Analysis of Alternatives

Assignment 1 - GitHub Inspector

Team Nibble

Ayesha Ali - 29587778

Ryley Angus - 22647597

Shreya Kathuriya - 29506751

Akshith Patil - 29717973

1st September 2019

Section 1: Software Platform

Terms of Reference

Decision
A software platform for the Git Inspector application must be chosen.

Options
The three potentially viable, modern options for a software platform are:

- Desktop application
- Mobile application
- Web application

Criteria
The principal criteria used to select the platform are:

- Portability between operating systems
- Usability of the platform across the entire gamut of device form-factors
- Maintainability of the application
- Security & integrity of the data accessed by the application
- Team member familiarity with the platform

Body
Portability between operating systems is a difficult goal to achieve for desktop & mobile applications. Whilst some components may be shared between target platforms, differences in graphical toolkits and networking related code will require substantially duplicated development efforts.Even within a target platform, differences between operating system versions can require additional development time and testing to address issues introduced by the platform vendor.

In contrast to this, a web application can effectively target all internet connected devices by utilising only the common features of the two to three predominant browser technology vendors. The wide availability of mobile browser simulators on desktop platforms further simplifies the testing and development process.

Similarly, web applications have advantages in usability across device form-factors. They are not susceptible to the scaling or display issues of many native applications on high resolution screens. The in-built zooming functionality of modern web browsers provides an additional usability advantage, particularly for graph and text-oriented projects such as our GitHub Inspector. The limited external accessibility tools provided by both desktop and mobile operating systems cannot provide the same clarity and readability.

A traditional desktop application may incorporate auto-update functionality, but this is not a native feature and will require development effort and on-going maintenance. The end-user may also choose to disregard updates entirely. Similarly, mobile applications may provide an option for auto-updates but this is not common due to data cap concerns. Both of these scenarios can result in the on-going use of older, buggy or insecure versions of an application. This ultimately reduces the user experience and limits the usage of the most recent development efforts.

Web applications are inherently capable of auto-updating without additional development effort. Each time a user accesses a web application, they will be presented with the most recent, complete and secure version of the application. Bug fixes and feature updates are transparently provided to the user, and minimal development effort is wasted on maintaining backend support for legacy versions of the application.

Whilst desktop applications on macOS, Windows and Linux are increasingly available in a “sandboxed” option, many continue to be provided as traditional applications. These are installed to a common location and data accessed and stored by such applications must be protected by the developers. Securing sensitive application data, particularly for critical information such as user credentials, requires substantial development effort. These credentials must be stored in either an operating system specific manner, such as a keychain or a bespoke solution must be created by the application vendor. Both of these options require ongoing maintenance and testing.

Mobile and web applications also have advantages over desktop applications in regards to data security. Operating system and browser enforced sandboxing is substantially more common and reliable compared to common desktop platforms. This isolation between apps and browser instances reduces the development resources directed towards data security.

Within our team, JavaScript is one of the most familiar programming languages. Each team member has completed at least one project in JavaScript and this familiarity is an asset. The time and resources required to train team members in an unfamiliar desktop or mobile application-oriented language and programming environment would be impractical. These unfamiliar environments may also reduce the accuracy of time estimates for the project. Further difficulties may also be experienced in the testing and deployment phases of the project.

Recommendations

After considering the alternative platforms in relation to the listed criteria, our team recommends the development of a web application. The portability of web applications across operating systems and devices is unmatched by either desktop or mobile applications.The maintainability and usability advantages of a web application were also critical factors in reaching this recommendation. If a desktop or mobile application were to be developed, the additional development resources consumed by testing and re-implementing functionality between operating systems and devices would pose a risk to the success of the project.

Section 2: Programming Language

Terms of Reference

Decision
We want to choose an appropriate programming language to develop the Git Inspector application.

Options
As a team, we've identified the following programming languages which could potentially be used for this project:

- Python
- Java
- JavaScript

Criteria
We will finalise our decision based on the criteria below

- Team member's experience with the language
- client-side capabilities of the programming language

Body
Our team's experience with Python can be described as intermediate. All members have used Python in previous units to develop simple programs, and we find Python convenient and quite easy to use. A majority of the team also prefers Python ahead of all other programming languages. However, some of the team members are unfamiliar with Web Frameworks for Python, which is a requirement for developing Web applications in Python. Python is also not a client-side language, it is a server-side scripting language. Server-side scripting languages require a separate server to be configured, whereas client-side languages do not need to be interpreted and executed by a server. Instead, the users web browser interprets and responds to the user's actions.

Only two of our team members have a substantial familiarity with Java, whilst the others have minimal experience and knowledge. Since the team already has programming experience, they may learn the fundamentals of Java and write sample programs during inception to gain some understanding. This could be costly if the two members are struggling with implementing some complex features, as they may need assistance from experienced team members. This will decrease developer efficiency and potentially delay delivery of the final product.

Java is also a server-side programming language. As mentioned earlier, using Java will require us to set up a server that process the scripts. This may increase the server load and decrease the responsiveness of the application. One benefit of server-side code execution is that the code being executed is not accessible to users. This reduces the risk of our code being copied and increases the difficulty of reverse-engineering our solution. User security is vital for this project, as end user's will be marking student projects. This may require the personal information and work of students to be processed by our Git Inspector.

JavaScript is the third programming language which must be considered for this project. Our team members are particularly confident with JavaScript, as it was one of the earliest languages they utilised. All members have used JavaScript in recent years to develop a web application of a similar scope to the current project. Our past experience and expertise in JavaScript is an advantage which should not be disregarded.

As JavaScript is a client-side programming language, it doesn't require persistent or ongoing interaction with our server. The code can be executed on the client device in their chosen browser, which allows for a much faster and responsive application. As discussed earlier, the JavaScript code is accessed and executed by the client web browser. This does require the code to be accessible to users, which can facilitate reverse-engineering or security analysis. Private information such as student names and contact details can be securely utilised by JavaScript, presuming they are not transmitted or stored in an insecure manner.

Recommendations

After evaluating several options and considering each alternative with regards to the listed criteria, our team has come to the conclusion of proceeding with JavaScript. JavaScript satisfies our criteria to the greatest extent. All team members have substantial experience working with JavaScript and have used JavaScript to complete similar projects. The client-side nature of JavaScript is a huge advantage, as it reduces the effort to configure a server. It is the most responsive solution with regards to user interaction, it allows the application to be run on any nearly platform, and the only explicit requirement is an internet connection and a modern web browser.