and Android using frameworks such .NET Multi-platform App UI / MAUI [5]). It provides a great rich set of libraries and tools for any kind of projects including both frontend and backend webbapp needs and client ran applications (As long as .NET Framework is installed). .NET supports only 3 languages those being C#, F# and Visual Basic.

Node.js. Node.js is aptly described on its website as an "an opensource, cross-platform JavaScript runtime environment." [6] It is the most popular web technology among the Stack Overflow 2022 Survey respondents [7]. Although most commonly used for Serverside processing as a webserver, it can also be used for running local applications as long as it is installed on the client's device (although it usually makes more sense to serve the app over a network). For package managers, npm is tightly linked with node.js and provides a huge repository of packages. With the addition of Web Assembly, which is a new standard assembly like language that acts as a compilation target from a wide range of other languages, it is technically possible to use almost any language with node.js that has a compiler to web assembly made.

Browser Engine. Browser Engine's- Modern browsers all have some JavaScript runtime engine, with V8 being the most widely used engine [8] Although all browsers are supposed to be crosscompatible, in practice this can vary, and some incompatibilities can arise. Browsers naturally only run client-side code but communicating with backend solutions is a common practice. Only JavaScript is natively supported and Web Assembly in "4 major browser engines".

JRE (Java Runtime Environment). JRE, is a runtime environment that allows an application to be cross compatible between Operating systems by acting as a compatibility layer. It runs Java bytecode which can be compiled from a large variety of languages. A standard library is also available with the runtime environment. Commonly used for backend development but also capable of Frontend.

Compiled Program (OS only environment). Compiled Program, One of the most flexible options on the list. Languages such as

C++ and C compile to machine code which are run directly on a user's device. No inherit cross platform support and a need to recompile for each OS. Usually, higher performance due to lower abstraction. But a lower abstraction means more functionality needs to be managed by the programmer.

2.2.3 Rendering Solution. Being a project with visual requirements naturally meant that some rendering technology would be needed. Like with all other technologies mentioned thus far- there is a wide range of options that each have their own distinct design and ability. This section then aims to segment the available options by two factors at the minimum- One, the rendering solution must support 3D graphics in some capacity, and two, the solution must run and integrate with an application targeting one of the above-mentioned run environments. With those key requirements in mind, these were the identified options:

OpenGL. OpenGL is a low-level cross-platform rendering API which can be traced back to a release date in 1992. Although no longer in active development as of 2017, OpenGL still remains highly supported across both newly releasing GPUs and older. This also includes mobile devices. In terms of language bindings, OpenGL is cross-language and can be called from almost any language and environment that has had bindings made for it.

Vulkan. Vulkan is a cross-platform low level open standard rendering API that has superseded OpenGL in active development. It's main driving improvement over OpenGL is lower overhead and more control over how code is run on the GPU. This greater control though does mean that development is more time consuming over OpenGL. Being much newer, with the initial release date being 2017, it is much less supported among older devices than OpenGL.

DirectX3D. DirectX3D is a subset API of the proprietary DirectX family of multimedia APIs developed by Microsoft. It is a lowlevel API similar to OpenGL but created exclusively for Microsoft Windows, Xbox, and some Embedded Windows versions. In terms of languages, only C++ is supported targeting an executable on one of the above-mentioned systems.

WebGL. A web integrated, JavaScript only low-level API version of OpenGL. It's developed as an open web standard and is implemented in most widely used browsers, without the need to install it in any way for the client or developer. It's a low-level API just like OpenGL and comes in two main versions based on OpenGL, 1.0 is based on the OpenGL ES 2.0 standard while 2.0 is based on OpenGL ES 3.0 which is less supported among older devices. As of writing, WebGL remains the lowest level of abstraction for running GPU code on the web.

Metal. Metal is a low-level rendering API designed by Apple for their devices. It has a very limited set of supported hardware and is further limited to only iOS and MacOS for the Operating System. It was created by apple to replace OpenGL on their hardware which was depreciated in 2018. In terms of language bindings, Objective-C and C++ are the only supported options.

2.2.4 Optional Abstractions for Rendering Solutions. The underlying rendering technologies mentioned in the last section, although

usable on their own, can be offloaded to be managed by a higher-level framework or engine. Although it must be stressed that the inherit properties of the underlying rendering solution still has an effect on the application being developed. The main reason to bring in an abstraction would usually be to lower development complexity, and in turn time. But the drawback often results in less flexibility regarding what can be made and lower performance.

Three.js. Three.js is a JavaScript library that abstracts WebGL to simplify the creation of 3D graphics. It is highly extensive and offers a wide array of useful functions and constructs to that end. It is available as a npm package and can be easily integrated into a web-based project. If comparing to WebGL though, which is already a part of the browser, three.js contributes to a much bigger application download footprint. It also suffers from the same general relationship mentioned at the start of this section, development is greatly simplified but a lot of the flexibility and performance is lost.

Unity. Unity, although first and foremost a game engine, can be applied to any development project where high performance graphics are needed. Unity supports DirectX, OpenGL, Vulkan and Metal- with only a few settings changes. A wide range of target devices can also then be built for. It is also possible to target WebGL as a platform in a similar fashion for web applications. Beyond this great cross-device and API support- Unity has an extensive collection of tools and APIs to streamline development. Although it should be mentioned that web applications are not as well supported, especially on mobile devices. The engine also only supports C# as the only option.

2.2.5 Languages. There are a large variety of potential languages that could be used for writing this program. The following were selected based on how extensively the languages are supported in previously mentioned Rendering solutions and Run Environment Sections. And some key exceptions that were considered to be important for consideration by the student.

C++. A well know general-purpose compiled programming language most often used in performance constraint applications. It is Object-Oriented but can be used to also write functional code. It is arguably the de-facto standard for systems programming.

Rust. Rust is a relatively new general-purpose compiled programming language often used in systems programming. It is a modern alternative to C++ that combines its high performance with a multitude of modern features such as a default package manager, and numerous safety features especially for memory (While still keeping a low performance impact).

Java. Java is a general-purpose, object-oriented focused programming language that compiles to Java Bytecode that can be run on any device running a JVM (Java Virtual Machine).

C#. C# is a general-purpose, multi-paradigm language developed by Microsoft. Although it is technically possible to compile to machine code, it is most often compiled to run on a .NET environment.

JavaScript. JavaScript is the natively supported programming language on the web. It is multi-paradigm and is just-in-time compiled. It requires a runtime which is often a browser or dedicated runtime environments such as Node.js

TypeScript. Typescript is a superset version of JavaScript developed by Microsoft that a number of features such as stricter syntax with the ability to have types. It transpiles to JavaScript which is then run as described above in 2.2.5.5.

3 SPECIFICATION

This section aims to inform the reader on the formal project plan that was created for the development of this projects, and the decisions on how it was structured.

3.1 Development Methodology

PXP as introduced in the research phase was selected for its well defined (extensively documented in paper) agile based methodology that was well suited to a single-person development team. Agile in general though, was selected over a waterfall technique to allow user feedback to drive design and development which was often collected. The details of the methodology are mainly based on this paper[] As is a fundamental principle of agile, this methodology was adapted to the project at hand with some main changes being:

- No automated testing setup at the start of the project. Graphics development is fairly tricky to automate the testing of as the large majority of testing is completely visual. It is still possible and was considered in section X but was chosen to set aside to rather focus on development and proving that the technology stack was suitable before any more work was committed on it.
- Allow refactoring to be raised at any point, also allow grouping of similar user stories to be refactored together. Allowed fixing problems before they became too big and interlinked.
- MoSCoW and Cost factors agile ceremonies were considered for the project's user stories. Allowed better planning and time management.
- Instead of tasks created from user stories, user stories themselves are treated like tasks with any non-user story work labelled as tasks instead and treated the same as stories. Done to lower duplication of information.
- Reworked development process to better fit the students workflow. See Section S

For a more in-depth look into how these changes were implemented and the overall development flow, see section X below

3.2 Requirements Gathering

With the goals that the developed application needs to achieve set as per the research phase- The next step was to create a concrete plan on what exact requirements / features would contribute towards achieving those goals. To do that, two main ideas were identified by the student.

The first idea was to conduct Interviews with individuals who commonly use visualization tools or do general data science work. This would provide key insight into what real users of visualization technology think is key for a successful application to have. These requirements would constitute the first phase of the project and set a strong start to either a waterfall type methodology or an agile project. But there were a couple of downsides that cancelled out this idea at that time:

- With no initial product to focus insight into actionable requirements- the interview process is more likely to return conflicting or infeasible requirements.
- It might be difficult to offer insight that is not generic for the same reasons. Application should plot data vs Application should do it like this instead of like this. With the latter insight being much more valued.
- Time and access to experts is very valuable and needs extensive preparation. It is important to make the most of it, which the student didn't feel like they could do at that time.

The next idea to identify requirements was a two-step process, and one which was inspired by the development methodology chosen to be followed by the student in section X. The student took on the role of the client to create a client brief using the insight gained during the research phase. This brief can be read in full in Appendix A but in short, creates a written source document describing the minimum simplest application that would need to be created. Further improvements would then come from user testing and analysis by the student. The main reason to do this instead of just skipping to creating user stories directly was to have a consistent main source from where user stories were pulled from and were focused on completing the brief's informal requirements.

With a brief set, the next step was to extract user stories (formal requirements) from the brief. Those user stories were all bundled up into an MVP (minimum viable product) feature set which was scheduled as further covered in section X. The specific user stories are all attached in Appendix A.

3.3 Project Management Tools

A number of tools and services were employed in the process to allow the organization and techniques mentioned to be applied. Those consisted of the following:

3.3.1 Git, Version Control. A version control tool is an important tool for controlling and recording changes to a codebase. Git specifically though was chosen due to the student having previous experience, and to allow GitHub to be used within the project, See subsubsection X

3.3.2 GitHub, Cloud Code Repositary. Github is a cloud host for git repositories with extensive tooling to support all aspects of a software development project.

3.4 Scheduling

With a backlog of user stories ready the next step was to combine similar stories into feature sets. Feature sets were then given a deadline on when all of their user stories should be completed. This would allow then the student to pick the most urgent Feature Set to focus on.

Once a Feature set was picked, each user story and task within was analyzed and given a development time cost and importance to the project. With this, a prioritized backlog was created that was split among the maximum iterations/sprints that would fit into the Feature set's time frame with a consideration for the students iteration velocity. On the off chance that there wasn't enough time to complete all stories and tasks, the student would re-analyze what stories and tasks could be dropped due to time constraints.
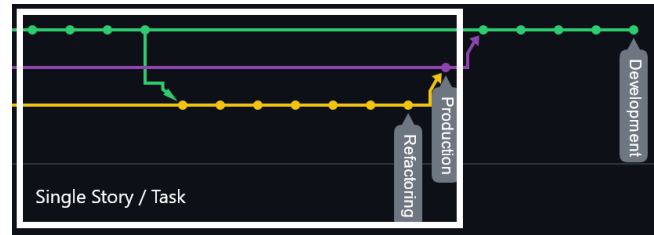

Figure 1: Development Timeline of a Single Story

Having feature sets allowed the student to put into perspective what are the major additions to the application and if there was enough time to fully develop those additions.

3.5 Development Flow

Development followed a predefined process where a user story or task was picked to be developed. Once the student has finished developing the one or more tasks picked in the simplest way possible, the student would then push the code to the refactor branch where it would be organized and possibly reworked to follow better practices. This process allowed the student to move fast and encounter obstacles in implementation much more quickly, which minimized the risks of large roadblocks knocking the schedule out of balance. But once the feature was reworked to a sufficient standard with bugs fixed, it was only then pushed to the final Production branch, which was publicly hosted. This process helped ensure that any code committed to production was vetted and would be unlikely to cause unexpected issues. It also allowed the student to always have a safe version of the application for demos and user testing without having to worry about what work was done before hand. The production version is then pushed to the development branch and the cycle repeats.

4 THE PLAN

This should outline what you have achieved so far in your project and what you still need to work on. You can present this in any relevant format.

4.1 Progress so far

What are the main things that you have achieved in your project so far? This may include research that you have done, major design and technology options you have explored, requirements gathering tasks, prototyping, testing, user-driven tasks, and so on. What are the main discoveries or stumbling blocks so far? Projects can vary of course in terms of the factors that are relevant to them. Convey what is relevant to your own project.

Be careful not to repeat your earlier background research. This should be written from the point of view of managing your project and not necessarily the specific features that your project will have.

4.2 Things still to do

What will you be doing for the remaining months of your project? Please provide a summary outline of the following, where applicable: technical milestones, evaluation milestones, testing milestones,

documentation milestones. You should also consider any major areas of risk that are remaining for your project and briefly describe any plans that you have for mitigating them.

5 REFLECTIONS

Use this section to reflect on how you feel things are going with your project so far. Are you where you want to be? Are you on-track for what you want to achieve (in terms of an outcome and/or a grade)? What have been the main challenges or learning curves for you so far? What achievements are you most happy with so far?

6 CONCLUSION

This section should summarise your report by highlighting key strengths and achievement.

6.1 Limitations

This section should outline any limitations that you have identified with your work so far. For example, have you only worked with a limited user demographic? What might be the impact of that? Can you mitigate this as the project progresses or not?

6.2 Generalisations and Future Work

This section should outline how you think your project may be used in the future. Consider it being used in different contexts with different user demographics. You may also have already identified some future work that would allow others to expand on your project in the future.

REFERENCES

A APPENDIX 1

Include any relevant appendices here (or delete if you want...)