**“Aegis – a macOS Package Manager and Tweak Installer”**

ethereus

Table of Contents

[Abstract 5](#_Toc165294999)

[INTRODUCTION 6](#_Toc165295000)

[Product Description 6](#_Toc165295001)

[Aim 6](#_Toc165295002)

[Objectives 6](#_Toc165295003)

[LITERATURE REVIEW 7](#_Toc165295004)

[The need for decentralization in software distribution 7](#_Toc165295005)

[Privacy concerns related to user account data 8](#_Toc165295006)

[A growing dissatisfaction 8](#_Toc165295007)

[Security concerns related to package managers 9](#_Toc165295008)

[Conclusion 10](#_Toc165295009)

[METHODOLOGY 11](#_Toc165295010)

[Software Development Process 11](#_Toc165295011)

[Software Development Methodology 11](#_Toc165295012)

[Requirements Engineering 12](#_Toc165295013)

[Requirements Elicitation 12](#_Toc165295014)

[Software Design 12](#_Toc165295015)

[Wireframing 12](#_Toc165295016)

[Software Construction 13](#_Toc165295017)

[Language and Libraries 13](#_Toc165295018)

[Graphical User Interface 13](#_Toc165295019)

[Software Testing 14](#_Toc165295020)

[Test Process 14](#_Toc165295021)

[Testing tools 14](#_Toc165295022)

[Professional, legal, ethical and social issues 15](#_Toc165295023)

[Professional 15](#_Toc165295024)

[Legal 15](#_Toc165295025)

[Ethical 16](#_Toc165295026)

[Social 17](#_Toc165295027)

[Time plan 17](#_Toc165295028)

[REQUIREMENTS 18](#_Toc165295029)

[System Specification 18](#_Toc165295030)

[Functional Requirements 18](#_Toc165295031)

[Home Screen 18](#_Toc165295032)

[Package Screens 19](#_Toc165295033)

[Sources Screen 19](#_Toc165295034)

[Manage Screen 19](#_Toc165295035)

[Update Screen 20](#_Toc165295036)

[Account Screen 20](#_Toc165295037)

[Software Qualities 21](#_Toc165295038)

[Safety 21](#_Toc165295039)

[Maintainability 21](#_Toc165295040)

[Efficiency 21](#_Toc165295041)

[SOFTWARE DESIGN 22](#_Toc165295042)

[Main function flowcharts 22](#_Toc165295043)

[Application Modelling 25](#_Toc165295044)

[Application Wireframing 25](#_Toc165295045)

[Software Testing Plan 27](#_Toc165295046)

[IMPLEMENTATION 27](#_Toc165295047)

[Sprint Planning 27](#_Toc165295048)

[Sprint 1 - Repository Implementation 27](#_Toc165295049)

[Sprint 2 - Package Information Functionality 27](#_Toc165295050)

[Sprint 3 - Download Functionality 28](#_Toc165295051)

[Sprint 4 - Manage Functionality 28](#_Toc165295052)

[Sprint 5 - Update Functionality 28](#_Toc165295053)

[Sprint 6 - Sources Functionality 29](#_Toc165295054)

[Sprint 7 - Security 29](#_Toc165295055)

[TESTING 30](#_Toc165295056)

[DISCUSSION: REVIEW AND REFLECTIONS 31](#_Toc165295057)

[Evaluation of the product 31](#_Toc165295058)

[Limitations 31](#_Toc165295059)

[Evaluation of the project & personal performance 31](#_Toc165295060)

[Summary and conclusions 32](#_Toc165295061)

[Future plans 32](#_Toc165295062)

[REFERENCES 33](#_Toc165295063)

[Appendix 34](#_Toc165295064)

[Supporting Documentation 34](#_Toc165295065)

[1.1 Product Specification Document 34](#_Toc165295066)

[1.2 Installation Guide 36](#_Toc165295067)

[1.3 Final repo file design 37](#_Toc165295068)

[1.4 Network Permission issue 37](#_Toc165295069)

[1.5 Network Permission Fix 37](#_Toc165295070)

[1.6 Local File Permission Issue 37](#_Toc165295071)

[1.7 Local File Permission Fix 38](#_Toc165295072)

[1.8 User Feedback 38](#_Toc165295073)

[1.9 Unit test examples 39](#_Toc165295074)

[1.10 Developer Guide 40](#_Toc165295075)

[1.11 Ethical Review Form 41](#_Toc165295076)

# Abstract

Aegis introduces a new approach to digital software distribution on macOS, presenting an alternative to the traditional App Store model. Positioned as a versatile digital distribution platform, Aegis empowers users and developers alike by offering a modern and decentralized solution. This project report delves into the design, development, and implications of Aegis.

At its core, Aegis redefines the relationship between users and developers on macOS by enabling direct software distribution from third-party servers. Developers can host their applications independently, allowing for greater flexibility and control over their software's deployment. Through Aegis, users gain access to a diverse range of software offerings, curated from various sources, expanding the ecosystem beyond the confines of a centralized marketplace.

Key to Aegis is its integration of system modifications, providing users with a proof-of-concept framework for installing and running modifications. This feature opens new possibilities for customization and innovation within the macOS environment.

Furthermore, Aegis incorporates a robust authentication system, enabling developers to validate software purchases using device-specific identifiers. By sharing a unique device ID, users establish a secure connection with developers, facilitating authorized software downloads and enhancing trust between parties. Additionally, Aegis implements file integrity checks through md5sum verification, ensuring the authenticity and integrity of downloaded files, bolstering security measures and safeguarding against malicious tampering.

Through comprehensive research, design, and implementation, this project explores the technical intricacies and user-centric considerations when making a digital distribution platform. By offering a decentralized approach, Aegis pioneers a new paradigm in macOS software delivery, fostering collaboration, innovation, and user empowerment within the ecosystem.

*Keywords:* macOS, digital distribution, decentralized, system modifications,

App Store, proof-of-concept

# INTRODUCTION

## Product Description

macOS software distribution is locked down and heavily restrictive, largely unsafe, and extremely complex and time consuming. The App Store offers users the ability to download quality applications safely but restricts the user’s choice to what is approved by Apple. Furthermore, the cost placed on developers to have their apps distributed through the store make software prices rise and often lead to developers distributing their software via download from their own website. This method of distributing via website download is highly unsafe as it allows fake sites to appear and trick the user into downloading malicious software instead of the desired application. The only current alternative is to download software through the macOS terminal, unrestricted and mostly safe but time consuming and far too complicated to set up for a regular user. This project aims to fills the gap created by current platforms by offering an alternative with no restrictions on software or costs placed onto developers, no complicated set up, the ability to install a variety of software packages, a secure and decentralized approach to verification, and the ability to pull software from third-party servers.

## Aim

**“**Offer a powerful decentralized platform for digital distribution on macOS”.

## Objectives

* To offer a viable and powerful alternative to the macOS AppStore.
* To provide users with a decentralized platform for installing and managing software.
* To offer flexibility and control by allowing users to choose between "Safe Mode" and "Full Mode" to disallow or allow adding third-party servers and system modifications.
* To provide a proof-of-concept for the distribution of system modifications on macOS.

# LITERATURE REVIEW

(Dixon, 2018) wrote about the growing importance of decentralization in the context of app stores, from the 30% tax on app revenue and apps being rejected or removed at will to the increased risk of security breaches and further privacy concerns, centralized platforms are abundant with problems. There is a growing awareness of these problems with existing distribution platforms which has led to many developers hosting their software on their own sites and users sourcing their applications from the web. Some users go as far as to jailbreak their devices (bypass the measures put in place by the device manufacturer) in order to install the software, they desire. There are currently no viable alternatives to these centralized distribution platforms for macOS, forcing users to choose between secure but restrictive stores and the unsafe but unrestricted option of searching the web for software.

Many people will use a mixture of distribution channels, such as the App Store to download free essential apps, the web to download more niche apps and tools such as archive utilities and storage cleaners, and the terminal to install development tools and packages. Due to the App Stores restrictive design, users cannot install anything that is not an application which rules out most of the available software for macOS, including things like command line tools for more advanced users. And even for the less technical users most applications either do not meet the App Store guidelines or the developers can’t afford the price of getting their app into the store forcing people to navigate to their website to download. If a decentralized distribution platform were to release, users would be able to source all of their required software from one place irrespective of type and gain the freedom of downloading from the web while maintaining the safety and simplicity of an app store.

## The need for decentralization in software distribution

Access to software is an extremely important topic in this project. Most people are aware of the downsides of centralized distribution without realising it, such as when a government forces companies like Google and Apple to remove software from their store for malicious reasons (Ververis et al., 2019). For example, when Russia and China force the removal of VPN applications from their stores, or certain social media apps that share information they disagree with. The ability for an external party to get involved and force the removal of software from a distribution platform is a major issue with centralized platforms, by having a single controlling authority, they can be targeted and forced to comply with demands. User expectations of a software distribution platform certainly include the ability to download software, so it should be a priority for a good platform to make software accessible instead of allowing it to be restricted.

## Privacy concerns related to user account data

There is an increasing realisation that a person’s data is often not as secure as they think it is, especially when it comes to user data shared with companies through account creation. (Dixon, 2018) mentioned an S curve that companies follow, where with time they shift from trying to cater to the user to make money from them, to using the user to make money from other companies. Selling data has become widely normalised and accepted, with companies promising to only share user data that has been anonymised with third parties to minimise concern, but this anonymised data can often still be used to target a user. In a study it was found that only 4 out of 120 tested subjects had no concern about their user data being shared by the company Facebook (Golbeck, 2016). After users were made aware of the privacy policy and what data was being shared overall concern was raised by 9.8% the study found. This study shows an increase in concern over privacy caused by knowledge of how data is shared by companies to third parties, and with time more people are becoming privacy aware and taking notice of these things which will lead to a big increase in overall concern for privacy in the coming years.

This paves the way for decentralized apps to take hold of the user market, and we are already seeing a rise in decentralized banking systems and message sharing protocols such as blockchain systems and ActivityPub/AT Protocol. We can expect this rise in interest and popularity to spread to other areas of technology in the coming months, and with a rise in decentralized software a decentralized distribution platform would easily find its place. This project aims to take advantage of the rising popularity in decentralization and work together with new software technologies and releases by facilitating the sharing of such software in a safe and user-friendly way.

## A growing dissatisfaction

(Lee & Soon, 2017) described the way that companies control their app stores as oppressive and state that they discourage customisation of their products. Customisation is important for a lot of people, especially when it comes to computers which is why there are desktop wallpapers and account photos. And by actively working against the user’s creativity and desire to customise their devices companies are creating a growing dissatisfaction with their products. Linux users continue to grow as the software gets better largely due to the restrictive nature of Windows and macOS, and users of macOS have an easy time migrating over since they are extremely similar operating systems. However, there is a lack of software support and user friendliness for non-technical users that keeps a lot of people stuck on macOS despite the freedoms offered by Linux.

This has led to an untapped market of users on macOS that wish to gain more freedom of choice and customisation on their devices. This project aims to provide a platform that would allow a user to easily find and install modifications such as themes or system behaviour changes and cater to this growing group of users dissatisfied with the restrictiveness of macOS. In that group of users there are some that take it upon themselves to find ways of customising their devices despite the lack of support for it, some will spend hours manually replacing icons for applications and folders or they might create a wallpaper that places a colour behind the menu bar to make it seem like it has changes colour. And inside of this group again is a group of technical users that create scripts and programs to automate these customisations that others wish they had on their own devices, and if there was an easy way for these developers to distribute their work to other users these sorts of programs would be highly popular.

## Security concerns related to package managers

Despite the praises so far of a decentralized software distribution platform, there are some big security concerns around their downloading functionality especially with the third-party servers hosting software. (Cappos et al., 2008) raised concerns about a man-in-the-middle attack on package managers stating despite security implementations all package managers they have studied are vulnerable to such an attack. This project for example will use md5 hashes to verify file integrity for software downloads, showing that the file has not been modified in any way during transmission. However, as stated a man-in-the-middle attack can defeat file integrity checks by intercepting the network traffic and supplying both a modified piece of software and by swapping out the md5sum to be used for verification after download. It is for this reason that this project implements an official repository that the user can download from, software on the official repository is verified to be safe and the server is well secured. All network traffic between the client and the official repository is encrypted using SSL certificates making the chance of a man-in-the-middle attack happening extremely unlikely. Though due to the open nature of the platform it cannot force third-party repositories to implement the same security measures, leaving a possibility for an insecure file transfer to take place and come under an attack.

To address that problem, while this project won’t force a third-party to follow specific security measures to share their packages on the platform, it can reduce the risk for non-technical users that might not tell the difference between a secure and insecure server by implementing user modes. By default, only the official repository will be available and system modifications will be disabled, the user can choose to turn off these protections to access software not distributed through the official repository.

When it comes to third-party repositories, the user is responsible for only adding and downloading from sources they trust. (Rubin, 1995) pointed out that a person with malicious intent that has access to a server hosting software could very easily replace the software on that server with something else. With access to a secure server hosting a package repository someone could change all the package information including the md5 hash effectively bypassing all security measures that could be put into place. The package manager and user would have no idea as the server would show a secure connection and the malicious software would also pass the integrity check. This issue opens the door to people creating fake repositories of software and tricking others into downloading from their server, so it is important to make the user aware of the risks of third-party sources and have them disabled by default.

## Conclusion

In this review, several problem areas have been discussed that the product will aim to solve, namely the restricted nature of macOS and the security concerns surrounding big organisations being the central authority of distribution platforms.

The most important problem area reviewed is the security concerns related to other package managers and the ways in which these can be mitigated. There is a fine line between security measure and restrictive feature when it comes to developing a distribution platform, it would be a security measure to block third-party servers that do not implement SSL certificates, however that would also be a restrictive feature as it raises the complexity for servers to distribute software.

# METHODOLOGY

## Software Development Process

### Software Development Methodology

The waterfall methodology is a linear and sequential approach to software development. It consists of 5 distinct phases: requirements gathering, design, implementation, testing, deployment, and maintenance. Where you must complete each stage before moving to the next making makes it unsuitable for development of a product that could change during development based on new findings and feedback. This method would be suitable for projects with well-defined requirements that are not subject to change. However, for this project it is unsuitable as it would make it difficult to respond to any user testing feedback or issues during development.

Agile is another methodology that is iterative and flexible when it comes to development, emphasizing collaboration, adaptability, and customer feedback. Scrum falls under this methodology which breaks the project down into smaller pieces called sprints, with each delivering working and viable code to be tested and reviewed. The Agile methodology prioritizes the user experience and input over strictly defined processes and phases, much better suited for application development than the waterfall methodology. Continuous feedback after each sprint allows for adjustments throughout the development process, allowing for changes in the requirements and features according to user views and opinions.

Rapid Application Development (RAD) is a methodology that prioritizes rapid prototyping and iterative development cycles. It involves quickly creating prototypes which are continuously tested throughout development and constantly changed based on feedback. RAD requires close communication between developers and users which helps to make sure the software meets expectations and is a great methodology to follow for projects with an extremely short development time. This methodology is suited to projects where requirements are subject to change, and speed of development is an essential factor. However, due to a lack of constant communication with the test users along with the problems that could come from a single developer having to constantly change their code and features it is not suitable for this project.

For this project the Agile methodology is chosen as it offers significant advantages even for solo developers. As an individual, employing agile principles allows for adaptability, efficiency, and continuous improvement throughout the development process without throwing the developer off with changes mid feature development. Breaking down the project into manageable tasks such as each screen and major feature and maintaining flexibility in responding to feedback after each sprint enables rapid progress and timely adjustments of the product. Additionally, agile practices like test-driven development and continuous integration enhance code quality and minimize issues while helping to stick close to the project goals.

### Requirements Engineering

The requirements of this product will be formed from multiple sources, such as observation of similar products and user feedback at the end of each sprint. As a functional proof-of-concept this project does not need to meet all of the requirements of a complete and ready to ship application, instead simply needing to be functional and able to demonstrate a solution to the given problems. This allows for saving time when it comes to implementation of features as a proof-of-concept does not need to include all of the in-app text like package descriptions and real downloadable packages, and can instead use test data for the purpose of showing functionality.

### Requirements Elicitation

Through observation of similar products, the requirements for this software will be gathered and ranked in order of priority according to what a user would expect of such software. This method of creating a list of requirements is highly efficient and is possible thanks to the problem area including competitor software that can be examined. This ensures that the product meets the existing expectations of users for software of its type, so that it is not trading existing functionality enjoyed by users in other products for the new unique features of this one. A situation where a user has to choose between one set of features and another is exactly what caused the problem area this product aims to solve in the first place and should be avoided at all costs.

## Software Design

### Wireframing

Wireframing is the creation of non-functional interfaces for the purpose of demonstrating the functionality of the interface to the user, as well as allowing for early feedback on the GUI of the software. It allows the developer to create an interface design based on requirements and features of the software and form a solid idea of how they will come together in the product. There are 3 main levels of wireframes being low, medium and high.

A medium fidelity wireframe will be chosen for this project as it allows the user to get a concrete view of how the product will look and function without locking in designs for the individual element such as how the packages will look on the screen as this would not yet be apparent without first creating the system by which they are displayed.

## Software Construction

### Language and Libraries

For macOS application creation, Swift and Objective-C are the two main options for languages, Swift being the native language for macOS apps and designed for use on Apple platforms. Swift is also regarded as being up to 2.5 times faster than Objective-C and includes better memory management as Objective-C requires manual memory management. This project will choose swift as the language for which it will compile and run on.

However, due to a lack of experience in working with Swift and trouble during experimentation with libraries and using Xcode to develop and build, the project will not be written in Swift. It will still compile and run on Swift though, which is made possible by a framework called Flutter. Flutter allows a developer to write application code in Dart programming language and have that code compile into native code for the target operating system in this case into Swift. This framework is extremely powerful given that Dart is an easy to learn language with fast execution, built-in support for asynchronous programming and a wide availability of libraries. Flutter itself is a highly impressive framework given that code is compiled into native languages making for smooth animations and high performance on device along with great memory management. The main benefits of using Dart over Swift are the faster development times from the easier language and the simple UI Widgets and Material Design components along with its inbuilt ability to perform a “hot reload” where it will refresh the content of the application while it is running during debugging to avoid the need to rebuild the app every time a code change is made.

### Graphical User Interface

The graphical user interface forms a significant part of the product as it is what the user actually sees and interacts with in order to use the platforms features. Due to writing in Dart language, the user interface will be created using Flutter UI Widgets which gives a native look and feel to the application and ensures efficient code and fast execution in app. Flutter makes use of “screens” to display content on and has a great and simple to use way of handling screen transitions by calling the desired screen into the main space like a function and passing in any arguments required from the previous screen. This makes tasks such as loading data with a button on one screen and displaying it in an information field on a new separate screen easy to implement, and even with a lack of experience in working with screens it is easy to pick up once you start using them and experimenting.

### Software Testing

Testing is an essential part of any products development, it helps to ensure the code is functional as expected and that it meets the requirements set out at the start of development.

There are a variety of accepted software testing techniques that can be used during and after each development sprint, and these can take the form of manual tests performed by the developer and test users or automated tests performed using the range of available software testing tools.

The three main types of testing that will be employed in this project are unit testing, integration testing and acceptance testing. In this project test users will be responsible for acceptance testing where they are encouraged to try and input bad data into the application to make it throw an error or behave in an unintended way.

The developer will be responsible for unit testing where a range of automated tests will be performed on each function to make sure that they function as expected. And also integration testing where the different functions will be tested together to make sure that they work as expected with each other as designed.

### Test Process

The application will make use of unit tests and integration tests where possible to ensure the functionality of the app is as expected and the main functions return the data that they are supposed to.

For manual tests carried out by test users and the developer, the results will be stored in a test table to make the contents of the test as well as the result easy to see and refer to. The format of the test table is as below in Table 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **REQUIREMENT** | **DESCRIPTION** | **TEST** | **PASS/FAIL** | **NOTES** |
|  |  |  |  |  |

*Table 1 – Test Table.*

### Testing tools

For unit testing Flutter includes a useful tool called “test” that allows you to perform unit testing on the various functions in your code by simply defining the test and running “flutter test”.

## Professional, legal, ethical and social issues

### Professional

Developing an application involves a series of professional issues, each critical to the app's success and legal compliance. One major concern is the quality of the application. Ensuring high quality is not only a matter of user satisfaction but also a reflection of the developer's professionalism. Neglecting quality control can lead to user dissatisfaction, negative reviews, and even legal repercussions if the app fails to perform as promised or causes harm due to technical glitches.

Especially when developing an app that includes functionality such as downloading from unverified sources and installing system modifications, it is extremely important to make the user aware of the risks and have them choose to enable said functionality so that the developer is not liable for any damages caused by misuse.

Copyright for used media is another significant issue. Utilizing images without proper authorization can result in copyright infringement, leading to legal disputes and financial penalties. Developers must either create their own images, use royalty-free images, or obtain explicit permission from the copyright holders to use their images in the application.

Moreover, due care must be exercised throughout the development process. This includes adhering to industry standards, implementing best practices in coding and security, and conducting thorough testing to identify and rectify any potential vulnerabilities or weaknesses. Failing to exercise due care can result in security breaches, data loss, and damage to the developer's reputation.

In summary, addressing professional issues such as app quality, copyright compliance for media and software, and exercising due care are indispensable for the successful development and deployment of an application while safeguarding against legal and reputational risks.

### Legal

When developing an application, legal issues become even more critical, especially when the app involves downloading from unverified sources and facilitating the installation of system-modifying code. One primary concern in this context is the potential for facilitating illegal activities or infringing on intellectual property rights.

Allowing downloads from unverified sources can expose users to various risks, including malware, pirated content, and security breaches. Developers must ensure compliance with relevant laws and regulations governing app distribution.

Facilitating the installation of system-modifying code raises additional legal considerations. Depending on the nature of the modifications and their impact on the device or system, developers may need to obtain explicit user consent, adhere to platform-specific regulations, and comply with laws related to cybersecurity, privacy, and consumer protection.

Furthermore, developers must be vigilant about potential legal liabilities arising from the app's functionality, such as copyright infringement, violation of terms of service agreements, or enabling illegal activities like piracy or hacking. Implementing robust terms of use and privacy policies, along with thorough user education about risks and responsibilities, can help mitigate legal exposure.

In conclusion, addressing legal issues related to downloading from unverified sources and facilitating system modifications is crucial for developers to avoid legal pitfalls, protect users' rights and security, and maintain compliance with relevant laws and regulations.

### Ethical

Ethical considerations are paramount in the development of any application, especially one of this type. One primary ethical concern is the potential for harm to users and third parties.

Allowing downloads from unverified sources can expose users to risks such as malware, phishing attacks, and privacy breaches. Developers have an ethical responsibility to prioritize user safety and well-being by implementing robust security measures, including thorough vetting of sources and encryption of data transmission.

Facilitating the installation of system-modifying code raises ethical dilemmas regarding user autonomy and the potential for unintended consequences. Developers must consider the implications of their actions on users' devices, privacy, and security, as well as the broader ecosystem. They should provide clear and transparent information to users about the purpose and risks of installing such code, along with mechanisms for informed consent and opt-out options. Moreover, developers should uphold principles of accountability, transparency, and user empowerment throughout the development process.

In summary, addressing ethical issues in the development of an application involving downloads from unverified sources and system modifications requires a holistic approach that prioritizes user safety while upholding principles of transparency, and accountability.

### Social

Developing an application with features such as in this project presents social issues that must be carefully considered.

One significant concern is the potential impact on online safety. Allowing downloads from unverified sources can contribute to the spread of malicious software, undermining users' ability to discern trustworthy sources and navigate the digital landscape safely. Developers have a social responsibility to promote digital literacy and provide tools and resources to help users critically evaluate online sources.

Facilitating the installation of system-modifying code can also have broader societal implications, particularly regarding privacy and security. Users may unwittingly expose themselves to surveillance, data breaches, and identity theft by installing code that compromises their device's integrity. Developers must prioritize user privacy and security by implementing robust encryption, anonymization, and data protection measures. This will be done in Aegis through file integrity checks, an anonymised account system and an encryption protected official source.

In summary, addressing social issues in the development of an application involving downloads from unverified sources and system modifications requires a nuanced understanding of the broader societal implications and a commitment to promoting digital literacy, privacy, and safety. Developers must work collaboratively with stakeholders to ensure that their apps contribute positively to the digital ecosystem and empower users to navigate the online world safely and responsibly.

## Time plan

For the duration of the projects development a monthly scrum schedule will be followed, where the first week is spent reviewing previous code, the second week spent implementing a new feature, the third week is for testing the new feature and its integration into the app and the fourth is for user testing and any necessary write-up.

# REQUIREMENTS

## System Specification

Aegis should be a comprehensive software solution designed to simplify the management and installation of packages on macOS and be a viable alternative to the AppStore. Aegis should have a user-friendly interface coupled with powerful functionality such as the ability to install packages from a variety of sources. The solution should allow the user to download and manage a variety of software from an official source under “safe” mode and further allow the installation and managing of system modifications and downloads from third parties under “full” mode. The project forms a collection of proof-of-concept systems being the package manager itself, the repository system which together with the manager form the digital distribution platform, and the framework for code injection based system modifications. There was a system specification document created which can be found in the appendix section 1.1.

## Functional Requirements

### Home Screen

|  |  |  |  |
| --- | --- | --- | --- |
|  | **REQUIREMENT** | **USER STORY** | **PRIORITY** |
| 1 | See featured packages | “I want to be able to see featured what packages are popular” | Required |
| 2 | See featured news/updates | “I want to keep up with any updates and news surrounding Aegis” | Required |
| 3 | See descriptions of featured packages | “I want the ability to see descriptions for software packages before downloading” | Required |
| 4 | Provide a menu to access other screens | “I want a simple menu that lets me navigate easily” | Required |
| 5 | Welcome the user to the package manager | “I want the home screen to be welcoming” | Required |

*Table 1 – Home Screen functional requirements.*

### Package Screens

|  |  |  |  |
| --- | --- | --- | --- |
|  | **REQUIREMENT** | **USER STORY** | **PRIORITY** |
| 1 | See packages | “I want to be able to see what packages are available” | Required |
| 2 | See descriptions of packages | “I want the ability to see descriptions for software packages before downloading” | Required |
| 3 | Provide a button to go back home | “I want to be able to easily navigate back to the home screen” | Required |
| 4 | Provide a button to download package | “I want to be able to easily download a package” | Required |

*Table 2 – Package Screen functional requirements.*

### Sources Screen

|  |  |  |  |
| --- | --- | --- | --- |
|  | **REQUIREMENT** | **USER STORY** | **PRIORITY** |
| 1 | See list of repositories | “I want to be able to see which sources I have currently added” | Required |
| 2 | See packages available in repository | “I want to be able to see what packages are available” | Required |
| 3 | Provide a button to go back home | “I want to be able to easily navigate back to the home screen” | Required |
| 4 | Provide a button to remove repository | “I want to be able to easily remove a source” | Required |

*Table 3 – Sources Screen functional requirements.*

### Manage Screen

|  |  |  |  |
| --- | --- | --- | --- |
|  | **REQUIREMENT** | **USER STORY** | **PRIORITY** |
| 1 | See list of installed packages | “I want to be able to see which packages I have currently installed” | Required |
| 2 | Provide a button to remove package | “I want to be able to easily remove a package” | Required |
| 3 | Provide a button to go back home | “I want to be able to easily navigate back to the home screen” | Required |

*Table 4 – Manage Screen functional requirements.*

### Update Screen

|  |  |  |  |
| --- | --- | --- | --- |
|  | **REQUIREMENT** | **USER STORY** | **PRIORITY** |
| 1 | See list of packages with available updates | “I want to be able to see which installed packages have newer versions available” | Required |
| 2 | Provide a button to update package | “I want to be able to easily update a package” | Required |
| 3 | Provide a button to go back home | “I want to be able to easily navigate back to the home screen” | Required |

*Table 5 – Update Screen functional requirements.*

### Account Screen

|  |  |  |  |
| --- | --- | --- | --- |
|  | **REQUIREMENT** | **USER STORY** | **PRIORITY** |
| 1 | Find device unique ID | “I want to be able to share my ID with developers to prove ownership of purchased software” | Required |
| 2 | Allow user to change between modes (safe/full) | “I want to be able to easily change between the modes” | Required |
| 3 | Provide a button to go back home | “I want to be able to easily navigate back to the home screen” | Required |

*Table 6 – Account Screen functional requirements.*

## Software Qualities

### Safety

To ensure the platform is safe, there should be multiple modes with the default state of the app when it is first installed being in a safe mode. In this mode, users cannot download system modifications or add third-party sources and a message will notify them of this on screen. They can only download verified safe software from the official source which is encrypted with SSL certificates and all downloads verified with a file integrity check. There should be a warning given to users about the potential dangers of leaving safe mode, an example of which can be found in the user installation guide in the appendix section 1.2.

### Maintainability

The platform should be adequately documented, through official write-ups and also code commenting. Manual testing should be performed after new functions are added and any concerns shown in the debugging console should be addressed before the public release of a new version.

### Efficiency

All code should be compiled into native Swift code, and the software should not exhibit any freezes, crashes or overall slow behaviour during its usage. Code should be written in such a way that it is as optimised as possible with both file size and memory usage kept in mind during development.

# SOFTWARE DESIGN

## Main function flowcharts

A diagram of a software system

Description automatically generated

*Figure 1 – Function logic to check for available updates.*

A diagram of a software process

Description automatically generated

*Figure 2 – Function logic to install a package.*

A diagram of a software system

Description automatically generated

*Figure 3 – Function logic for the update screen.*

## Application Modelling

### Application Wireframing

A screenshot of a web page

Description automatically generated

*Figure 4 – Home Screen Wireframe. The two larger boxes are for update posts, the smaller ones are placeholders for featured packages.*

A screenshot of a package

Description automatically generated

*Figure 5 – Package Screen Wireframe. A screen design for all screens that show packages.*

A screenshot of a box

Description automatically generated

*Figure 6 – Manage Screen Wireframe. Lists installed packages with an uninstall button next to them.*

A screenshot of a computer

Description automatically generated

*Figure 6 – Sources Screen Wireframe. Lists added sources, the user can click on a source to view its packages, the user can also enter the url of a new source into the top bar and press the plus button to add it.*

## Software Testing Plan

This project will be tested using a variety of methods to ensure quality, such as unit testing, test user manual testing, acceptance testing, debugging and integration testing.

# IMPLEMENTATION

## Sprint Planning

As previously stated, this project will follow a monthly sprint run based on the scrum ideology. The project was split into key parts being Repository Implementation, Package Information Functionality, Download Functionality, Manage Functionality, Update Functionality, Sources Functionality, Security. Working out to one sprint per each of the 7 months of the project timescale. Following the requirements specification seen in appendix 1.1 development would continue steadily throughout the duration of the project. The home screen and GUI were worked on in between sprints since they are not considered major features.

## Sprint 1 - Repository Implementation

The first step was to decide what package information to store on the server that Aegis would need in order to function, and a format for this information was settled on as seen in appendix 1.3 showing the repository file design. The package manager would make use of “.json” files and their key and information structure to easily store, load and work with the necessary package data.

This was decided on after researching the information that DPKG (Debian Packager) utility stores about its packages.

The server was then configured to host a web page and a folder created to hold the repository files for Aegis, and some sample files were created and their information stored in the “test.json” file that Aegis would read to load the servers package content information.

No issues were encountered in this sprint and testing was not carried apart from checking to see if the site was live.

## Sprint 2 - Package Information Functionality

The next sprint involved creating the system for loading information from a repository, reading the server contents file and understanding the contents.

While implementing the http request functionality to pull information from the file the first development issue was encountered, (appendix 1.4). And eventually a fix was discovered as seen in appendix 1.5. After this issue was solved, the data was able to be loaded successfully and stored into a map to preserve its structure.

## Sprint 3 - Download Functionality

The next sprint involved creating the system for downloading packages from a repository, reading the server contents file to gain the files download location and other package information that would need to be stored locally.

While implementing the download functionality, specifically the part where the file is saved to the users’ device, another issue was encountered (appendix 1.6). The fix to this was similar in nature to the previous issue as seen in appendix 1.7. After this issue was solved, downloaded packages could now be saved to the users device.

## Sprint 4 - Manage Functionality

This next sprint was focused on the manage screen and the ability to view and removed installed software, at this point the Account Screen was added to the program along with a button to remove all system modifications.

The main part of this sprint was the creation of the Manage Screen which loads the local package information and displays a list of installed packages with an uninstall button next to each one.

It was during this sprint that a change had to be made universally to classes in the app, the move from stateless classes to stateful classes in flutter. This made the passing of data between classes possible which allows for the uninstall button to pass the information of the package to be removed onto an uninstall function.

## Sprint 5 - Update Functionality

Sprint 5 would complete the basic functionality for a digital distribution platform, the ability to install, uninstall, view and update packages.

This functionality follows the previously mentioned logic design and makes use of the package information fetching functionality from Sprint 2 and the download functionality from Sprint 3. By loading the data of installed packages, and then cross checking their version number against the version number of the same package on the origin server, the function can create a list of packages that have updates available.

It then calls the download function for each package in the list to update the software and its local information on the user’s device.

This also came with the implementation of the Update Screen that shows the list of packages with available updates and a button next to them to initiate the update.

## Sprint 6 - Sources Functionality

In Sprint 6 support for third-party sources of software was developed, up until this point all packages were loaded from the official repository.

This included the creation of the Sources Screen that displays a list of added repositories and lets the user select them to view the package contents of that server. It also provides a url bar and add button at the top for users to add additional sources of apps to Aegis.

The development behind these features was relatively simple, allow the user to input a source url, add that to a local file and then load that file to display the list of installed sources. When a source in the list is pressed it opens a new package view screen showing the packages available for download from that server as well as a button to remove the source from the local list.

This was all implemented using existing functions such as the fetch data function and the functionality to display packages on the screen.

## Sprint 7 - Security

In the final sprint the package manager software was polished up with some final GUI work and it was time to consider the security aspect of the system. As mentioned in the literature review package managers are vulnerable to security threats including man-in-the-middle attacks and therefore place the user at risk by default. To prevent this user modes were added to the application where the initial state of the package manager after download was “safe” mode.

The Account Screen was modified to have a menu to change modes between “safe” and “full” and after user feedback descriptions were added to each button to describe their function.

The “safe” mode was made to automatically disable and remove all system modifications and block the ability to download any further system modifications once enabled, as well as to block access to third-party sources. This was intended as an ‘emergency stop’ of sorts for the user to activate should they experience unexpected behaviour on their device after installing a system modification software package.

Aegis when first installed is in “safe” mode to protect non-technical users from the advanced and potentially dangerous software outside of the official repository. The official repository also gained SSL encryption to secure downloads and work to prevent man-in-the-middle attacks.

As a further security measure all downloads ended in a file integrity check being performed by comparing the md5sum of downloaded files to the value of the file on the server to verify that the user did in fact download the correct file and that it was not tampered with.

And as a final security feature while keeping in mind the previously discussed need for decentralization, was to ditch user accounts in favour of a device unique ID, generated by taking the devices MAC address and hashing it using sha256 algorithm. If a developer wanted to charge for their software through Aegis, it is their responsibility to implement a payment system on their repository site, it was decided that Aegis would not play any part in taking or facilitating payments due to the extensive privacy and security considerations that come with such a system.

The device unique ID presents a way for developers to check if a device has purchased a piece of software before allowing the web connection to download that software.

By default, though Aegis is a free and open digital distribution platform and the official repository is not intended to ever have paid software hosted on it.

At the end of this final sprint the experimental support for system modifications was implemented in the form of a process that checks for installed modifications and starts them.

# TESTING

Manual testing was carried out throughout the duration of the project, and the useful flutter debugger was used to catch issues early on such as those mentioned previously.

User acceptance testing revealed certain issues such as the entering of an invalid repository url did not give any notice and the url would be added to the sources list regardless of if it was a valid Aegis source or not.

While testing the software in person with a test user it was also revealed that when an md5sum does not match between the downloaded file and what the server states it should be, Aegis did not notify the user but just silently removed the software, leaving the user questioning if the download button was broken.

Feedback from user testing throughout the project duration can be seen in appendix section 1.8.

Some unit testing on the key functions was carried out according to the tests seen in appendix section 1.9.

# DISCUSSION: REVIEW AND REFLECTIONS

## Evaluation of the product

During the development sprints all major requirements which were initially requested have been implemented. All major functionality is present, and Aegis works as expected as a decentralized package manager/digital distribution platform for macOS with proof-of-concept system modification support. Code injection differs from the initial project proposal in that it uses a different code injection technique, the technique “DYLD\_INSERT\_LIBRARIES” has been chosen instead of the mac injection platform due to this technique supporting all modern versions of macOS increasing availability for users.

## Limitations

While the development of this project has gone mostly well, Aegis cannot provide a system to securely download from third-party servers and so is limited to using different modes to protect users from malicious software. Furthermore, the proof-of-concept code injection framework is limited in its current functionality. With more knowledge and time, the project could be expanded to include full support for system modifications.

Another limitation is that Aegis has been developed by one person while only having two test users and a limited time at the end to complete testing. This has led to some issues getting past the development and testing phase and into “final release”.

A major limitation of this project is the lack of research material available about decentralized app distribution, it proved difficult to find reliable sources to talk about the subject.

A final limitation encountered was availability to work on the project along with the availability of the test users to review the product. This severely limited the report content in certain areas as well as the ability to complete the report to a high standard.

## Evaluation of the project & personal performance

In the end the project failed to meet its deadline, the time management part of this project was severely lacking and too much time was spent on development of the product as opposed to the write-up. Time management was lacking, partly due to events and situations but mainly due to poor planning.

As far as meeting the other aims of the project though, it has largely been a success with almost all planned features implemented and bug free. The GUI is intuitive and user-friendly, and the more technically advanced sections of the product are kept out of the way of non-technical users unless they choose to enable them.

Due to the nature of sprint runs, a lot of issues encountered were quickly fixed and forgotten without including them in the report, also lacking in documentation is the tests and reviews done by the two test users.

Looking back to the research and design phase of the project, more time could have been spent early on trying to find more sources about decentralized software and package managers. And when in the design phase, it would have been helpful to spend some time creating more advanced diagrams such as use case diagrams to show the user journey through the app.

Understanding and confidence writing desktop applications and working with servers has increased greatly through this project along with the ability to create useable GUI applications moving away from the usual command-line interfaces.

## Summary and conclusions

The aim of the project was to **“**Offer a powerful decentralized platform for digital distribution on macOS”. Through research and development carried out over the project’s duration and through error handling and user testing Aegis has met the goals set out in the project specification document. Aegis is a simple to use alternative to the macOS App Store, packing some advanced features and catering to both technical and non-technical users alike.

It simplifies and unifies the methods of software acquisition on macOS and provides an experimental framework for running system modifications (tweaks).

The project has proven successful in its aims of researching and developing a working product and could possibly be the first of its kind as a modern decentralized App Store alternative on macOS.

The project contains a few bugs that have made it past the production phase but is in a release ready form as an experimental software and proof-of-concept. Towards that end, included in the Project is an installation guide and clear instructions for developers wanting to host their software for Aegis to load (Appendix 1.10).

By the end of the project Aegis has evolved from an idea about a decentralized distribution platform into a collection of tools, forming a framework for open software distribution and the running of advanced system modifications on a platform that is traditionally locked down and tightly controlled.

## Future plans

Future plans for the project include bug fixes and general quality improvements such as more work being put into the GUI of the application. More research and development time spent on the code-injection based system modifications would be beneficial also, as any expansion of this advanced feature would bring interest in the project and with it more users.

It would be interesting to delve further into decentralization and to use decentralized payment technology such as blockchains to craft a purchasing framework for developers to make use of should they wish to charge for software. Leaving the developers responsible for implementing a payment system based around unique ID’s is secure and removes all liability from Aegis, therefore it works great as an initial solution.

However, as an alternative to the App Store not having a unified method of payment for paid software damages its appeal to a normal user that values simplicity and ease of use.

# REFERENCES

**Dixon, C. (2018, February 18). Why Decentralization Matters. Medium.** [**https://onezero.medium.com/why-decentralization-matters-5e3f79f7638e**](https://onezero.medium.com/why-decentralization-matters-5e3f79f7638e)

Ververis, V., Isaakidis, M., Weber, V., & Fabian, B. (2019). Shedding Light on Mobile App Store Censorship. 27th Conference on User Modeling, Adaptation and Personalization, Larnaca, Cyprus, 193–198. <https://doi.org/https://doi.org/10.1145/3314183.3324965>

Golbeck, J. (2016). User Perception of Facebook App Data Access: A Comparison of Methods and Privacy Concerns. Future Internet, 8(2), 9. <https://doi.org/https://doi.org/10.3390/fi8020009>

Cappos, J., Samuel, J., Baker, S., & Hartman, J. H. (2008). A look in the mirror: attacks on package managers. Proceedings of the 15th ACM conference on Computer and communications security, Alexandria, Virginia, USA, CCS '08, 565–574. <https://doi.org/https://doi.org/10.1145/1455770.1455841>

Rubin, A. D. (1995). Trusted distribution of software over the Internet. Proceedings of the Symposium on Network and Distributed System Security, San Diego, CA, USA, 47-53. https://doi.org/https://doi.org/10.1109/NDSS.1995.390646

Lee, M. S. W., & Soon, I. (2017). Taking a bite out of Apple: Jailbreaking and the confluence of brand loyalty, consumer resistance and the co-creation of value. Journal of Product & Brand Management, 26(4), 351-364. https://doi.org/https://doi.org/10.1108/JPBM-11-2015-1045

# Appendix

## Supporting Documentation

### 1.1 Product Specification Document

Product Specification Document

Project Name: Aegis

Version: 2.0

1. Outline

A comprehensive software solution designed to simplify the management and installation of packages on macOS and be a viable alternative to the AppStore. Developed with the needs of both novice and advanced users in mind, Aegis offers a user-friendly interface coupled with powerful functionality. This document outlines the key features, functionalities, and requirements of Aegis.

2. Objectives

* To offer a viable and powerful alternative to the macOS AppStore.
* To provide users with a decentralized platform for installing and managing software.
* To offer flexibility and control by allowing users to choose between "Safe Mode" and "Full Mode" to disallow or allow adding third-party servers and system modifications.
* To provide a proof-of-concept for the distribution of system modifications on macOS.

3. Features

Aegis will include the following features:

Home Page: To display a curated selection of top packages, to showcase the most popular and useful offerings from the official Aegis server, as well as a news feed.

Package Pages: Dedicated pages for different types of packages, to allow users to browse and select packages based on their specific needs.

Update Page: To check installed package versions against their versions on the servers they were downloaded from and let the user update them easily.

Third-party Source Management: To let users to add, remove, and manage third-party sources that host packages, expanding the available selection.

Safe Mode and Full Mode: To give users the option to toggle between "Safe Mode," where third-party sources and system modifications are disabled, and "Full Mode," where all features are enabled for advanced users.

4. User Requirements

Throughout the development process, feedback will be gathered from the three test users to ensure that Aegis meets the following user requirements:

Intuitive interface: Users should be able to navigate the application easily and intuitively, even without prior experience with package management software.

Stability: The application should be stable and reliable, minimizing crashes or errors during use.

Flexibility: Users should have the flexibility to customize their experience, including adding custom servers and choosing between different installation modes.

Security: The application should prioritize user security by offering a "Safe Mode" option, adhering to best practices for package installation and modification and protecting user information.

5. Technical Requirements

Aegis will be built using the following technologies:

Programming Language: Dart

Framework: Flutter

Operating System Compatibility: macOS

### 1.2 Installation Guide

A screenshot of a computer application

Description automatically generated

*Image 1, Installation Guide Document*

### 1.3 Final repo file design

Example “repo.json” file:

[

{

"name": "Example Package",

"image": "https://example.com /repo/icon.png",

"url": "https://example.com/repo/package\_file",

"description": "An example of a software package distributed through Aegis.",

"version": "0.1",

"md5": "03b36gh37f91h2fh37f7578hfi465d5"

},

]

*Image 1, Installation Guide Document*

### 1.4 Network Permission issue

Error fetching app data: ClientException with SocketException: Connection failed (OS Error: Operation not permitted, errno = 1)

### 1.5 Network Permission Fix

In order to fix the network permission it was necessary to change the network permission to true for the app:

<key>com.apple.security.network.client</key>

<true/>

### 1.6 Local File Permission Issue

MissingPluginException(No implementation found for method getApplicationDirectory on channel com.example.aegisFlutter/get\_directory)

### 1.7 Local File Permission Fix

In order to fix the documents permission it was necessary to disable the applications sandbox:

<key>com.apple.security.app-sandbox</key>

<false/>

### 1.8 User Feedback

**User testing feedback from early development phase**

“Separate the packages loaded from third-party sources into their own screens, otherwise it gets confusing where an app is coming from”

“Give packages descriptions and let the user click on the app icons to view the description like in the App Store”

“Provide with the release a setup guide for third-party sources”

**User testing feedback from mid development phase**

“Give a description in the app of what the modes do”

“User accounts are a bad idea, an open store solution should aim to be decentralised, user accounts require a central authority”

“Show on the screen when an app is installed after clicking “get” the button should change to “installed” like in other stores”

**User testing feedback from end of development**

“Overall a bit complicated to use with the different modes and features, but easy to navigate and to install/uninstall/update apps”

“The home page with featured apps and news is good, the UI isn’t super modern but it doesn’t look bad”

Bug: “when uninstalling a package, all screens buttons change back to “Get” except the Home Screen where it stays as “Installed”

This final bug find shows that bugs and bug fixes are a never-ending part of software development even after “final” version is released.

### 1.9 Unit test examples

A screen shot of a computer program

Description automatically generated

*Image 2, Unit Tests*

### 1.10 Developer Guide

A screenshot of a software setup guide

Description automatically generated

*Image 3, Developer Guide Document*