



## ReferAll/ResourceFull Technical Documentation

Version 1.0.0

Written by AnnieCannons Team

Updated: 8-25-2023

<b>ReferAll/ResourceFull Technical Documentation</b>	<b>1</b>
<b>Application Information</b>	<b>2</b>
<b>Key AWS Resources Used</b>	<b>2</b>
<b>Additional AWS Resources Used</b>	<b>3</b>
<b>VPCs and CloudFormation</b>	<b>3</b>
<b>Data Audit</b>	<b>4</b>
<b>Databases</b>	<b>4</b>
<b>Data Model</b>	<b>5</b>
<b>Code Repositories</b>	<b>6</b>
<b>Code Deployment</b>	<b>7</b>
<b>Config</b>	<b>9</b>
<b>Administration</b>	<b>9</b>
<b>Analytics</b>	<b>9</b>
<b>Security</b>	<b>9</b>
<b>Design Assets</b>	<b>14</b>
<b>Logs</b>	<b>14</b>
<b>Analytics</b>	<b>14</b>
<b>Build Tracking</b>	<b>15</b>
<b>API Endpoints</b>	<b>15</b>
<b>Process Docs</b>	<b>19</b>
<b>Common Bugs</b>	<b>21</b>
Run Front End App on Device DURING Development*	21
Undefined symbol: _OBJC_CLASS_\$_	21
Library not found	21
Database Configuration.	21
<b>Decisions</b>	<b>22</b>

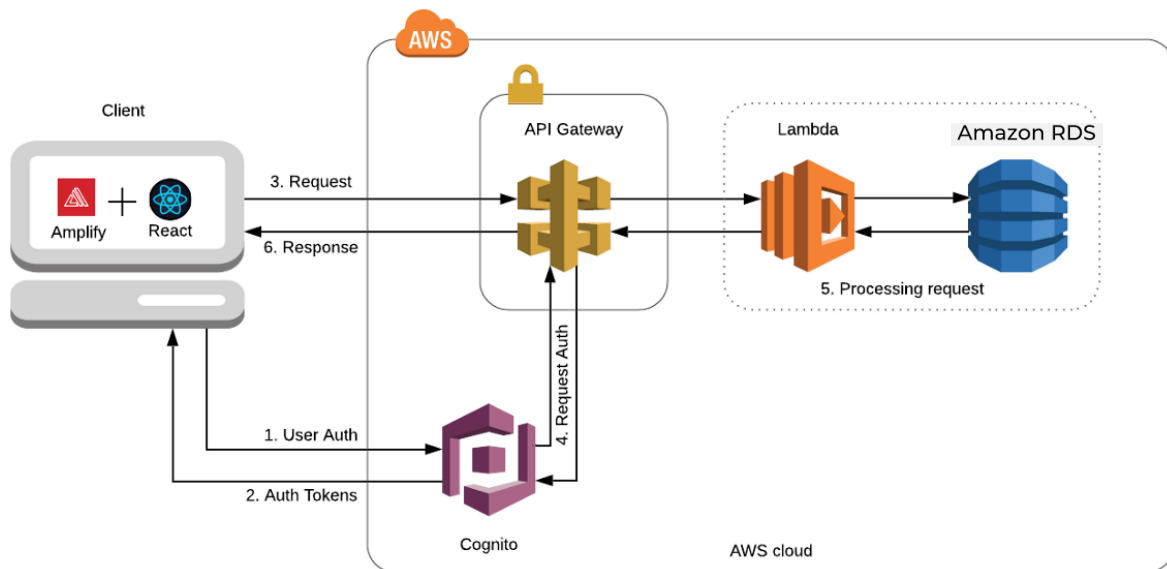
---

## Application Information

The ResourceFull Platform consists of two applications: a mobile application and a web application. This platform aims to help survivors of human trafficking find and connect with organizations that provide services to survivors. The mobile application is a React Native application with backend resources hosted in AWS. This application is referred to as the “Survivor App”. This application helps survivors enter intake information, search providers, and submit intake information to providers. The web application is a React.js application with backend resources hosted in AWS. This application is referred to as the “Provider App”. This application helps providers keep information about their services up to date and allows for them to directly connect with survivors who are requesting access to their programs.

## Architecture Map

This architecture is replicated for the web and mobile application.



## Key AWS Resources Used

- VPC → Cloud network
- API Gateway → API interface

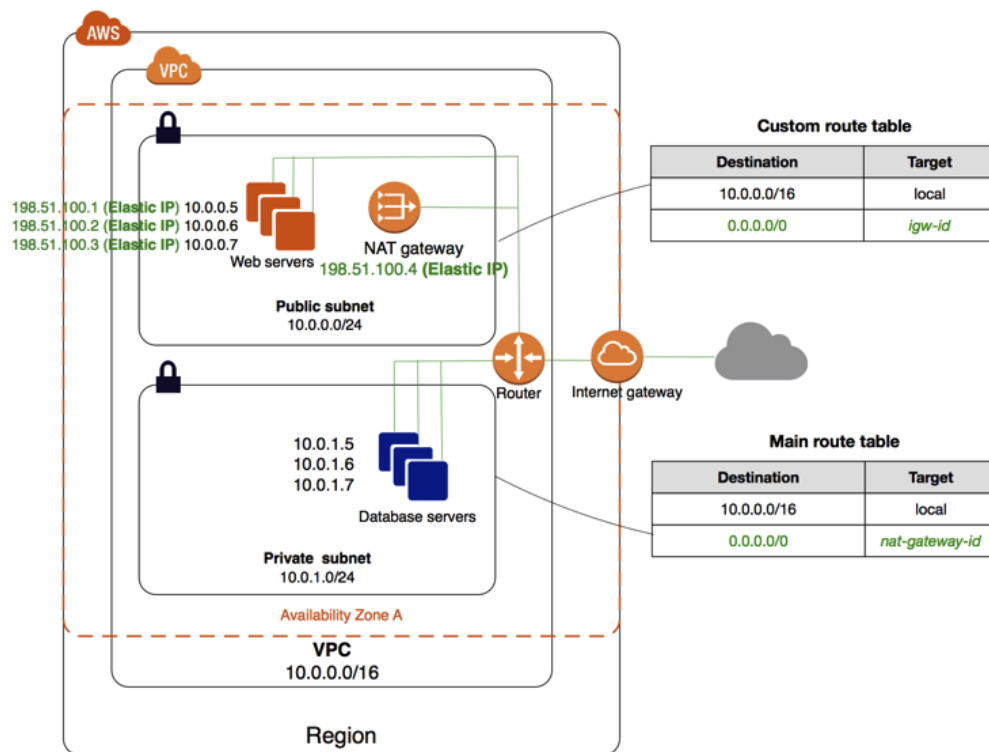
- RDS → Database system and server
- Lambda → Cloud functions
- CloudFormation → Resource management
- CloudWatch → Logs
- Cognito → handles user sign-up and authentication to mobile and web apps
- SES (Simple Email Service) → scalable email to communicate with users
- S3 → Asset Storage

### Additional AWS Resources Used

- EC2 (Elastic Computer Cloud) → scalable cloud computing
- Elastic Load Balancing → automatically distributes incoming traffic across multiple healthy targets in one or more Availability Zones
- Config → provides an AWS resource inventory, configuration history, and configuration change notifications to use security and governance.
- Key Management Service → centralized control over the cryptographic keys used to protect data
- CloudTrail → enables operational and risk auditing, governance, and compliance of your AWS account
- Data Transfer → data is moved either to the Internet from AWS or moved between AWS instances across regions or Availability Zones
- SNS (Simple Notification Service) → messaging service re: lambda errors
- SQS (Simple Queue Service) → send, store, and receive messages between software components at any volume

### VPCs and CloudFormation

A virtual private cloud (VPC) is a virtual network dedicated to an AWS account and is logically isolated from other virtual networks in the AWS Cloud. A VPC spans all Availability Zones in the AWS Region. One or more subnets is to be created in each Availability Zone. All AWS resources are contained inside the VPC's private or public subnets. Private subnets do not have a route to an Internet Gateway. The Lambda functions in the private subnet can access the Internet (for API calls) through the AWS NAT Gateway.



CloudFormation is a service that helps to model and set up AWS resources, thereby simplifying infrastructure management. This service provides ease of scalability, replication of infrastructure, and controlling and tracking changes.

- Backend resources will be deployed and maintained as an individual Stack through Cloudformation.

Serverless computing, is an execution model where the cloud provider (AWS) is responsible for executing a piece of code by dynamically allocating the resources. And only charging for the amount of resources used to run the code. The code is typically run inside stateless containers that can be triggered by a variety of events including http requests, database events, queuing services, monitoring alerts, file uploads, scheduled events (cron jobs), etc.

## Data Audit

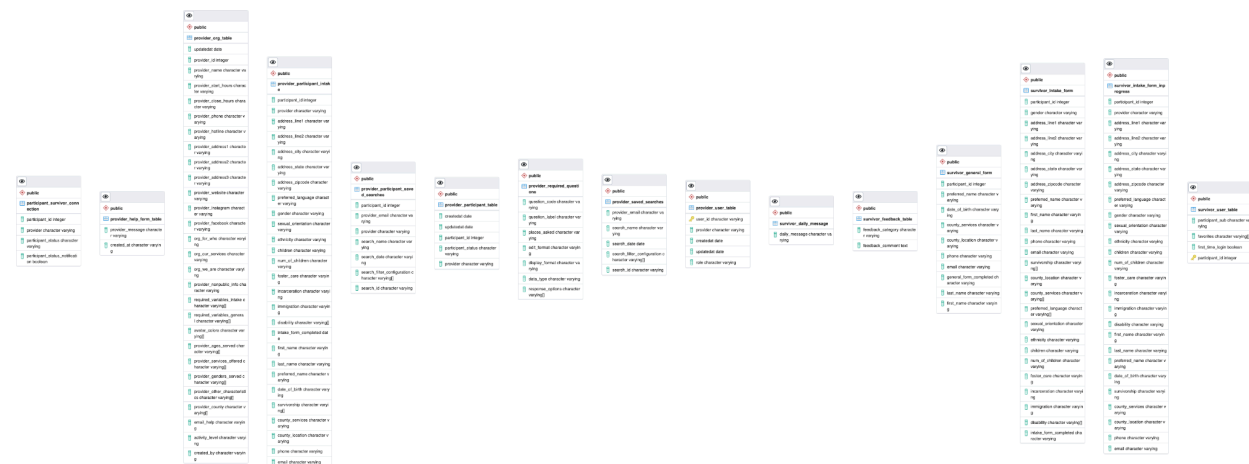
- [Current Spreadsheet](#)


## Databases

The application's database system utilizes Amazon Relational Database Service (Amazon RDS), which allows for easy set up, operation, and scalability in the cloud. RDS has been configured to allow hardware provisioning, database setup, patching and backups.

The database itself uses PostgreSQL, an open source object-relational SQL database system that can safely store and scale data workloads. Database tables are created and maintained in pgAdmin. The databases are configured and deployed from the serverless YAML file and AWS CloudFormation. Any edits to the throughput or authorization of the databases should be managed through the serverless framework in the serverless.yml file. Auto Scaling can be enabled, and the database will scale automatically.

## Data Model



Database Sync:  ResourceFull Database Sync

### Authentication and Authorization:

Authentication and Authorization in login is provided by AWS Cognito.

Amazon Cognito User Pools is a full-featured user directory service to handle user registration, storage, authentication, and account recovery. Cognito User Pools returns JWT tokens to the application. Amazon Cognito Federated Identities authorizes users to use the various AWS services and backend resources. With the identity pool, the application provides temporary AWS credentials with defined permissions to access other AWS services directly or to access resources through Amazon API Gateway.

Cognito User Pools acts as a source of user identities (identity provider) for the Cognito Federated Identities. AWS Amplify, a frontend library, uses User Pools to store user information and handle authorization, and it leverages Federated Identities to manage user access to AWS Resources. With Amplify, user's login information can be stored locally and allow for ongoing sessions from the same browser.

## User Onboarding Flow in Cognito

### Code Repositories

We use Git for version control, and the repository is hosted by Github.

This app comprises the following deploys to the public:

- Front-end React Application
  - Github Private Repository:
  - [https://github.com/ReferAll-app/frontend\\_provider\\_mvp1](https://github.com/ReferAll-app/frontend_provider_mvp1)
  - Deployed from the main branch
  - Dev environment published from the develop branch
- Frontend React Native Application
  - Github Private Repository:
  - [https://github.com/ReferAll-app/survivor\\_frontend\\_mvp1](https://github.com/ReferAll-app/survivor_frontend_mvp1)
  - Deployed from the main branch
  - Dev environment published from the dev branch
- Backend Resources in AWS Survivor App
  - Github Private Repository: [https://github.com/ReferAll-app/backend\\_survivor\\_mvp](https://github.com/ReferAll-app/backend_survivor_mvp)
  - Deployed from the main branch
  - Dev environment deploys from the develop branch
- Backend Resources in AWS Provider App
  - Github Private Repository: <https://github.com/ReferAll-app/backend-provider-refactor>
  - Deployed from the master branch
  - Dev environment deploys from the develop branch

Survivor App				Provider App			
Frontend (React Native App) Repo		Backend (Serverless/Node.js) Repo		Frontend (React.js App) Repo		Backend (Serverless/Node.js) Repo	
TestFlight Build (main)	<a href="#">TestFlight Build (dev)</a>	Main branch	<a href="#">Dev branch</a>	<a href="https://github.com/llant-johnson-6a96">https://github.com/llant-johnson-6a96</a>	<a href="https://github.com/mfy-unicorn-2bba1">https://github.com/mfy-unicorn-2bba1</a>	Main branch	Dev branch

branch)	branch)			<a href="#">3b.netlify.app/</a> (Main Branch)	<a href="#">4.netlify.app/</a> (Dev Branch)		
---------	---------	--	--	--	--	--	--

**\*Now live**

### **Launch Lite - This is the service provider list with no user authentication**

- Github: Repo: <https://github.com/ReferAll-app/frontend-provider-list>
  - Cloned from main
- URL served from Netlify: [serviceproviders.resourcefullapp.org](https://serviceproviders.resourcefullapp.org)
- APIs that will be called
  - getAllProviders
  - addHelpForm

Launch Lite	
Frontend (React App) Repo	Backend (Serverless/Node.js) Repo
<a href="#">gentle-quokka-e76489.netlify.app</a> (Main Branch)	Main branch, production



### **Service Provider Table**

- provider\_org\_table
- Currently:
  - Production DB: working on encryption, not to be set live, only have test accounts
  - Dev DB: Mock data from spreadsheet + test data from devs
- Plan (as of August 31, 2022)
  - Connect the service provider list app to the production BE
  - Upload “Live” provider list to the production DB, remove encryption on the addProviderOrg lambda (aka, don’t touch that table when adding encryption moving forward, or until ready)
  - Continue using Dev DB and BE for testing and updates

### **Code Deployment**

- Front-end React Native Application for Survivors
  - a. Any major feature or change gets its own branch, which is merged into the development branch once the feature is completed. When it is time to push an

update, a release branch is created and pushed to a staging environment and tested. Once any changes are finalized, the release branch is merged into both the master and the development branch.


- Commit and push your feature branch code to Github.
- Go into Github to merge and resolve any merge conflicts the feature branch has with the dev branch for AC testing.
- In Github, merge and resolve any merge conflicts with the main branch for security testing.
- Pull the updates to the main branch to your computer.
- b. Running Locally on iOS
  -  Running the Survivor App Locally - iOS
- c. Production Notes:
  - Mobile home screen logo icon and application splash screen logo images are updated in **XCode > SurvivorAppFrontEndMvp > LaunchScreen**.
  -  [Production] CI/CD for iOS
  - Be sure to update the [Build Tracking](#).
- Front-End WordPress Marketing Website
  - a. Code is built with Elementor within the WordPress platform.
  - b. For trademarking purposes, we needed to [transfer domain names](#).
  - c. After transfer, our URL went live at: <http://resourcefullapp.org/>
  - d. Once our apex domain was established, we [created subdomains](#) (to follow) for:
    - Front-End React Website for All
    - Front-End React Application for Providers
- Front-End React Website for All
  - a. Loaded from CI/CD from Netlify
    - Deploy: build/
    - Load from / on the master branch
    - URL endpoint:
      - <https://gentle-quokka-e76489.netlify.app/>
        - Custom Domain: <https://serviceproviders.resourcefullapp.org/>
- Front-End React Application for Providers
  - a. Loaded from CI/CD from Netlify
    - Deploy: build/
    - Load from / on the master branch
    - URL endpoint:
      - <https://gallant-johnson-6a963b.netlify.app/>
        - Custom Domain: <https://my.resourcefullapp.org/>
    - Dev build
      - Deploy: build/
      - Load from / on develop branch




- URL endpoint: <https://comfy-unicorn-2bba14.netlify.app/>
- Backend Resources in AWS
  - a. Any major feature or change gets its own branch, which is merged into the development branch once the feature is completed. When it is time to push an update, a release branch is created and pushed to a staging dev environment in AnnieCannons AWS environment and tested. Once any changes are finalized, the release branch is merged into both the master and the development branch. This is then synced with the Cloudformation Stack for production in the AWS environment and tested.
    - Process: Export AWS Access Keys
    - Execute serverless deploy
    - Cognito configuration (for any production change)
  - b. Backend Deployment process to AWS
    - Commit and push the dev branch code to Github.
    - In VS code, with the correct secrets.json file, deploy the dev branch code to AWS using serverless.
    - Go into Github and merge the dev branch into the main branch.
    - Pull the updates to the main branch to your computer.
    - In VS code, with the correct secrets.json file, deploy the main branch code to AWS using serverless.


## Config

Serverless.yml file handles all of the configuration and deployment of AWS resources

- Resource handles to the database, Lambdas, API gateway, and Cognito
- Environment variables are maintained through the serverless.yml file as well as through AWS\_IAM functions.
- AWS Lambda runtime support occasionally deprecates for Node.js.
  -  [Local Development] Node Updates for Serverless

## Administration

- Updating the backend architecture is performed with access to the AWS environmental access and secret keys as executed in the command line and configured in the AWS IAM Users service.
  - Additional users may be added in IAM at any time.
- All services are deploying to the AWS us-east-2
- AWS IAM service currently maintains two roles: 1) Cognito authorizers, 2) Lambda authorizers to access API Gateway
- Roles and Access:  ReferAll Database Audit , see user access tab

- Dev Volunteers for AC:
  -  [Local Development] Backend Development Repo Access

## Security

- Data Theorem
  - Data Theorem is a security company that is providing pro-bono security testing and a security dashboard and SDK for ResourceFull
    - Video overview: [recorded Overview/Demo](#)
  -

Asset			
Mobile App (Survivor App - React Native)	iOS - Github Actions, when new production deployment, DT receives app for testing, results displayed in <b>Mobile Secure Dashboard (SAST/DAST)</b>	Android - Github Actions, when new production deployment, DT receives app for testing, results displayed in <b>Mobile Secure Dashboard (SAST/DAST)</b> (dt needs apk file)	Mobile Protect - An SDK that sends alert data to dashboard, in <b>Active Protection</b>
Web App - (Provider App - Frontend - React.js)	Shows up in Web Secure in v1; best to check in API Secure in v2 - <a href="https://www.securetheorem.com/web/inventory/web-assets">https://www.securetheorem.com/web/inventory/web-assets</a> (currently only has dev site, will need to add prod site after launch and get dt user and login creds there) <b>DT needs to scrape the web app on their end</b>		
Survivor App and Provider App Backend (Serverless + Node.js API Gateway)	Dynamic testing of the APIs in the app; API protect vtap to allow traffic monitoring on nginx/glue edge/istio		

	ingress point (easier setup); SDK active protection (more involved setup)		
Static Code security test (SAST) for all repos (front and backend)	<a href="https://www.securetheorem.com/api/v2/security/sast">https://www.securetheorem.com/api/v2/security/sast</a>		

- Dashboard: <https://www.securetheorem.com/mobile-secure/v2/dashboard>
  - Inventory: This where we can see an overview of the two different apps. Helpful for verifying pre-production testing that will be deployed to production, and tracking which version is live and scanned in the app stores.
  - Security: SAST - The section does code analysis. Every commit can have issues assigned to them. DAST- The binary is run on real devices, real testing on the app. DAST issues are a bit more helpful because they are realtime issues. This can be linked directly to JIRA, and they provide code snippets to identify the problem as well as solution suggestions.
    - Privacy: Able to see where 3rd party code is leaking information or data, can make decisions on what to keep and/or not
  - Active Protection: SDK that is defensive in nature, see what attacks are being made on the application, can see logs for all attacks.
    - Threats: Can see all of the vulnerabilities. Can track or actually block through the SDK in the application.
    - Spyware: Can see spyware kits and see which devices are vulnerable (and how many). Can also block the app on those compromised devices.
    - OSS/SDK Firewall: Prevent 3rd party libraries from stealing and scraping data
- SDK: This is set up on mobile: Mobile Protect
  - On the web, it is the Node.js API Protect.
  - Information [here](#). Added mobile-protect to project
- Security Tests
  - Survivor Mobile App [Results](#)
  - Provider Web App - SAST scan is initiated by empty commit to main

## Design Assets

1. Designs in Figma:

<https://www.figma.com/file/KPrTqLBRowfLqplt6KcRcl/ReferAll-Designs?node-id=911%3A6684>

## Logs

All lambda functions and their corresponding API endpoints have maintained and stored logs in the AWS console under the CloudWatch service.

Bugsnag is used to catch routine crashes and bugs. [Bugsnag dashboard](#) user is jakki@ac. Password is the common ReferAll one.

## Analytics

### ResourceFull Analytics Tracking Plan

- Google Analytics dashboard can be accessed here:
- Mixpanel analytics dashboard can be accessed here:

## Build Tracking

### ReferAll Build Tracker

## API Endpoints

### **Provider Application**

- Admin
  - API: **/addproviderorg**
    - Method: POST
    - Sending data to the backend to add a new provider user and a new organization to the database, or to update the values for a particular provider
  - API: **/getproviderorg**
    - Method: POST
    - Get list of all providers added by that admin user
- Login
  - API: **/getuserprovider**
    - Method: POST
    - Once the user is Authenticated, run this API call to get the provider name (the organization that this user is associated with). The provider name and the user's email will be stored in Global Context

- Services Providers Dashboard
  - API: **/getallproviders**
    - Method: GET
    - When the user lands on the Service Provider dashboard, this API receives all of the provider information from the provider\_table
  - API: **/addprovidersearch**
    - Method: POST
    - When a user creates a new search based on the filter menu, store this search configuration as well as the Name the user gives the search.
  - API: **/getprovidersavedsearches**
    - Method: POST
    - Based on the user's email address, get all of the saved searches that the user has created from the provider\_saved\_searches table
- People Dashboard
  - API: **/addproviderpeoplenotes**
    - Method: POST
    - Add or update the participant's status (ie, receiving services, etc).
  - API: **/getintakes**
    - Method: POST
    - Get the list of all participants who have submitted an intake to this provider
  - API: **/getclients**
    - Method: POST
    - Get the list of all participants who are actively receiving services from this provider
  - API: **/getparticipantsavedsearches**
    - Method: POST
    - Get the saved searches for this participant
  - API: **/getpeopledrawernotifications**
    - Method: POST
    - Get all status and notification updates for a user when logging into the dashboard
- Profile Basics
  - API: **/addproviderprofilebasics**
    - Method: POST
    - When a user creates a new profile, store this information as well as the name the user gives the search.
  - API: **/addproviderprofiledetails**
    - Method: POST
    - When a user creates a new profile/details, this information is stored based on organization name.

- API: **/addproviderprofilenonpublicdata**
  - Method: POST
  - When a user creates a new profile/details, this information is stored based on organization name.
- API: **/getproviderprofiledata**
  - Method: POST
  - Retrieve provider data based on provider\_id (or provider\_name)
- Settings
  - API: **/updateproviderusername**
    - Method: POST
    - User can update their username. This is not tied to the organization/provider information, only to the specific user
- Help
  - API: **/addhelpform**
    - Method: POST
    - User can submit the help/feedback form data

## Survivor Application

- Login
  - API: **/addnewuser**
    - Method: POST
    - Adding a new user to the backend database
  - API: **/signinuser**
    - Method: POST
    - Getting user participant id. Check the current user in Cognito.
- Home
  - API: **/getappdata**
    - Method: GET
    - Gets the information for the homedashboard
  - API: **/getuserdata**
    - Method: POST
  - API: **/updateusernotification**
    - Method: POST
    - This API updated the red dot from the cards of participant status in the survivor app
- Service Providers
  - API: **/updateuserfavorites**
    - Method: POST
    - This API updates the favorites array of provider in the survivor\_user\_table
- Profile
  - API: **/updateuserprofileinfo**

- Method: POST
  - API: /deleteuseraccount
    - Method: POST
    - This API deletes the user from the following tables:  
provider\_participant\_intake, survivor\_general\_form,  
participant\_survivor\_connection, survivor\_intake\_form\_inprogress  
survivor\_intake\_form, and survivor\_user\_table
- General Form
  - API: /getgeneralform
    - Method: POST
    - This API retrieves data from the survivor\_general\_form table
  - API: /deletegeneralform
    - Method: POST
    - Users are able to delete this row from the table survivor\_general\_form table
  - API: /addgeneralform
    - Method: POST
    - This API sends a form to the survivor\_general\_form table. Users are able to update these values.
- Intake Form
  - API: /deleteintakeform
    - Method: POST
    - Users are able to delete their intake form with the delete button.
  - API: /getintakeform
    - Method: POST
    - This API gets the intake form from a specific user and provider. This is tied to participant id and provider id.
  - API: /addintakeform
    - Method: POST
    - Sends an intake form to the service provider selected by the user. This information doesn't update.
- Feedback
  - API: /survivorfeedback
    - Method: POST
    - The user can send a message to [referral@anniecannons.com](mailto:referral@anniecannons.com) about how the app can be improved