Chat Away

# Software Design Template

Version 1.2

## Table of Contents

[Software Design Template 1](#_Toc101191177)

[Table of Contents 2](#_Toc101191178)

[Document Revision History 2](#_Toc101191179)

[Executive Summary 3](#_Toc101191180)

[Requirements 3](#_Toc101191181)

[Design Constraints 3](#_Toc101191182)

[Rationale 3](#_Toc101191183)

[Evaluation 4](#_Toc101191184)

[Reccomendations 5](#_Toc101191185)

## Document Revision History

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/14/2022 | Sereno Dominguez | Initial creation of design template |
| 1.1 | 04/03/2022 | Sereno Dominguez | Analysis of potential operating platforms |
| 1.2 | 04/17/2022 | Sereno Dominguez | Recommendations for operating platforms |

## Executive Summary

*Chat Away is a social media company with a working website and user base. They are looking to expand their website to mobile devices, via apps for both iPhone and Android.*

## Requirements

*Chat Away hasn’t specified a timeline for us to meet, but they are hoping to stay below a specified budget. Chat Away may need to expand server space to account for new growth. For the iPhone app, we’ll code the app in Swift, test it on multiple generations of iPhone, and publish it to the Apple App Store. For the Android app, we’ll code the app in JavaScript, test it on several varieties of Androids, and publish it on the Google Play Store. We’ll have to monitor user feedback to eliminate any bugs we couldn’t catch initially.*

## Design Constraints

*Create a user interface that will be shared by both apps.*

*Plug existing functions from the Chat Away website into the interface.*

*Translate this UI into Swift for the iPhone app and JavaScript for the Android app.*

## Rationale

*We design a single user interface to be used by both apps to establish cross-platform consistency, so users switching between devices will still be familiar with the app.*

*We don’t need to address the cloud storage of all the Chat Away user profile data, since they have a perfectly working website. All we need to do is use the app to pull existing content from their current servers to the mobile app and push new content from the mobile apps up to those same servers.*

*We will need to write the iPhone app in Swift because this is the language of Apple products, and is the only way we can get it published on the Apple App Store. The same goes for JavaScript and the Google Play Store.*

## Evaluation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Development Requirements*** | ***Linux*** | ***Mac*** | ***Windows*** | ***Mobile Devices*** |
| ***Server Side*** | *Extremely flexible terminal access to set up server*  *Low availability of apps for server hosting*  *Very cheap, with many distros being free* | *Terminal commands are relatively flexible for configuring server*  *Costly* | *Wide variety of software available because it is so widely used*  *Costly* | *Selective compatibility and weak security*  *Androids are cheap and generally configurable*  *iPhones are expensive and extremely limited* |
| ***Client Side*** | *Cost is the cheapest*  *Time and expertise is the highest to support wide variety of distributions* | *Cost is the most expensive*  *Expertise is moderate*  *Time is moderate because narrow range of devices* | *Cost is high*  *Time and expertise are minimal; if device can run Windows, it can run the program in most cases* | *Cost is more than Linux but less than Mac or Windows*  *Time and expertise are moderate* |
| ***Development Tools*** | *Can be coded in most major IDEs, but may require additional development teams to support different distributions* | *Can run any language, but Swift is preferred, which can take specialized expertise* | *Easiest to code, and least testing required, can be written in most IDE’s on most OS’s in any language* | *Androids require Javascript, iPhones require Swift, both can be written on any of the three other platforms*  *Require additional testing because of variety of devices and Android variants* |

## Recommendations

1. ***Operating platform:***

*I would recommend that the first platform we branch out to be Windows. It will require the least amount of skill for coding and the least amount of testing, which will save a lot as far as upfront development costs. But more importantly, it is the most widespread of the Big 3 operating systems, which will allow plenty of growth in markets all over the world.*

1. ***Operating Systems Architectures:***

*I would also advocate for Windows for the operating systems architectures. It can be used with a multitude of IDE’s and languages, in addition to access to the powerful and fast PowerShell, and the transition to a GUI is particularly easy with Windows.*

1. ***Storage Management:***

*As far as storage management, we should store all 200 images in the application itself. If we are going with a platform like Windows or any non-mobile platform, 2 GB is unlikely to deter customers given that they’re likely going to be storing the application on a laptop or desktop with a lot of storage. They can refresh the picture sets by pushing out application updates.*

1. ***Memory Management:***

*Given that Windows is used by such a varied set of devices with different capabilities, we should keep the application lean and only load what’s necessary. This is why I propose that we leave as much in storage as possible and only pull the current image pertinent to the current game session into memory. Even extremely slow and limited hardware will be able to load a 8 MB image and the insignificant metadata associated with team / player info.*

1. ***Distributed Systems and Networks:***

*I recommend that we use a cloud service to manage the server component of this game so that we don’t have to worry about all the logistics and costs of hosting it on our own. However, if we do go forward and host it ourselves, Linux would be the best platform to host it on; it’s the cheapest and the most flexible, and will be able to deliver an API that would be accessible to the client-side Windows computers.*

1. ***Security:***

*On the client side, Windows has many built-in protocols to handle security. It will already be screening their system as they go, so that side is covered. As for our server side, if we opt for a cloud service to host the distributed system, we can choose a service that ensures proper security of all the game, player, and team data. If we choose to host it ourselves, we will have to spend more time and resources in ensuring that our API is properly compartmentalized to only provide necessary information.*