Project Proposal Report

Team 5: Myra Dotzel, Colin Floyd, Eric Higgins, Zak Kulphongpatana, Ryan Pope

Project Name: skilltree

Project Synopsis: A community-driven approach to skill verification.

Project Description: In today's society there are countless ways to gain new skills and acquire new knowledge. The industry around knowledge and skill acquisition is massive: Public K-12, Universities, Online educators like Khan Academy, Certification programs of all sorts; all this to the tune of $1.2 trillion annually.

However, if somebody wants to prove to others that they have those skills, their options are much more scarce. If you want to claim that you're competent in a broad field, you better have a college degree to back that statement up. If you want to claim knowledge in a specific domain, hopefully there'll be a certificate program built for that topic. There are significant problems with both of these routes, though. College educations are extremely varied across institutions, and, absent an intimate knowledge with every university's degree program, companies will tend to recruit based on broader school reputation instead of individual performance. Certificate programs, on the other hand, are often specialized to the point of limitation. They are often organized to accompany specific training regimens by the people who created those trainings, and there is little guarantee that the skills will generalize beyond that technology.

There is a need for a generalized and centralized platform for skill verification, and that's what we're building.

Project Milestones:

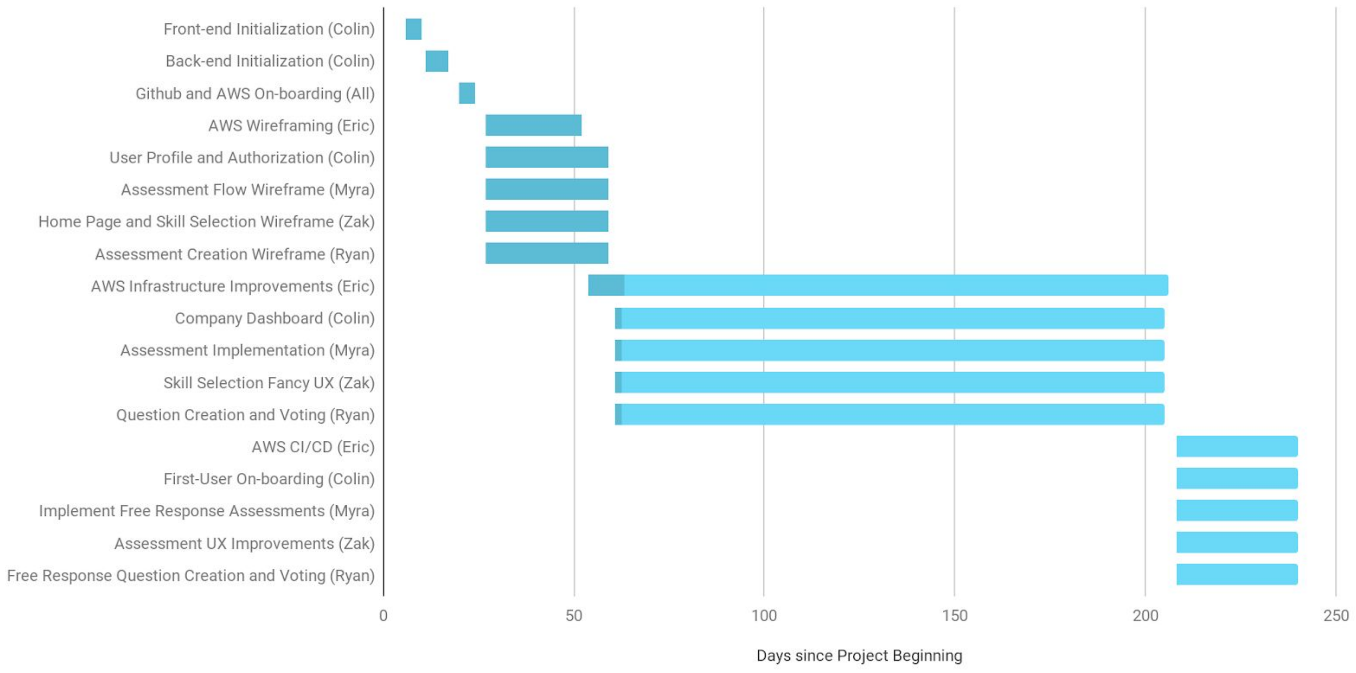
Fall 2020:

1. Page Mockups and User Flow Diagrams -- (10/16/20)
2. Integrate React Front-end with AWS Back-end -- (10/30/20)
3. Implement First Front-end pages -- (11/13/20)
4. Barebones Application with Working Endpoints -- (11/27/20)

Spring 2021:

1. Skill Selection Page Designed and Implemented -- (1/29/21)
2. Company and Recruiter Dashboard Implemented -- (2/26/21)
3. Assessment Flow Implemented -- (3/26/21)
4. Assessment Creation Implemented -- (3/26/21)
5. First Users On-boarded -- (4/16/21)

Gantt Chart



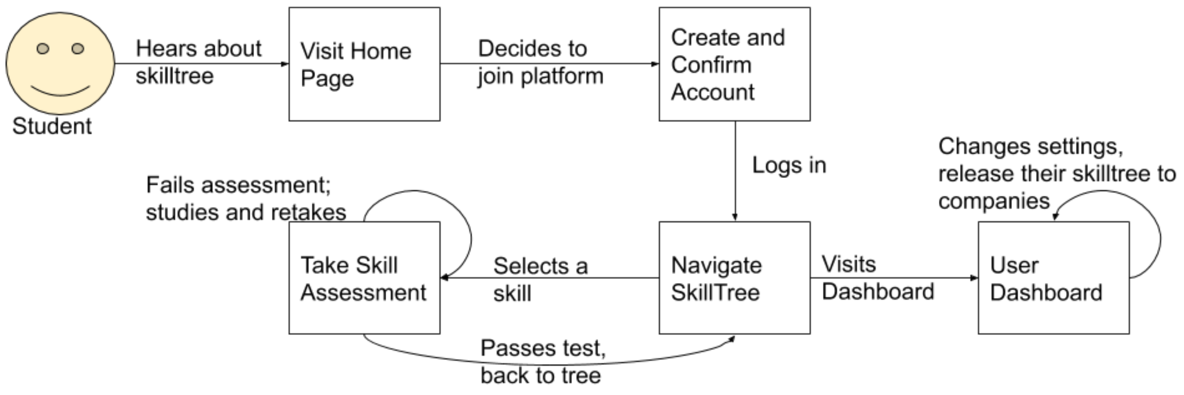
Project Budget:

Item Name	Vendor	Date Needed	Amount
Website Domain	AWS	10/31/20	$30
AWS Credits	AWS	12/31/20	$50
Total		--	$80

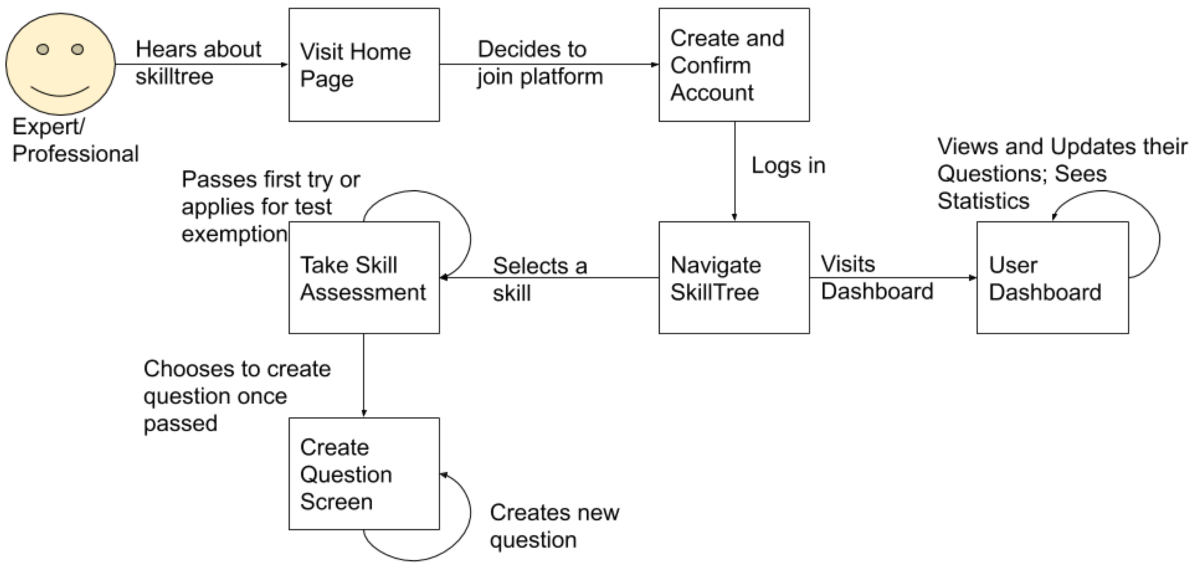
Preliminary Project Design:

We are building a web application with a ReactJS Frontend and a serverless backend based on AWS. The key motivations for this architecture were the importance of a seamless user experience and the necessity for the application to scale to accommodate large numbers of concurrent users. The first is necessary for two reasons: We are attempting to make knowledge certification *enjoyable*, this means that the assessments need to be easy to sit down to and the navigation pages absolutely cannot frustrate users. The second reason is that, if we're going to attract experts to share their valuable knowledge and contribute to our assessments, then we need to make it as easy to do so as possible. Building our application with the ability to scale in mind is important for the same reasons. It's necessary because this platform has the potential to scale to thousands or even millions of users, and we want to accommodate that.

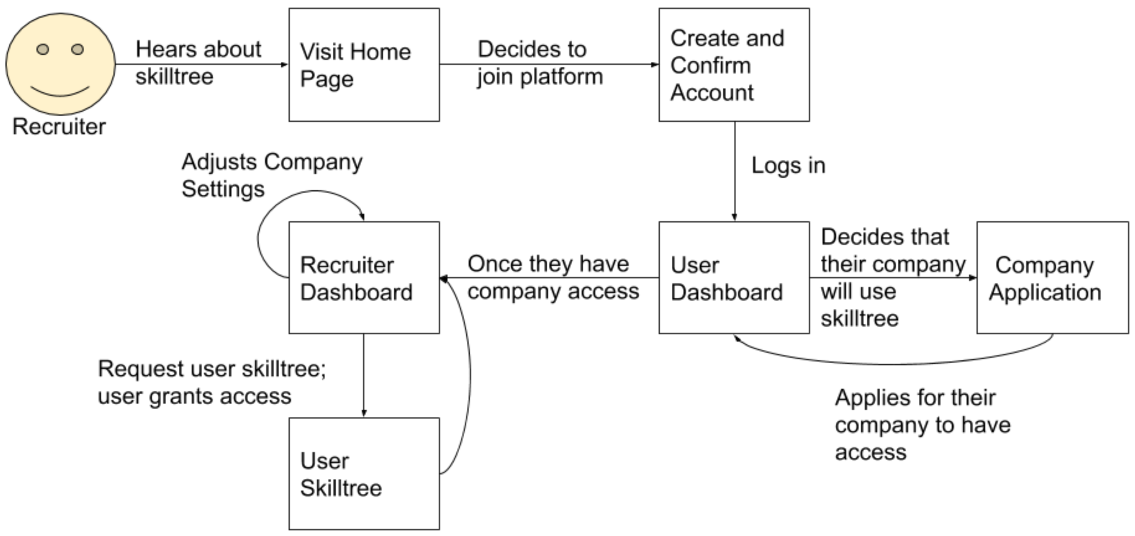
Our web app needs to support and facilitate three interconnecting user flows. First and foremost, we need to handle students taking assessments:



We then need the parallel flow for experts and professionals who want to share their knowledge, contribute to the assessments, and potentially grade written assessments down the road:



Lastly, we hope to allow for recruiters to utilize our platform. We want them to be able to define company settings for skills that the company believes to be valuable, technology specific skills that their engineers need for instance. Then they can also send requests for access to the skilltrees of people that apply to their company, which the user can then choose to release. The recruiters can then view all the skilltrees that have been released to them in the dashboard.


Recruiter
Hears about skilltree
Visit Home Page
Decides to join platform
Create and Confirm Account
Logs in
User Dashboard
Decides that their company will use skilltree
Company Application
Applies for their company to have access
Once they have company access
Recruiter Dashboard
Adjusts Company Settings
Request user skilltree; user grants access
User Skilltree

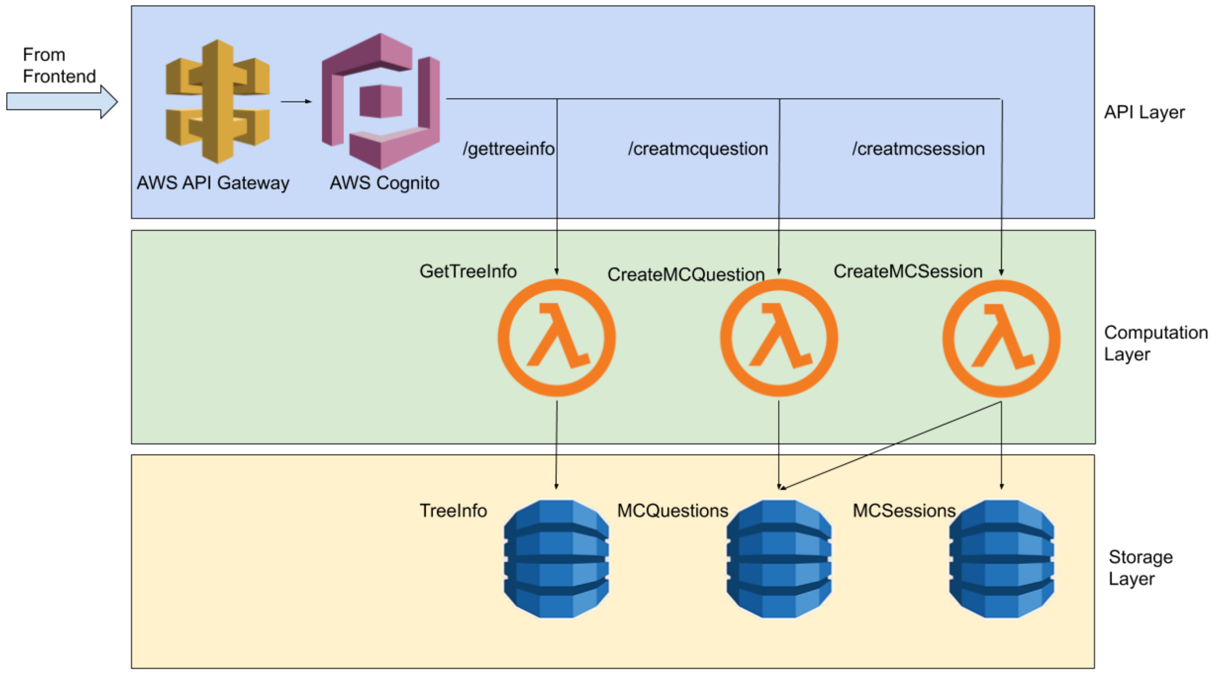
The need for a seamless UX for all three of these user flows drove us to select ReactJS as the frontend technology. React was a good choice for a variety of reasons. First off, it is very developer friendly. We can have built-in Typescript support to streamline bug-fixing, the routing and page-building are intuitive and flexible, and the hot-reloading massively speeds up the development cycle. For this reason, React has a thriving development community surrounding it, which helps us for another set of reasons. This community has contributed to a wide set of free-use page templates and components that we can lean on during our development. One of these is the Material-UI component set, which looks great and is easy to incorporate into React projects.

The ReactJS framework generates the HTML and CSS instead of us having to write them directly. The HTML is produced from the TSX that we write in the typescript files, and the CSS is produced from the Material-UI Style sub-library. We can also supplement this last part with the SASS library to allow for more dynamic styles.

Scalability concerns motivated the search for backend implementations other than the typical client-server approach. If we were to go with this traditional server approach, we would have had to spend the time and resources to provision any computers that we might have needed to handle the load we'd have for the website. We would also have had to forecast the load and provision the network in advance of how many users we were expecting, or otherwise we'd be running the risk of the website crashing as soon as we gain any sort of traction. This led us to seek other backend options, and one particularly appealing framework that has gained traction in recent years is the idea of a serverless backend.

The serverless model for computing is based on the premise that we no longer need to have standing servers that need to have a capacity to accommodate a certain number of requests at any given time. Rather, the code can just leverage the scale of cloud computing centers, and requests can cause the deployment of the code onto one of these centers' computers, at which point the request can be handled and the result returned. This is a fundamental paradigm shift because instead of having a set capacity for handling requests, each one can get handled individually as it comes in. The latter option is much more suited to scaling, because handling one thousand requests per second is functionally equivalent to handling one request per second.

We chose to implement this serverless architecture using AWS. Building serverless infrastructure in AWS led us to starting off with a three layer architecture, with an API Layer, a Computation Layer, and a Storage Layer, looking something like this image:



First, we have an API Layer. This is the main entry and exit point to the AWS backend. We chose to use AWS API Gateway. API Gateway wireframes the endpoints that can be reached from our frontend code through the internet. We can also integrate API Gateway with AWS Cognito, which is AWS's way of providing for user authentication. Tying this in here allows for us to validate that all requests to our API come from authorized users, which gives us more freedom in how we can allow our API to be reached, since we always guarantee that we only pass information to verified origins.

The different endpoints in API Gateway then can directly call the various AWS Lambda functions. This is the key difference between server-based architectures and serverless ones: The requests are not processed by one single computer but are rather diverted to the segments of code that they are specifically intended to run. AWS Lambda is massively parallelized, and AWS has strong integration for Lambda functions with the rest of its services. Each computation that we could possibly want to do on our backend would be its own Lambda function, isolated from all the others.

The Lambda functions then will need to access and store data, which we can do with AWS DynamoDB. DynamoDB is a NoSQL document storage system. This is perfect for our uses because we don't need the rigidity of SQL schemas in our application, and DynamoDB is very well tuned to fracture itself and recombine to accommodate large amounts of concurrent users, especially for read-heavy uses like ours. We can give our Lambdas access to various DynamoDB tables as needed, so functions can access multiple tables or none depending on needs.

Ethical Issues:

- We will be handling and sharing student data.

The primary ethical issue that we will be facing is that we are going to be in control of students' data concerning tests that they're taking and the scores that they get on the assessments. We plan to have a very rigid policy regarding data sharing. First off, we will only give out student assessment data with the explicit permission of the student. We will have recruiters request the data of the students by using the students' usernames, at which point the request will appear in the student's dashboard. The student can then choose whether or not to release their skilltree to the company that requested it. We also have decided that we will only release positive results from assessments, and we won't report precise scores when we do. This way students won't be afraid to take tests at risk of failure.

Intellectual Property Issues:

- Experts will be contributing their hard fought knowledge to create questions for the assessments. So whose IP will these questions be?

We want to be a knowledge sharing platform, in the same vein as Wikipedia or Stack Overflow. We will not be making money off of the questions themselves, at least not directly, so from that perspective, we would not need to "own" the questions explicitly. However, we also are inevitably going to contend with people posting the questions from our platform on the internet, so in order to strike these kinds of platforms down, we will need to have some ownership of the questions. If we released the questions under an open source license, this would be allowed, and it could make the assessments less valuable. In order to make this process of striking these sources down as streamlined as possible, it makes much more sense for us to have control of the question IP as opposed to the experts that contributed the questions. However, we will make it clear to the people submitting questions that we will *not* use those questions for profit outside of the skill assessment platform(like creating training courses or things in that vein).