DevAlarm Deliverable 2:
Analysis of Alternatives

Team: **Cool Have Fun**

FIT2101 *Software Engineering Process and Management*

Table of Contents

1. Terms of Reference

1.1. Hardware availability

The platform and language chosen must be supported by the hardware owned by the team members. For example, if no team member owned an Android phone, it would make little sense to develop an Android app. This is of high importance.

1.2. User/Client requirements

The client requirements take priority when choosing a language and platform, while end users will be considered when making architecture decisions. This is of very high importance.

1.3. Cost of required services

Some services, such as cloud services and developer accounts, have associated financial costs. The team should consider these costs when making infrastructure and architecture decisions. This is of lower importance.

1.4. Skills/experience of team members

The relevant skills and experience of team members should be taken into account when choosing frameworks and services. Ideally, at least one team member will be intimately familiar with the tools used for a given part of the project, and they will act as the project lead for this part. This is of relatively high importance.

1.5. Ease and speed of development

Tools which facilitate quick development and minimal administration overhead should be preferred. Scripting languages and out-of-the-box frameworks should be preferred to languages with large amounts of boilerplate and frameworks with lacking package ecosystems. This is of medium importance.

2. Team Skills and Interests

The following table should be used as a reference for the languages known by, and technical area that interest, each team member. These are taken into account when analysing alternatives based on term of reference (1.4).

Name	**Languages known**	**Technical interests**
Ehtesham Ghani	Java, Javascript, Python, C	Android App Dev, Project Management, Web App Dev
Dana Casella	Python, JavaScript, Java, basic C++, Matlab, markdown	Web/web app dev, a little bit of game dev
Uyen Tran (Sara)	Python, Java, Javascript, Matlab	Android app dev
Patrick Brett	Java, JavaScript (+ ES7, TypeScript, React), Python, Swift, C, Matlab, PHP	Web dev, iOS dev, Android dev, DevOps, systems programming, data science, machine learning
Andy Zhan	Python, Javascript, C++, MATLAB, Java, C	Web design, interactive apps/programs

3. Evaluation of Options

3.1 Platform/architecture

Our project consists of both a front-end application as well as a back-end to handle requests from this front-end, as well as from Github. While the implementation of the back-end is largely abstracted away from the end user, the choice of platform for the front-end is very important. The three primary types of application are as follows:

- Mobile applications, running on mobile devices
- Desktop applications, running on desktop/laptop computers
- Web applications (whether optimised for desktop/laptop computers, mobile or both).

We considered the following options for the platform and architecture:

- Web application for desktop/laptop browsers
    - Advantages of this approach include:
        - Every team member has experience developing web applications
        - Platform independent
        - Easier to deploy updates and bug fixes
        - No installation required for users
    - Disadvantages of this approach include:
        - Desktop applications are usually more responsive and run faster
        - Desktop applications often provide a more thorough user experience.
- Developing a native iOS application
    - Advantages of this approach include:
        - Access to the rich library of UI assets bundled with the iOS platform would make parts of development quite simple
        - iOS provides reasonably straightforward protocols for network connectivity and REST API communication, upon which our back-end is based
    - Disadvantages of this approach include:
        - Steep learning curve for the majority of the team - only two of the five members have any experience developing iOS applications
        - Publishing issues - applications must either be installed with a developer certificate or issued via the official App Store, which requires manual approval from Apple
        - Unlike a web application, the user must manually update the software each time a new version is released
        - iOS devices might not be owned by all those who want to use the application
- Developing a desktop application for Windows
    - Advantages of this approach include:
        - Building on top of a well-supported and time-tested system
        - A native Windows application runs much faster than a web-based equivalent
    - Disadvantages of this approach include:
        - Windows applications are not by default designed to connect to the internet, and must implement libraries to do so, complicating development
        - No members of the team are familiar with Windows development
        - A general industry trend is that native desktop applications are migrating to either web-based applications (e.g. Zoom) or thin

desktop layers over web applications (e.g. Slack), and so a system built for this platform may not be future-proof

These platform/architecture options are evaluated using the five terms of reference (weighted according to their importance):

	Web application for desktop browsers	**Native iOS application**	**Windows desktop application**
Hardware availability (/5)	5	3	4
User/client requirements (/6)	6	6	6
Cost of required services (/2*)	2	2	2
Skills/experience of team members (/4)	3	1	0
Ease and speed of development (/3)	3	2	1
Total (/20)	**19**	**14**	**13**

3.2. Programming Language

Our project demands a decision for a programming language on both the front-end, which consists of the client-facing web app, and the back-end, which includes the server that handles requests from the web front-end as well as webhook requests from Github, and needs to be able to send email.

Numerous programming languages were considered for this goal. The main alternatives considered are evaluated below.

3.2.1. Front-end Programming Language

The options being considered for the front-end language/framework are:

- JavaScript with no front-end framework (i.e. vanilla JavaScript)
    - Benefits of this choice include:
        - Every member of the team has experience with Javascript
        - The application is not particularly complex and is entirely doable with vanilla Javascript
    - Disadvantages of this choice include:
        - Frameworks can increase code tidiness/cleanliness by abstracting complex code
        - Frameworks make it easy to maintain browser compatibility, simplifying the development process
- A React-based JavaScript application
    - Benefits of this choice include:
        - Fast development once proficiency is reached
        - Use of the React ecosystem (for example, D3.js has a React plugin for easy creation of charts and visuals)
        - Automatic toolchain setup with Create React App
    - Disadvantages of this choice include:
        - Majority of the team has no experience with this
        - Reasonably steep learning curve given the short time period of the project
- Elm, a front-end language that compiles into JavaScript
    - Benefits of this choice include:
        - Extremely type-safe development would reduce errors
    - Disadvantages of this choice include:
        - Limited ecosystem
        - Majority of the team has no experience with this
        - Very steep learning curve

These front-end programming language options are evaluated using the five terms of reference in the table on the following page (weighted according to their importance):

	Vanilla JavaScript	React-based JavaScript	Elm
Hardware availability (/5)	5	5	5
User/client requirements (/6)	6	6	6
Cost of required services (/2*)	2	2	2
Skills/experience of team members (/4)	4	1	1
Ease and speed of development (/3)	3	2	1
Total (/20)	**20**	**16**	**15**

* *where 3/3 is considered cheap and 0/3 is considered expensive*

3.2.2. Back-end Programming Language

The options being considered for the back-end language/framework are:

- TypeScript running in Node.js powering an Express server
    - Advantages of this choice include:
        - Type safety is an important consideration for medium-large projects that expect data to be presented in a certain way
        - Using TypeScript saves time spent debugging (type-related errors become very easy to spot) and results in easier-to-understand code
        - TypeScript results in code that is much easier to refactor when necessary
        - The team's JavaScript experience can also be used on the back-end
    - Disadvantages of this choice include:
        - Most team members only know vanilla Javascript
- Python with the Flask framework
    - Advantages of this choice include:

        - All team members know Python
        - Extensive ecosystem
    - Disadvantages of this choice include:
        - No team members are familiar with the framework
- Vanilla JavaScript running in Node.js powering an Express application
    - Advantages of this choice include:
        - Simplicity of using the same language on both the back-end and front-end
        - Very little upskilling required - all team members already know JavaScript
    - Disadvantages of this choice include:
        - TypeScript has few drawbacks and many advantages over JavaScript, and is also very easy to read for someone who understands JavaScript - therefore, it would make sense to use TypeScript instead
        - JavaScript has no built-in type safety

These back-end programming language options are evaluated using the five terms of reference (weighted according to their importance):

	TypeScript with Node.js and Express	**Python with Flask**	**Vanilla Javascript with Node.js and Express**
Hardware availability (/5)	5	5	5
User/client requirements (/6)	6	6	6
Cost of required services (/2*)	2	2	2
Skills/experience of team members (/4)	2	2	2.5
Ease and speed of development (/3)	3	1	2
Total (/20)	**18**	**16**	**17.5**

4. Recommendations

The ultimate decision was as follows:

- front-end: JavaScript with no front-end framework (i.e. vanilla JavaScript)
- back-end: TypeScript running in Node.js powering an Express server.
- We chose to deliver our product in the form of a web application optimised for desktop/laptop computers, due to its flexibility, simplicity and cross-platform compatibility.

These choices capitalise on the experience of all team members with JavaScript, and favour ease of development while meeting the client's requirements. A spike to prime members with Express experience was also created (link to Git). The cost of developing with this architecture is low, and the required hardware is readily available.
