# Surfworld

Ethan Garry, Matthew Duran, Nana Antwi, Kerem Guventurk, Andrew Schreiber
{epg2136, md3420, nka2122, kg2900, aj2409}@columbia.edu

## ABSTRACT

Our goal with *Surfworld* is to provide surfers around the world easy access to as much information about their favorite surf spots as possible. We designed a highly-scalable web-based application primarily utilizing Amazon Web Services for the implementation.

## INTRODUCTION

In the US alone, about 1.73 million people go surfing at least once annually. For a lot of these surfers, finding comprehensive information about a surfing spot is non-existent. That is why we conceived of this idea to create a unified platform for surfing enthusiasts to find information about surfing spots as quickly as possible. Surfing is one of the fastest growing sports in the world, and we believe our website can be integral in the adoption and expansion of the sport. As such we design every aspect of our systems to account for scalability and reliability at all times.

Our main user group is young technology-savvy surfers around the world (approx. ~30M). However, the intuitive design is easy to pick up for anyone who has used modern web-based applications.

We focused on three main areas of surfing: (1) Beaches (surf-breaks), (2) Surf Shops, and (3) Surf Lessons. Our core service is centered around beaches, but users can also search directly for shops and lessons. Users are able to search each category listed above by location and we return the application with the 10 nearest hits, displayed in “cards” as well as marks on an interactive map.

Users can then click on a specific item to get more details. For beaches, that includes our proprietary *Surf Score*. Surf Score differentiates our application from other Surf forecasting tools by aggregating a variety of factors besides swell. Our algorithm combines current weather and swell with nearby shop, lesson and restaurant availability to promote breaks with the highest rated breaks.

When users find a surf spot they want to check back on frequently, they can add it to their profile favorites for easy access. Favorites in Surfworld operates like any other application and is easy to remove from and add to.

In addition, we provide users with the opportunity to email themselves the lesson, break, and shop pages that interest them the most. The email will contain all of the details regarding nearby restaurants, lessons, shops, and surf specific data that was present on the page.

We believe that this platform is a one stop shop that will cater to surfers’ needs and provide them with value that simply doesn’t yet exist on the market.

## ARCHITECTURE

**Please see appendix A for diagram.** The following is a description of the amazon services we used and *how* we used them.

### Amazon Services

**Cloudfront** - AWS CloudFront CDN acts as the middle man between our frontend hosting and users. With CloudFront, we cache HTML, CSS, JavaScript, and images. Since the cache is closer to the user, the content will be delivered with minimal latency. We were able to leverage this service to allow for HTTPS secure traffic to our website.

**Lambda Functions** - The main backbone of our back-end. They allow building scalable, asynchronous, flexible and secure backends. We use lambdas to execute http requests from our front-end and to speak with our databases.

**API Gateway** - API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, CORS support, authorization and access control, throttling, monitoring, and API version management. We put an API gateway in front of all of our lambda functions.

**S3** - An object storage service that offers industry-leading scalability, data availability, security, and performance. We use S3 to store our front-end files as well as user profile images.

**SQS** - A fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications. SQS eliminates the complexity and overhead associated with managing and operating message-oriented middleware, and empowers developers to focus on differentiating work. We use SQS to handle email requests from users and prevent overload to SES.

**SES** - A cost-effective, flexible, and scalable email service that enables developers to send mail from within any application. We use SES to send emails to users upon request. We currently only have 5 email addresses verified in our sandbox, but if we were to fully deploy this service could scale accordingly.

**RDS** - A collection of managed services that makes it simple to set up, operate, and scale databases in the cloud. We use RDS to store location based data for our beach, shop and lesson data.

**CloudWatch** - Amazon CloudWatch is a monitoring and observability service built for DevOps engineers, developers, site reliability engineers (SREs), IT managers, and product owners. CloudWatch provides you with data and actionable insights to monitor your applications, respond to system-wide performance changes, and optimize resource utilization. We use CloudWatch to schedule two of our lambdas, allowing us more predictable control over these compute intensive operations. First, we schedule a cloudwatch trigger to poll SQS on the minute, to check if there are emails that need to be sent through SES. Second, we schedule our Surf Score lambda which is a huge batch operation to be performed at a low load time every day, around 3 AM local time. *Note*: our triggers are currently off (so we can stay in the free tier), but the hooks and infrastructure are ready to deploy.

**CodePipeline** - AWS CodePipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates. We use CodePipeline to automate the update and build for our front end code once changes are pushed to our GitHub repository.

**DynamoDB** - Amazon DynamoDB is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale. We use this NoSQL database to enforce a flexible schema design for our beach data, our bread-and-butter. For each beach, we will store nearby restaurants, lessons, and shops as a large payload to be returned for our beach details page. The flexibility for this design allows us to add to the beach schema, as we implement more features.

## API DESIGN

### /get_favorite
Param: id, category
Description: Fetches a favorited item (beach, shop, or lesson) from either surfline, shops, or lessons tables
Methods supported: GET

### /profilepic/{user_id}
Param: user_id, picture (in binary format)
Description: endpoint is an S3 proxy on API Gateway used to store and retrieve the profile picture for a user
Methods supported: GET, PUT

### /search
Param:  is_nearby, location, nearby_lat, nearby_lon, search-type
Description: Based on search type we will return an ordered list (by distance to location) of the requested data to the user. If is_nearby is true, then we expect the app to pass by geo-coordinates for my search, otherwise they will pass in a string like “New York City” which gets parsed to a geo-location using a third-party API.
Methods supported: GET

### /send_email
Param: type, body, user_id
Description: Sends an email to the user containing select information the details page they are on. Can be from a beach, shop, or lessons page.
Methods supported: POST

### /surf_score
Param: id
Description: Fetches the details for a beach. Several external APIs are called in conjunction with stored static data from our beaches database.
Methods supported: GET

### /user/favorites
Param: user_id, favorite_id
Description: Fetches or adds/deletes the favorites for a user id. Stores the data in a dynamoDB table.
Methods Supported: GET, PUT, DELETE

### /user/details
Param: user_id
Description: Stores and retrieves the account details for a user. We chose a schema less design for account details so that it was easy to evolve the definition of information we store for a user overtime. In the future we would like to add more

structure around user details so that there could be certain guarantees around what data will always be there.
Methods Supported: GET, PUT, DELETE

**Lambda Functions**

LF1-Search:
An AWS lambda function triggered when a user searches for a beach, surf shop or surf lesson by location. This function returns a list of results within a certain radius of the location. More details in project details section.

LF2-surf-score:
An AWS lambda function triggered when a user wants more details on a beach. This function relies on external data to calculate a surf score (how good is the surf right now). More details in project details section.

LF3-get-coords:
Takes a location in the form of a string and returns latitude and longitude. Utilizes a free external API.

LF4-get-weather:
Takes coordinates (lat, lon) and returns weather data for that location. Utilizes a free external API.

LF5-get-surfline:
Takes in a surfline id (external API) and returns surf data for the next three days. More details in project details.

LF6-Send-Email-To-SQS:
Takes the information displayed on a details page for a beach, surf shop or surf lesson and produces a message for SQS

LF7-user-profiles:
Handles all user related endpoints, deployed as one lambda for simplicity and code sharing. Currently endpoints available are user favorites and user details.

LF8-AbsorbSQS-Send-Email:
Absorbs a message from SQS, formats the email depending on the information and utilizes SES to send an email to the user.

LF9-get-favorite:
Return information about a user's favorite beach, surf shop, or surf lesson.

## PROJECT DETAILS

### Search

Search is our key functionality which populates the front end with real data based on query parameters. We parse locations to geo-coordinates, and then query our database for the top 10 nearest items based on the search type. A user can search for nearby beaches, surfshops, or lessons. The search implements a growing search radius algorithm which will continue to expand until we return 10 items. We needed this algorithm since for our users, our data is very niche and specific to beaches that have high quality surf breaks. This lambda's functionality is key to Surfworld's functionality, and provides user with real valuable surf data.

### Surf Score

Surf score is our "special sauce". Other surf companies generally just rely on surf conditions to calculate a score, but we take in a multitude of factors: weather, surf conditions, nearby surf shops, nearby lessons, and nearby restaurants. When a surfer is interested in finding a new spot, they are interested in how easy it is to find equipment and also to talk with locals who know the spot (surf shops). For beginners, nearby lessons is more important. Regardless if you are a beginner or veteran, most surfers are ravenous post-surf and nearby restaurants are key - no one wants to drive an hour after a session to get an acai bowl.

**Users**

We choose to model users simply using the ID provided by OATH2 and then storing blobs of information per user. This allows us to dynamically update and change what information we store from the UI over time while we are building out the MVP. We accomplish this by storing any JSON that the UI provides into DynamoDB and then we can restream the entire contents back when requested. In the future we would like to define certain sections of the user data, some with more defined schemas and some still without schemas, that way we can provide guarantees around certain data points being available, and have better security over confidential data points.

**Emails**

In order to ease access to certain information or simply to view it in a different format, we allow users the option to receive an email with most of the information they see on the details page. We use SQS and SES to facilitate this functionality.

**Data and Third Party APIs**

We originally sourced our beach data from two popular, well-established websites. We ultimately decided to use data from only one website, Surfline ([www.surfline.com](www.surfline.com)), since dynamic data fetching on the details page was too slow to utilize both.

For surf shops, surf lessons, and restaurants we utilized the geo location of our sourced beach breaks to then query the Yelp API to get the data. This was an arduous process, which required many sanity checks. But in the end, we were able to tie all of that data to all of the Surflien beaches and write that data to Dynamo DB. This data is the workhorse for all of the detail pages that exist.

**Beach DynamoDB:** As noted above, we queried the Yelp API using the geo-coordinates for all of our beaches to get real-world data for restaurants, surf shops, and surf lessons. During the scrape, we made a json payload which ties all of that data to a beach id, and then wrote it to Dynamo DB on the fly. This database will be accessed when a user clicks on a particular search result item to see more detailed information.

**Search Tables:** We have three relational databases that are used for location based searches: beaches, surf shops, and surf lessons. Each table contains latitude and longitude to facilitate a users search, plus some additional information for ease of use in other areas of the application.

**User Tables:** We store user info in DynamoDB tables, keyed by user id from OATH.

**GeoLocation API:** We used a free geolocation API that takes a string as input and returns the closest matching location: https://developer.myptv.com/.

**Weather API:** We used a free weather location API that takes in a latitude and longitude and provides detailed weather data for that region: https://www.weatherapi.com/.

**Google Authentication:** We used a React JS OAuth library called client-oauth2 and a Google Authentication key in order to authenticate our users. If the login was successful, Google Authentication flow would give us that user's Google Account ID which we would use as the user's ID in its Surfworld profile as well.

**Maps API:** We leveraged React Leaflet API library to create interactive maps to display results of search queries. We ultimately chose to use Leaflet instead google maps as specified in our project writeup since Leaflet is free, and performance is comparable.

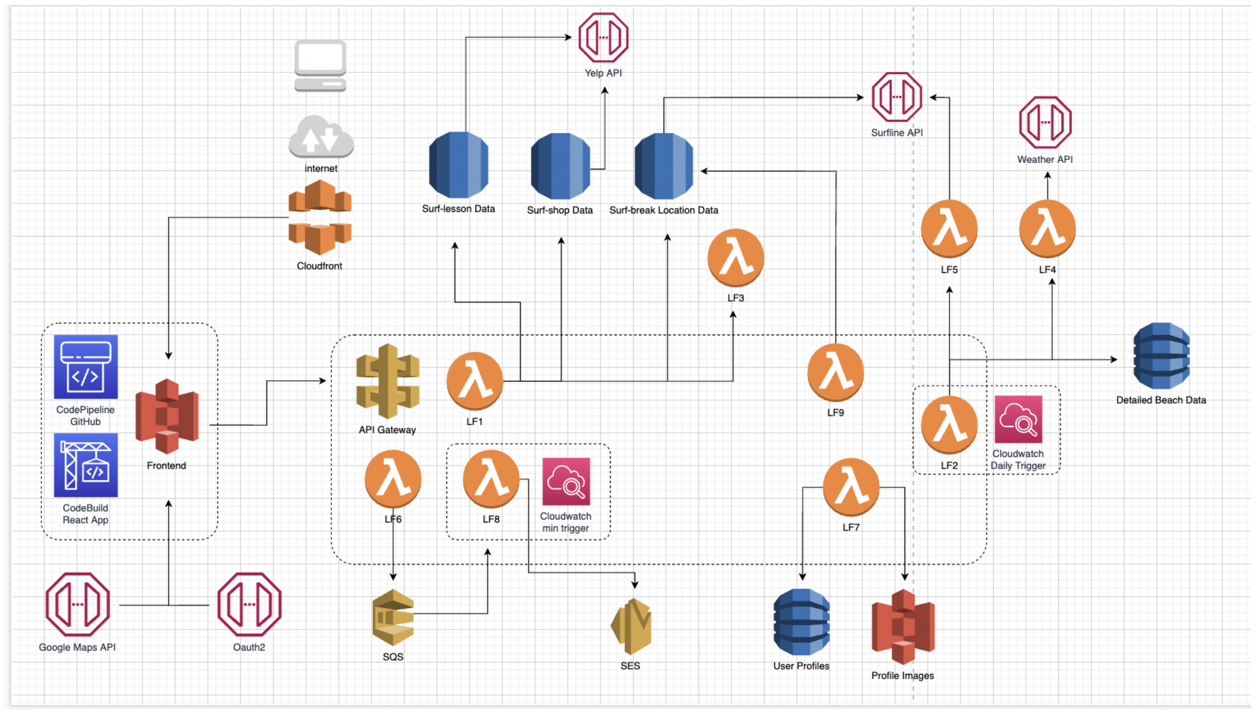
## CONCLUSION AND FUTURE WORK

Overall, we're satisfied with our product as it gives technology-savvy surfers information about surf locations, surf shops, and surf lessons all around the world. We also believe that our profile section is important as well as it allows our users to customize their own surfing experiences.

In the future we would like to add more social interaction opportunities for our users by letting them leave comments and reviews directly in the app and create sub pages for local communities. We would also like to make our product deployed in more regions for stability and availability purposes. Finally, we would like to create a mobile version of our application to make it more convenient for our users to access Surfworld on-the-go.

## Appendix

### A. Architecture Diagram



### B. Demo Walkthrough Link

https://www.youtube.com/watch?v=WhptlGJk7VQ&ab_channel=EthanGarry

### C. Surfworld Link

https://day6lya0kl7ns.cloudfront.net/