GreetNGroup

Technical Specifications Document

CECS 491A Sec 05 8332

Team Gucci
Dylan Chhin 014430570
Eric Lee 014303261
Jonalyn Razon 014580772
Winn Moo 014633357 (Team Leader)
October 9th, 2018

Table of Contents

Backend Frameworks

ASP.NET Core 2.1	**Ruby on Rails 5.2.1**	**Django 2.1.2**
Pros: ● Maintained by Microsoft, so its well documented ● 3 team members are familiar with C# ● Multiplatform, can be developed on Windows, MacOS, and Linux ● Lightweight ● Higher performance than ASP.NET	**Pros:** ● Tons of libraries to add and use ● Testing automation provided through libraries ● Object oriented nature allows for faster development ● Framework makes it easy to accommodate for changes	**Pros:** ● All team members know Python ● Less coding, as Python has minimal syntax ● Scalable ● Works out of the box (includes ways to help the developer implement user authentication, or content administration)
Cons: ● Does not support many CMS (content management systems) ● Lacking library support	**Cons:** ● Taxing on system resources on runtime ● Slow boot-up time ● Lack of flexibility when implementing more complex functionalities ● Compounding mistakes (mistakes early on get worse the later you are into the development cycle) ● Poor documentation for libraries ● Our team doesn't know Ruby ○ High learning curve	**Cons:** ● Dependant on Python libraries ● Any changes to Django framework requires changes on the developers part ○ Changes to django are frequent, sites need to be updated frequently ● Dependant on Python ORM

Backend Frameworks cont.

The 3 backend frameworks that were contenders for our final backend framework came down to ASP.NET Core, Django, and Ruby on Rails. For RoR (Ruby on Rails), we opted to not use it as both the boot time and run time are slower than what it would be if our web app used Django or ASP.NET. There's also a language barrier as only one of our members has worked with RoR before which would add time needed to learn the language. The slow framework and language barrier is what led us to cross RoR off as a potential framework.

Next on the chopping block was Django. A huge benefit to Django that led us to a serious consideration of the framework was the fact that it was simpler, and therefore easier to develop with because it's all in Python. Not only that, but all our team members have used Python in some form or another. That being said, in the end the ultimate factor that led us away from Django was the fact that Django gets updated frequently and that means that the site needs to be changed frequently to accommodate for those updates. We feel that devoting time to updating the site would take away from time that could be used for developing new features.

We finally decided to develop our backend on the ASP.NET Core framework. What led us to this conclusion was the multitude of cons presented in the RoR and Django frameworks. ASP.NET isn't slow and it doesn't require constant upkeep. Our team has also developed applications with C# before, so we are very familiar with the language. These 3 pros were big enough to choose ASP.NET Core over the other backend frameworks.

Frontend Technologies

AngularJS 5.2.11	**ReactJS.NET 3.4**	**Vue.js 2.5.17**
Pros: ● Developed and maintained by Google ○ More likely that the language will continually be developed ● Mostly decouples frontend from backend ● Client side rendering ● Server performance increase ○ The amount of traffic between client is server is reduced, only need to send static files and serve json data through API calls ● Good for OOP developers, and for those who don't like javascript	**Pros:** ● Supported by Facebook ○ Facebook's design philosophy is API stability ● Purely javascript, possible to develop without using JSX ● Works out of the box, doesn't require extra tools to get started ● One-Way binding ○ Child elements don't affect parent data ○ Debugging is easier ● Virtual DOM allows React to render the page quicker ● Migrating between versions is easy ○ Facebook provides methods to streamline this process	**Pros:** ● Lightweight ● Simple, therefore cleaner code ● Easy to read ● Lower learning curve ● Large scaling ● Virtual DOM allows React to render the page quicker

Cons:	**Cons:**	**Cons:**
● Angular specific syntax ● Lot of tools that come with the application are confusing for first time Angular developers ● Migration issues when updating to a newer version	● JSX is recommended for easier reading ○ Team is unfamiliar with JSX ● Lack of official documentation ● Not opinionated, developers have too much freedom	● No clearly defined future roadmap ○ The framework can be completely different when updated to the next version ● Lack of English documentation ○ Most documentation is in Chinese ● Evolves fast, examples online would be outdated ● Fewer resources because of the size of the community ● Lack of common plugins/components

Frontend Technologies cont.

The best approach for developing our web application would be to completely decouple the backend and frontend frameworks. To best do this, we would create an API for the backend that the frontend would call to update itself. This would effectively allows us to use whatever frontend framework as they'd all work the same way to call the API. What our decision comes down to is how much as a team we like the different frontend frameworks.

The first framework we decided not to use was Vue. Although the framework is relatively simple and is very lightweight, the lack of direction that the framework has was a major disadvantage. Vue could be completely different from what it is today than what it would be a year from now, which doesn't bode well for longevity. Not only that but, but since Vue is so new, there isn't a large backing for Vue which means that there is a lack of common plugins/components as well as lack of resources to learn from.

For React vs Angular, it came down to ease of use. React is simply a library and is only the view in MVC. It's purpose is very straightforward and aligns with how we want to develop the web app. Angular, on the other hand, is a fully fleshed out framework with MVC thus complicating frontend implementation for us. It also uses Typescript, an addition to Javascript that our team is unfamiliar with. Because of its simplicity, our team chose React for our frontend.

Databases

SQL Server 2017	**MongoDB 3.6**	**MySQL Community Edition 8.0.12**
Pros: ● Support for ACID transactions with commit and rollback ● Built in data encryption/decryption ● Has support for .NET applications ● Our team is familiar with SQL	**Pros:** ● Free ● Scalability through using more machines ● No schema, which allows for changes to data model without much impact	**Pros:** ● Free ● Our team is familiar with SQL ● Support for ACID transactions ● Drivers for .NET ● Built in data encryption/decryption ● MySQL Workbench is cross-platform
Cons: ● Very costly, instances on AWS would cost at minimum $0.15 an hour ● Once you start with SQL Server, hard to migrate databases ● SQL Server Management only runs on Windows	**Cons:** ● Data size is higher (each document has field names stored in it) ● Less querying statements ● No support for ACID transactions ● Our team is unfamiliar with NoSQL	**Cons:** ●

Databases Cont.

Our very first choice for the database system that we would use for GreetNGroup was Microsoft's SQL Server. SQL Server is made by Microsoft, therefore would have had long lasting support. Not only that, but SQL Server also inherently supports .NET, a big welcome for our ASP.NET backend. However, SQL Server is a commercial product that is very expensive, even for a core of CPU. So we opted against SQL Server.

We considered MongoDB, as it was a free database system and is one of the top 5 most popular databases. But, it uses NoSQL, a query language that our team is unfamiliar with. This drawback alone is something that we weighed very highly in choosing against it.

For MySQL, our team is very familiar with the database, and we are relatively well versed in SQL. It has all the benefits that SQL Server has, but is also free. The fact that we have prior experience with MySQL and that it's free made the choice to choose MySQL our database very easy.

Web Servers

IIS 10	**Apache 2.4.35**	**Nginx 1.6**
Pros: ● Native support of .NET framework ● Included in Windows, the price is bundled with buying the OS ● Supported by Microsoft therefore has longevity	**Pros:** ● Good documentation because of widespread use ● Can be run on Windows, MacOS, Linux ● Free ● Low performance under high load	**Pros:** ● Lightweight resource utilization ● Developed after Apache, aimed to fix problems present in Apache ● Responsive under load ● Resource efficiency ● Can be run on Windows, MacOS, Linux
Cons: ● Only runs on Windows operating systems ● More prone to attacks as malware is more commonly written for Microsoft systems	**Cons:** ● Doesn't have as good as support for .NET that IIS does	**Cons:** ● Doesn't have as good as support for .NET that IIS does

Web Servers Cont

Apache is the most popular web server technology that is multi platform as well as free. As the most popular web server technology, this means more documentation which is helpful for our team as none of us has experience with it. What Nginx has going for it is it's high performance and has high scalability. Despite the pros for both those servers, neither has as great support for .NET as IIS does. Because IIS is developed by Microsoft, IIS natively supports .NET. This single advantage was enough to make our decision to use IIS for our web server technology.

Web Hosting Services

Amazon Web Services	**Google Cloud**	**Microsoft Azure**
Pros ● All services of AWS has APIs ● Data center groups globally, allows for scalability of web apps ● Pay as you go	**Pros** ● Long term discounts on price, begins after 1 month of usage ● Best average latency	**Pros** ● Straightforward pricing because of fixed storage amounts ● Convenient for Windows
Cons ● Recent high profile outages ● Price, more expensive than Azure or Cloud ● Steep learning curve ● Discounts after 1 year of usage, and payment in advance	**Cons** ● Google will take servers offline to do updates. ○ Can bring down sites at random ● Product and service range very limited in comparison to AWS and Azure	**Cons** ● Price increases as cost increases ● No support for bugs, need to pay for supports account to fix these as you encounter them

Web Hosting Services cont

For our choice of web hosting services, Google Cloud was an easy choice to not use. The fact that Google will take servers offline without warning to do updates is enough to dis-count them from the list of viable options. Azure seems to best support Windows Server only while AWS can do Linux or Windows servers. In terms of experience, members of our team have used AWS before while none of us have used Azure. Therefore, our team has decided to use AWS as the amount of learning won’t be as much as Azure.

Operating Systems

For development, our team is using a mix of Windows and Mac simply because that’s what we have. The member that has a Mac has the ability to dual boot into Windows if the need arises for Windows specific development.

Minimum Hardware Requirements

All our development machines have at minimum: Intel Core i3, 6gb of ram, and onboard graphics.