SD502783

Web App Development with Autodesk Forge Data Exchange and AWS Amplify

Jon Clark, Sr. Partner Solutions Architect
Amazon Web Services (AWS)

Enrico Chionna, BIM Technology Manager
Amazon Web Services (AWS)

Learning Objectives

1. Understand the use of Autodesk Forge Data Exchange
2. Integrate disparate data into a single application
3. Build an application using AWS and Autodesk Forge
4. Identify new use cases for Autodesk Forge Data Exchange

Description

Learn how to combine Autodesk Forge and AWS services to easily create a modern web application. This class will dive deep into the use cases for Autodesk Forge Data exchange and show how to export subsets of data from a Revit model. This data will then be joined with IoT data from typical building sensors and displayed in a web application built using the AWS Amplify service, which makes it easy to create a web app.

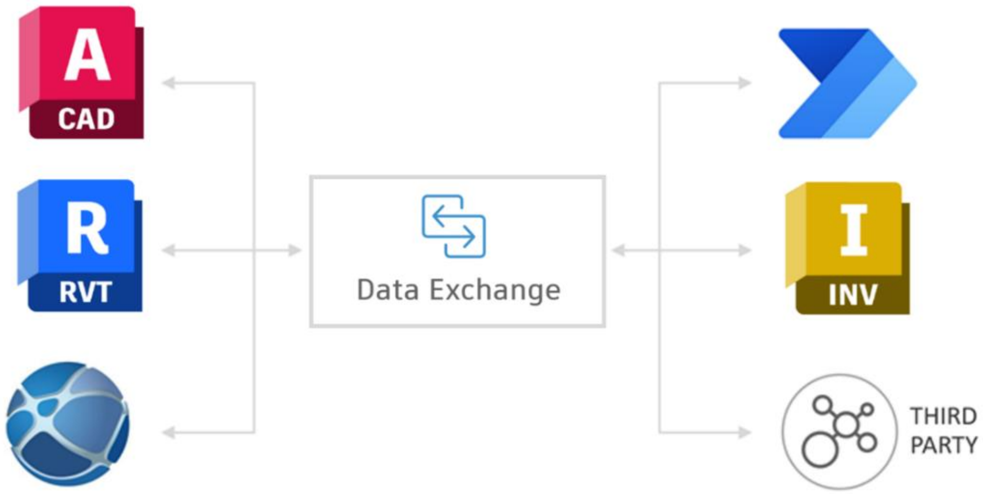
Contents

Autodesk Forge Data Exchange

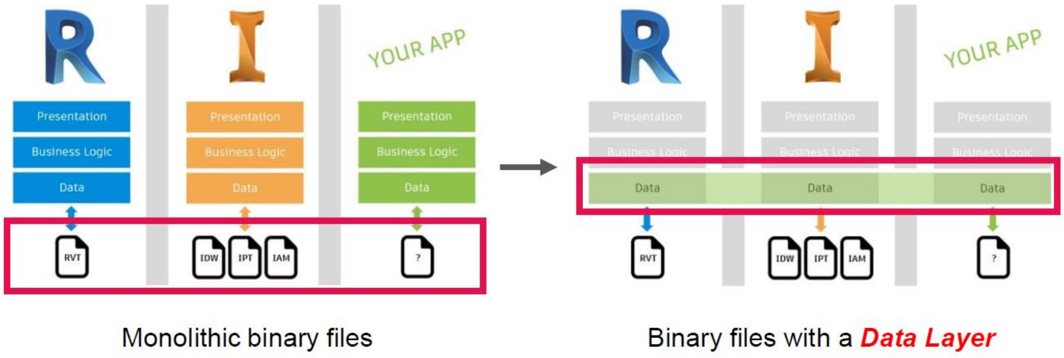
What is a Data Exchange?

Data Exchanges give you a way to share portions of your data (e.g., a Revit model) with collaborators. You can think of Data Exchanges as a sort of Amazon Locker. Amazon provides customers with the item they need, delivered directly to the locker they have access to, and they grab the item in a secure way at their leisure. You can play either role – producer or consumer of the item – or both!

Here is a simple diagram showing how Data Exchange can be used to exchange design information between Autodesk and third-party applications:



Forge Data Exchange works on a new data layer incorporated into binary files saved from Autodesk applications:



How does Data Exchange work with Forge API?

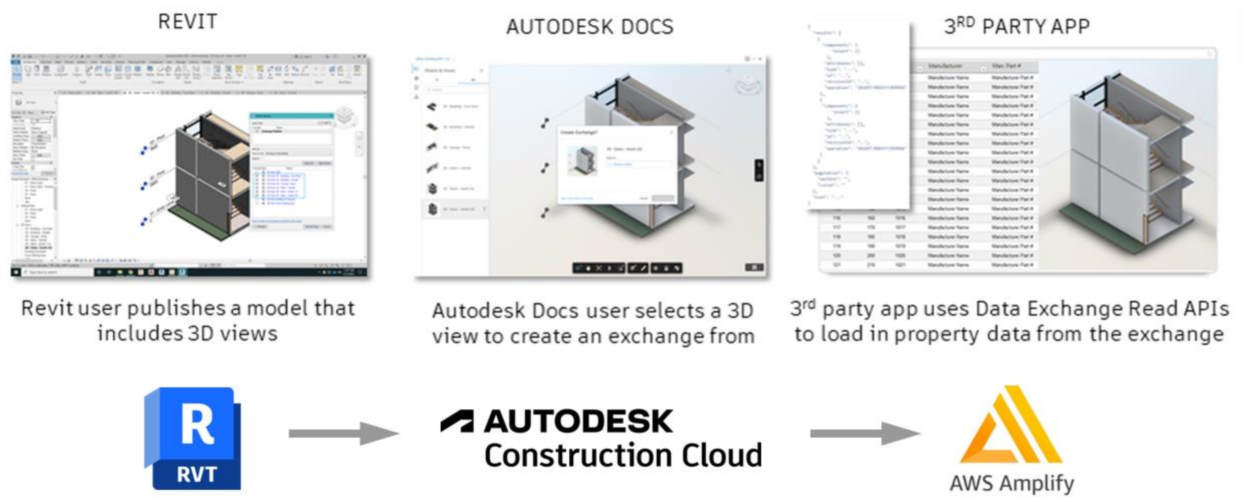
Individuals and companies around the globe work across disciplines and tools to create the awesome products, buildings, and experiences of today and those of the future. Collaborating seamlessly across those project teams and apps – many developed by Autodesk – has become more important than ever. Forge Data Exchanges help you unlock your data and give you the flexibility to share the right bits with the right stakeholders at the right time – no matter the app or industry you work in.

Example Workflow

Consider how an architecture firm – say, designing a school building in Revit – might interact with the building manager once the school is built. The building manager wants to map all the lights in and outside the building to a 3D model of the building in order track the usage and status of each light which is IoT enabled. When the building manager comes to the architect to ask for a 3D model of the building, the architect might not want to share the entire school building, perhaps to protect IP (Intellectual Property), or because the model might simply be too large for the use-case the building manager has in mind.

Enter **Autodesk Data Exchange**. Both parties can leverage Data Exchanges to achieve their perspective goals. The architect can release only the information needed for the building manager, and the building manager gets exactly what is needed for the building lights application. Exchanges act as neutral secure containers for subsets of data and can be shared with any number of apps connected into the ecosystem. In this example, the architect can designate the subset of data they want to share with the building manager – i.e., the Data Exchange. On their end, the building manager can consume that data in the app they use, in this case AWS Amplify. Moreover, as the architect updates their model, exchanges derived from that model automatically update, allowing the building manager to take in the changes made in the upstream design process.

As an example, a typical Data Exchange workflow might start with Revit:


REVIT
AUTODESK DOCS
3RD PARTY APP
Revit user publishes a model that includes 3D views
Autodesk Docs user selects a 3D view to create an exchange from
3rd party app uses Data Exchange Read APIs to load in property data from the exchange
R
RVT
AUTODESK
Construction Cloud
AWS Amplify

The Revit file is stored in Autodesk Docs which is part of the Autodesk Construction Cloud. Here, we can create Data Exchanges of the Revit model.

And since Data Exchange is part of Forge API, we can reference information in the Data Exchange and consume it in our 3rd party app. In this case, we are using AWS Amplify to build our app around the Autodesk Data Exchange.

What is AWS?

Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform, offering over 200 fully featured services from data centers located around the world.

Today, there are millions of businesses running on AWS to lower costs, become more agile, and innovate faster. There are many benefits to running your applications on the AWS platform, such as:

Security
At AWS, security is priority #1 and supports 98 security standards and compliance certifications. And, when you run on AWS, you run in a shared security model, where AWS takes care of security of the cloud, and you take care of security in the cloud, enabling you to put more emphasis on securing your sensitive data.

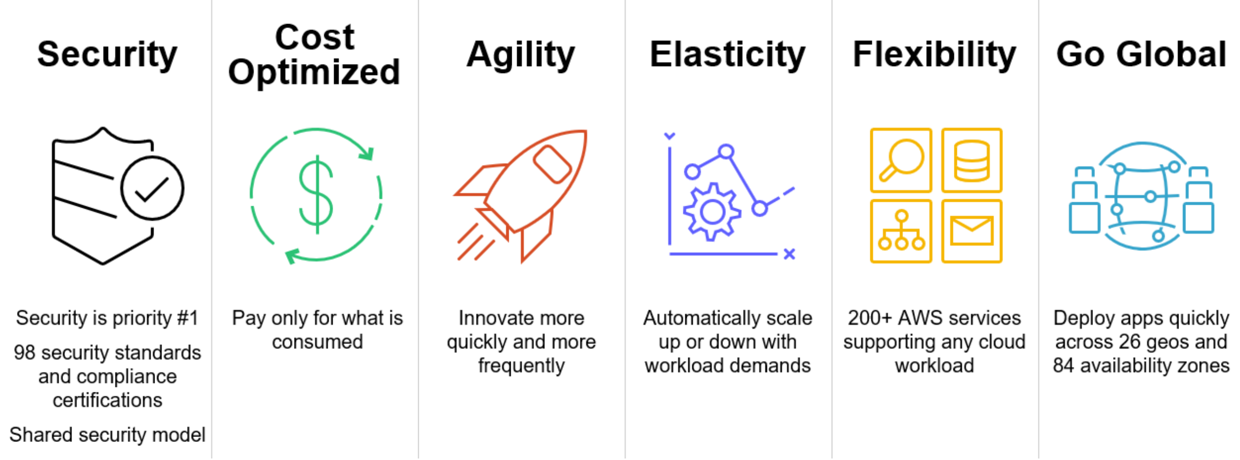
Cost Optimized
No need to use capital expense on local hardware. Pay only for what you use.

Agility
With having optimized security, and optimized costs, AWS gives you the agility to innovate more quicky and more frequently than ever before.

Elasticity
Plus, AWS is elastic. The system automatically scales up or down resources depending on workload demands in real-time.

Flexibility
You also have more flexibility with access to 200+ AWS services supporting virtually any cloud workload or use-case.

Go Global
And last, you can deploy your applications globally in seconds. Quickly and easily deploy your apps across 26 geos around the world, including 84 availability zones, all running on AWS's own multiple redundant fiber-optic network.



Security	Cost Optimized	Agility	Elasticity	Flexibility	Go Global
Security is priority #1 98 security standards and compliance certifications Shared security model	Pay only for what is consumed	Innovate more quickly and more frequently	Automatically scale up or down with workload demands	200+ AWS services supporting any cloud workload	Deploy apps quickly across 26 geos and 84 availability zones

What is AWS Amplify?

AWS Amplify is a set of purpose-built tools and features that lets front-end and full-stack web and mobile developers quickly build feature-rich, full-stack applications on AWS, with the flexibility to leverage the breadth of AWS services as their use cases evolve. With Amplify, you can configure a web or mobile app backend, visually build a web front-end UI, connect the two together, and easily manage app content outside the AWS console. You and your team are able to ship faster and scale effortlessly, without deep cloud expertise.

A typical web application built with AWS Amplify is a full-stack serverless app consisting of a **backend** built with cloud resources such as **GraphQL** or REST APIs, file and data storage, and a **frontend** built with single page application frameworks such as React, Angular, or Vue.

Amplify Hosting offers a fully managed service for deploying and hosting web apps and static websites using **Amazon CloudFront**, a content delivery network (CDN) with hundreds of points of presence globally, and with built-in CI/CD workflows that accelerate your application release cycle.

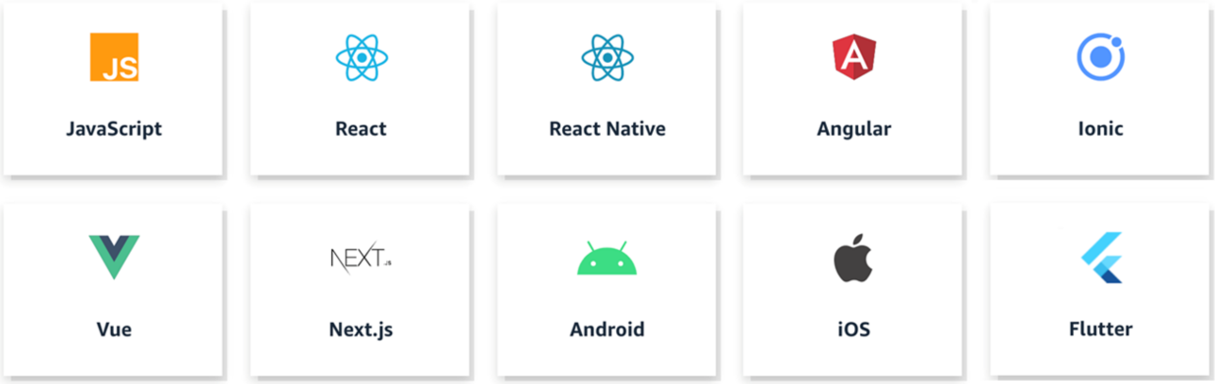
AWS Amplify supports the following web frameworks and mobile platforms:

Web:

- JavaScript
- React
- Angular
- Vue
- Next.js

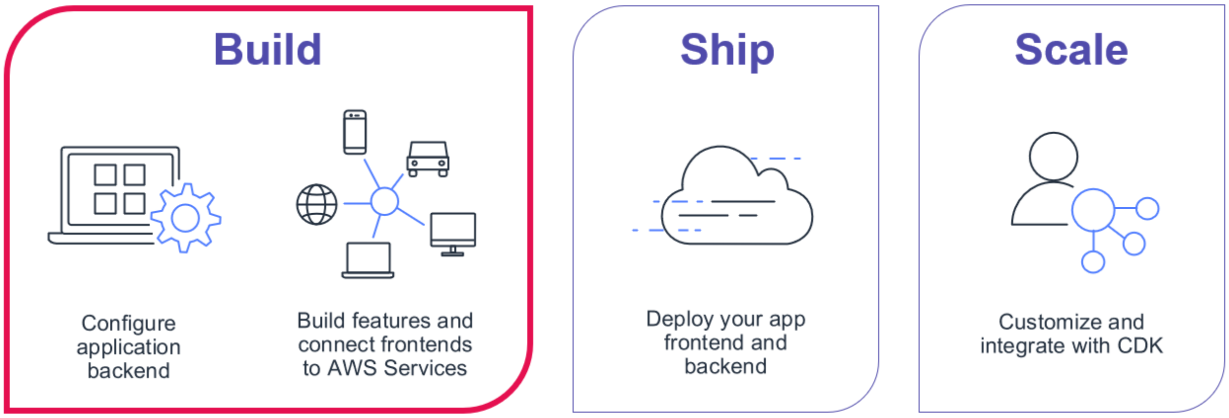
Mobile:

- Android
- iOS
- React Native
- Ionic
- Flutter



AWS Amplify: Build, Ship, and Scale

AWS Amplify provides the tools you need to build, ship, and scale your app!

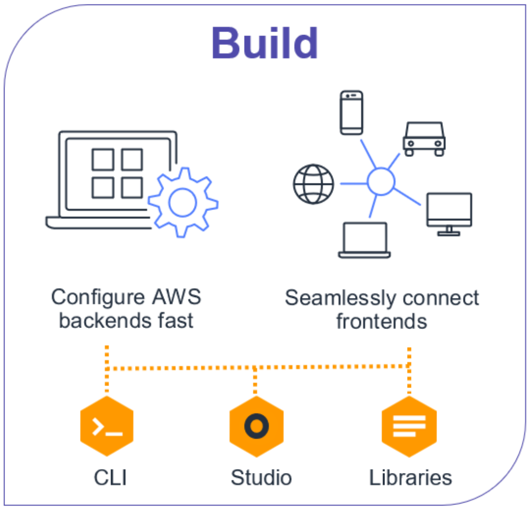

Build
Configure application backend
Build features and connect frontends to AWS Services
Ship
Deploy your app frontend and backend
Scale
Customize and integrate with CDK

BUILD
Quickly create and configure the backend of your app, and connect the backend to the UI frontend.

SHIP
Deploy your application in a matter of seconds.

SCALE
Extend your use cases with direct access to 175+ features of the AWS CDK (cloud development kit).

BUILD with AWS Amplify



Build

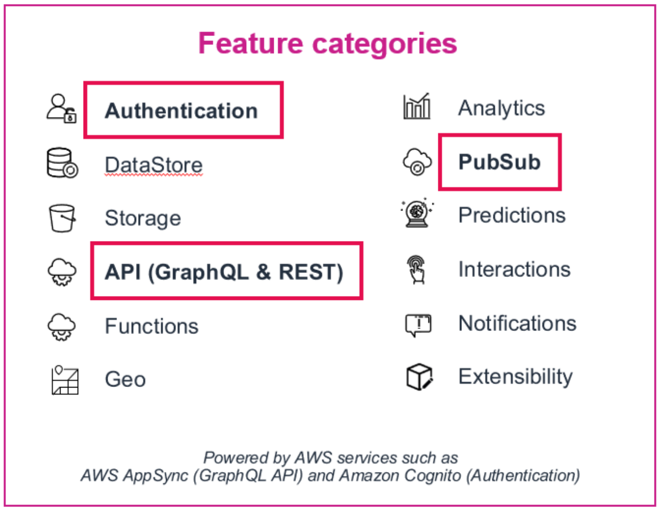
Configure AWS backends fast

Seamlessly connect frontends

CLI

Studio

Libraries



Feature categories

- Authentication
- DataStore
- Storage
- API (GraphQL & REST)
- Functions
- Geo
- Analytics
- PubSub
- Predictions
- Interactions
- Notifications
- Extensibility

Powered by AWS services such as
AWS AppSync (GraphQL API) and Amazon Cognito (Authentication)

Configure backends fast
Use Amplify Studio and CLI's intuitive workflows to set up scalable AWS backends with authentication, storage, data, and other common use cases.

Seamlessly connect front ends
Use the Amplify libraries in your web, Android, and iOS apps to connect to new and existing AWS resources in just a few lines of code.

Studio

Amplify Studio
Amplify Studio provides a visual interface to configure and maintain your app backend outside the AWS console, build a frontend UI, and quickly bind the two together. Easily configure your backend data model, authentication, authorization, and more. All backend resources created by Amplify Studio are usable with the Amplify CLI.

CLI

Amplify CLI
The Amplify Command Line Interface (CLI) is a toolchain to configure and maintain your app backend from your local desktop. Configure cloud functionality using the CLI's interactive workflow and intuitive use cases such as auth, storage, API. Test features locally and deploy multiple environments. All configured resources are available to customers as infrastructure-as-code templates enabling better team collaboration and easy integration with Amplify's CI/CD workflow.

Libraries

Libraries
JavaScript >> **iOS** >> **Android** >> **Flutter** >> – AWS Amplify offers use case-centric open-source libraries in the Amplify Framework to build cloud powered mobile and web apps. Amplify libraries are powered by AWS services and can be used with new backends created with the Amplify CLI and admin UI, or your existing AWS backend.

Authentication
Create seamless on-boarding flows with a fully-managed user directory and pre-built sign-up, sign-in, forgot password, and multi-factor auth workflows. Amplify also supports login with a social provider such as Facebook, Google Sign-In, or Login with Amazon and provides fine grained access control to mobile and web applications. Powered by Amazon Cognito.

DataStore
Use a multi-platform (iOS/Android/React Native/Web) on-device persistent storage engine that automatically synchronizes data between mobile/web apps and the cloud, powered by GraphQL. DataStore provides a programming model for leveraging shared and distributed data without writing additional code for offline and online scenarios, which makes working with distributed, cross-user data just as simple as working with local-only data. Powered by AWS AppSync.

Analytics
Understand the behavior of your web, iOS or Android users. Use auto tracking to track user sessions and web page metrics or create custom user attributes and in-app metrics. Get access to real time data stream and analyze the data for customer insights and build data driven marketing strategies to drive customer adoption, engagement, and retention. Powered by Amazon Pinpoint and Amazon Kinesis.

API
Make secure HTTP requests to GraphQL and REST endpoints to access, manipulate, and combine data from one or more data sources such Amazon DynamoDB, Amazon Aurora Serverless, and your custom data sources with AWS Lambda. Amplify enables you to easily build scalable applications that require real-time updates, local data access for offline scenarios, and data synchronization with customizable conflict resolution when devices are back online. Powered by AWS AppSync and Amazon API Gateway.

Functions
Add a Lambda function to your project which you can use alongside a REST API or as a data source in your GraphQL API using the @function directive in the Amplify CLI. You can update the Lambda execution role policies for your function to access other resources generated and maintained by the CLI, using the CLI. Amplify CLI enables you to create, test and deploy Lambda functions across various runtimes and once a runtime is selected, you can select a function template for the runtime to help bootstrap your Lambda function.

Geo
Add location-aware features like maps and location search to your JavaScript-based web app in minutes. Amplify Geo includes pre-integrated map UI components (based on the popular MapLibre open-source library) and it updates the Amplify Command Line Interface (CLI) tool with support for provisioning all required cloud location services. You can customize embedded maps to match your app’s theme, or choose from many community-developed MapLibre plugins for more flexibility and advanced visualization options. Powered by Amazon Location Service.

Interactions

Build interactive and engaging conversational bots with the same deep learning technologies that power Amazon Alexa with just a single line of code. Create great user experiences through chat bots when it comes to tasks such as automated customer chat support, product information/recommendations or streamlining common work activities etc. Powered by Amazon Lex.

Predictions

Enhance your app by adding AI/ML capabilities. You can easily achieve use cases like text translation, speech generation from text, entities recognition in image, interpretation of text, and transcribing text. Amplify enables simplified orchestration of advanced use cases like uploading images for automatic training and using GraphQL directives for chaining multiple AI/ML actions. Powered by Amazon Machine Learning services, such as Amazon SageMaker.

PubSub

Pass messages between your app instances and your app's backend creating real-time interactive experiences. Amplify provides connectivity with cloud-based message-oriented middleware. Powered by AWS IoT services and Generic MQTT Over WebSocket Providers.

Push notifications

Improve customer engagement by using marketing and analytics capabilities. Leverage customer insights to segment and target your customers more effectively. You can tailor your content and communicate through multiple channels including email, texts as well as push notifications. Powered by Amazon Pinpoint.

Storage

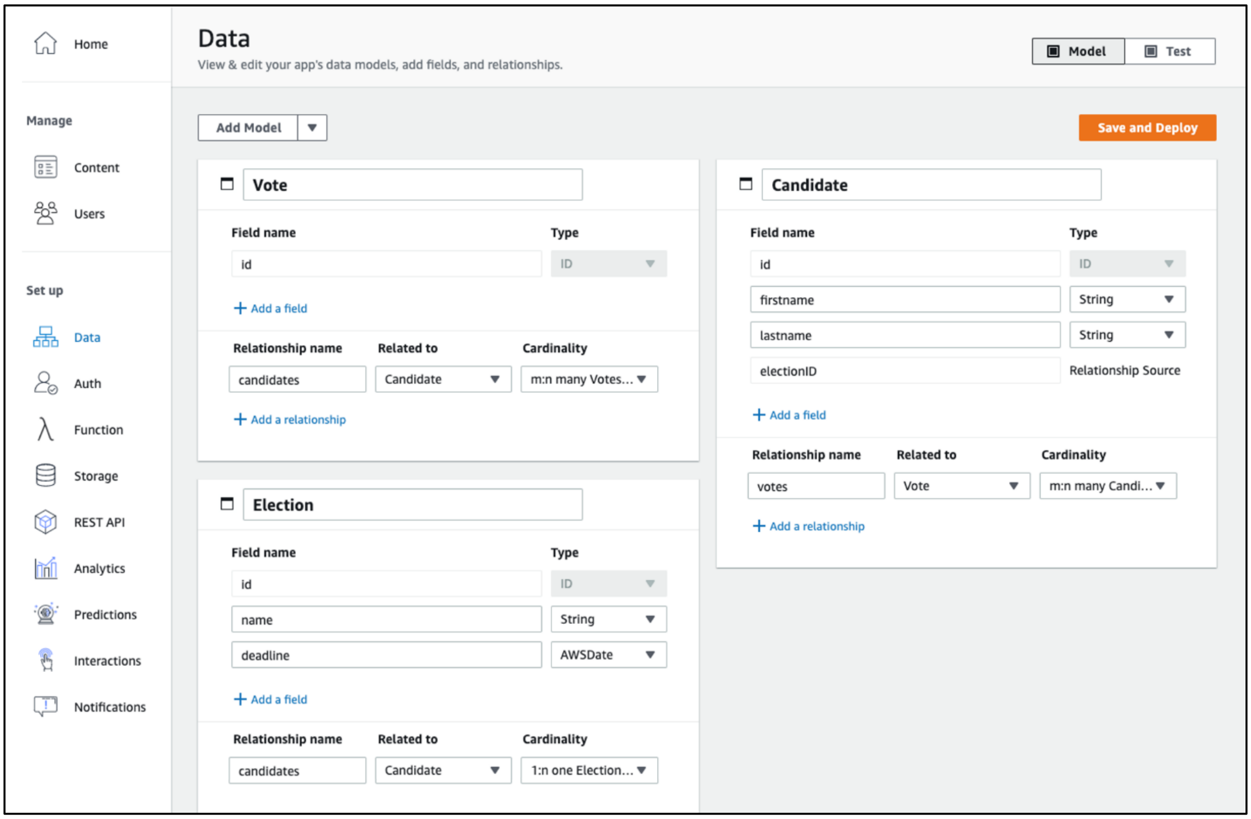
Store and manage user generated content such as photos, videos securely on device or in the cloud. The AWS Amplify Storage module provides a simple mechanism for managing user content for your app in public, protected or private storage buckets. Leverage cloud scale storage so that you can easily take your application from prototype to production. Powered by Amazon S3.

Extensibility

Amplify’s new extensibility features give developers the flexibility to customize their AWS backend and deployment capabilities. Reconfigure Amplify-generated backend resources to optimize for specific use cases, or modify Amplify's deployment operations to comply with your enterprise DevOps guidelines as your needs evolve.

BUILD with Amplify Studio

Configure the backend of your application VISUALLY from the AWS console.



Data

View & edit your app's data models, add fields, and relationships.

Vote

Field name	Type
id	ID

Relationship name	Related to	Cardinality
candidates	Candidate	m:n many Votes...

Election

Field name	Type
id	ID
name	String
deadline	AWSDate

Relationship name	Related to	Cardinality
candidates	Candidate	1:n one Election...

Candidate

Field name	Type
id	ID
firstname	String
lastname	String
electionID	Relationship Source

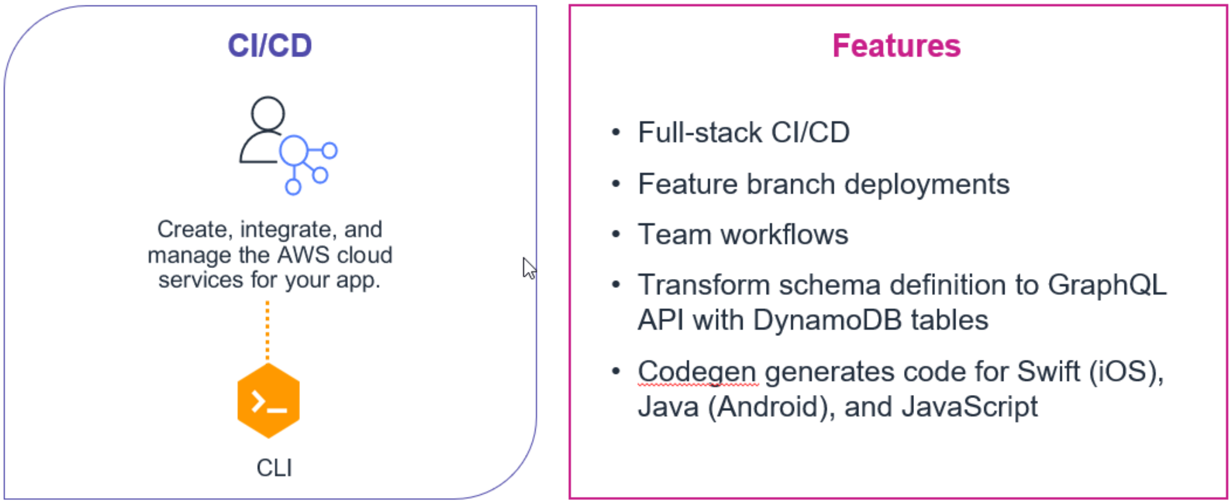
Relationship name	Related to	Cardinality
votes	Vote	m:n many Candi...

With Amplify Studio, you can:

- Model data with GraphQL API and DynamoDB tables
- Manage users, groups, file storage
- Configure authentication
- Extend with CLI toolchain to add functions

BUILD with Amplify CLI

Amplify CLI is a ***Unified toolchain and CI/CD for full-stack developers***.



CI/CD

Create, integrate, and manage the AWS cloud services for your app.

CLI

Features

- Full-stack CI/CD
- Feature branch deployments
- Team workflows
- Transform schema definition to GraphQL API with DynamoDB tables
- Codegen generates code for Swift (iOS), Java (Android), and JavaScript

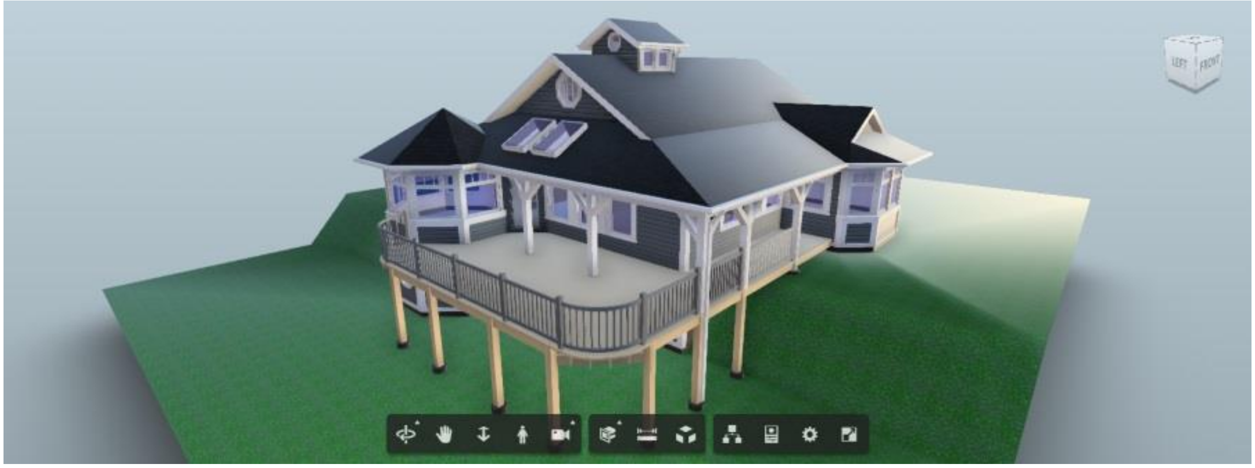
In addition to Amplify Studio, Amplify CLI gives you the ability to ***transform the schema definition to a GraphQL API running in AppSync and interact with DynamoDB***. This toolchain generates 100s of lines of boilerplate API resolver code, and client-side code that developers would have to hand-code without the Amplify CLI. If you want to make a change to your data structure, no problem, just update the schema definition and let Amplify generate the server-side and client-side code!

AWS Amplify – UI Development

For the UI development of our example AWS Amplify app, we used a few components:

Autodesk Forge Viewer

The Viewer is a WebGL-based, JavaScript library for 3D and 2D model rendering. 3D and 2D model data may come from a wide array of applications, such as **AutoCAD**, **Fusion 360**, **Revit**, and many more.



The Viewer communicates natively with the Model Derivative API to fetch model data, complying with its authorization and security requirements.

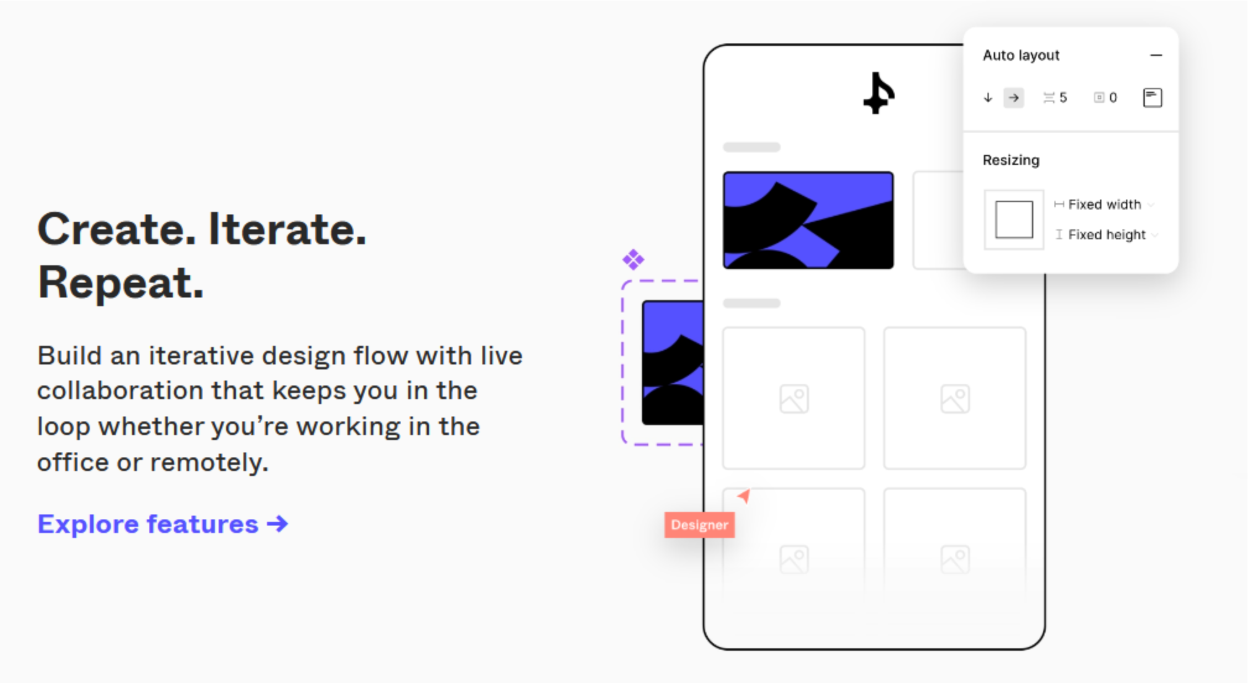
The Viewer requires a WebGL-canvas compatible browser:

- Chrome 50+
- Firefox 45+
- Opera 37+
- Safari 9+
- Microsoft Edge 20+
- Internet Explorer 11

Implement the Forge Viewer to display and interact with the Revit 3D model in the browser.

Figma

https://figma.com



Create. Iterate. Repeat.

Build an iterative design flow with live collaboration that keeps you in the loop whether you're working in the office or remotely.

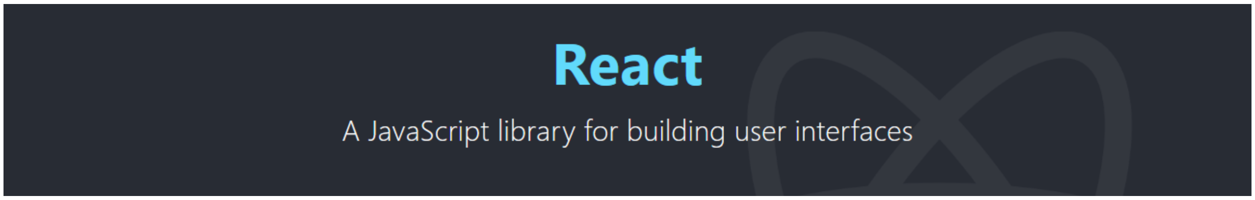
Explore features →

Figma is a free and modern interface design tool that helps you to create anything: websites, applications, logos, and much more.

Use Figma to create a wireframe mockup of your UI.

React

https://reactjs.org



React

A JavaScript library for building user interfaces

React is a powerful JavaScript UI library for create modern, composable user interfaces. It encourages the creation of reusable UI components that present data that changes over time.

React is Declarative

React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

Declarative views make your code more predictable and easier to debug.

React is Component-Based

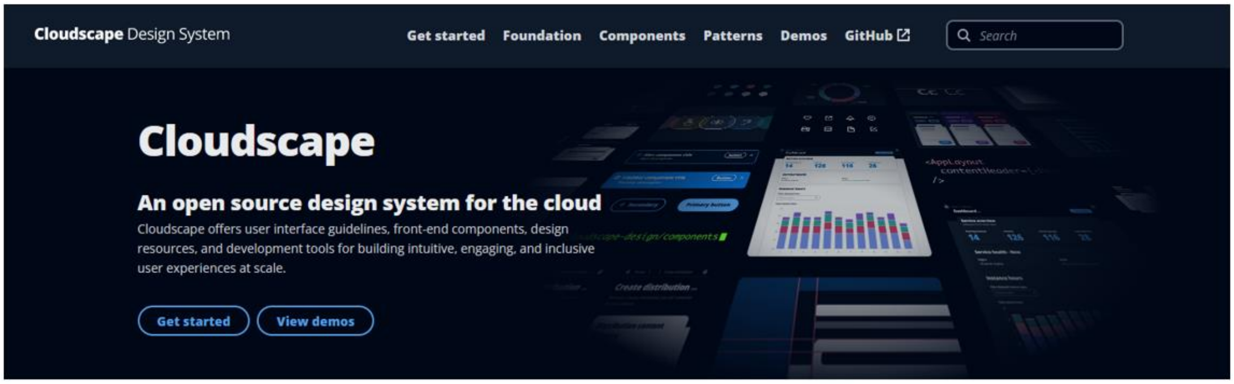
Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

Use React to create the UI framework of your web application.

Cloudscape

https://cloudscape.design

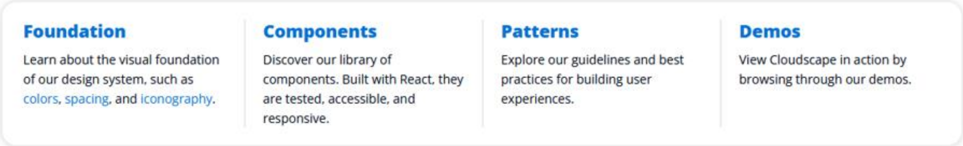


Cloudscape Design System

Cloudscape

An open source design system for the cloud

Cloudscape offers user interface guidelines, front-end components, design resources, and development tools for building intuitive, engaging, and inclusive user experiences at scale.

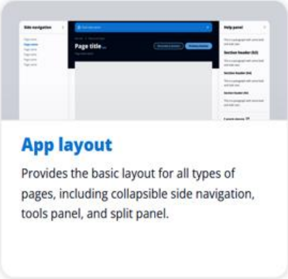


Foundation	Components	Patterns	Demos
Learn about the visual foundation of our design system, such as colors, spacing, and iconography.	Discover our library of components. Built with React, they are tested, accessible, and responsive.	Explore our guidelines and best practices for building user experiences.	View Cloudscape in action by browsing through our demos.

Cloudscape is ***an open-source design system for the cloud***. Cloudscape offers user interface guidelines, front-end components, design resources, and development tools for building intuitive, engaging, and inclusive user experiences at scale.

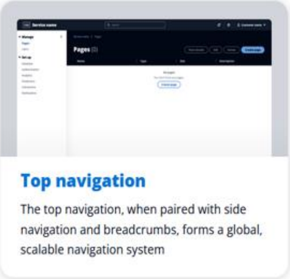
Cloudscape was built for and is used by Amazon Web Services (AWS) products and services. AWS created Cloudscape in 2016 to improve the user experience across web applications owned by AWS services, and also to help teams implement those applications faster. Since then, the Cloudscape system is continuous enhanced based on customer feedback and research.

> *Implement Cloudscape components to produce a consistent look and feel across your application. The main components we implemented are: App Layout, Top navigation, Side navigation, and Help panel.*



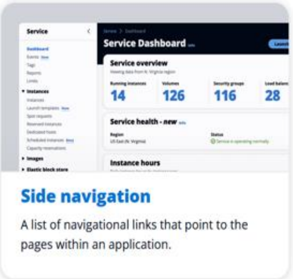
App layout

Provides the basic layout for all types of pages, including collapsible side navigation, tools panel, and split panel.



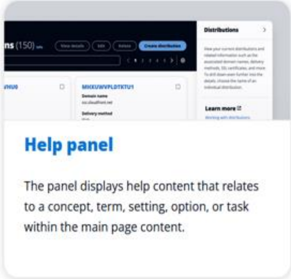
Top navigation

The top navigation, when paired with side navigation and breadcrumbs, forms a global, scalable navigation system



Side navigation

A list of navigational links that point to the pages within an application.



Help panel

The panel displays help content that relates to a concept, term, setting, option, or task within the main page content.

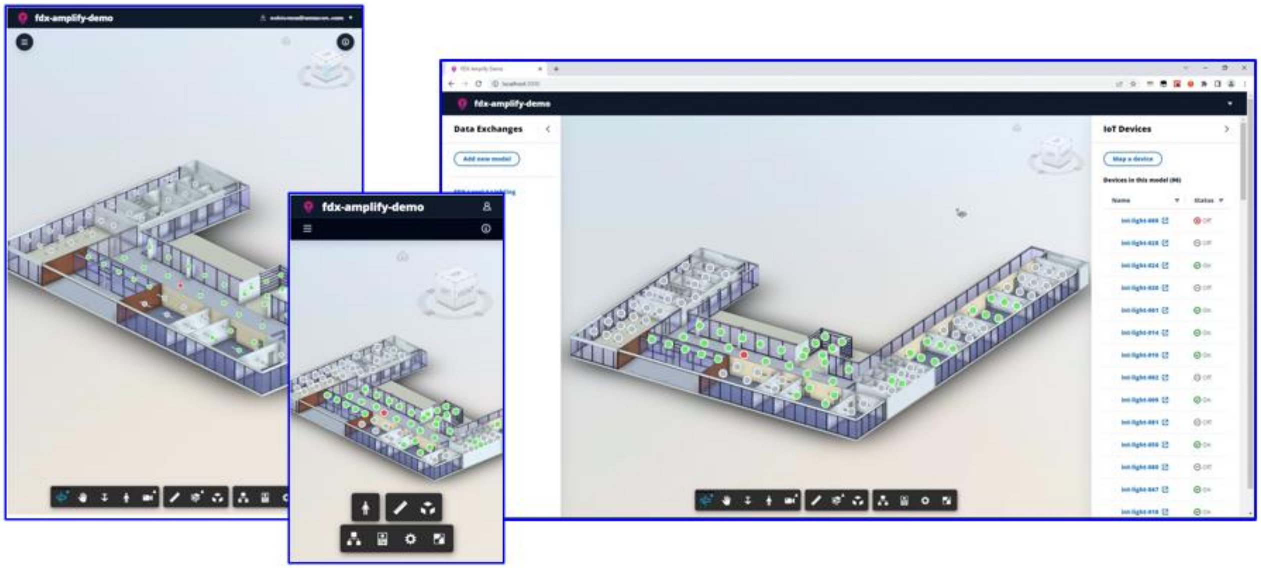
Single-Page vs. Progressive vs. Mobile Apps

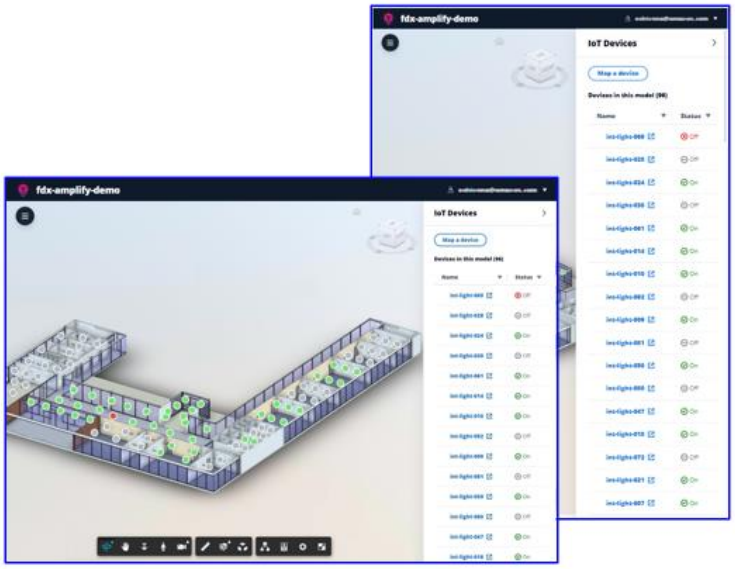
- **Single-Page Application** (SPA)
  - A website that loads content dynamically within a single page.
  - Runs within a single Document Object Model (DOM).
  - Requires network connection.
- **Progressive Web App** (PWA)
  - A web application designed to run as if it's a native mobile app.
  - Installable
  - Uses background workers that load cached data even without a network.
- **Native Mobile App**
  - Built to run on a specific mobile OS (iOS or Android).
  - Network connection not required.

	Single Page App	Progressive Web App	Native Mobile App
Platform	Any browser, any device	Any browser, any device	Device OS
Language	HTML, CSS, JavaScript	HTML, CSS, JavaScript	Mobile-specific
Building and updating	Faster	Faster	Slower
Hosting/distributing	http or https hosting	https hosting (required)	App Store
Search engine	None	SEO	ASO
Installable	No	Yes	Yes
Offline-able	No	Yes	Yes

Responsive Single Page Web App

- Prebuilt Cloudscape React components are responsive
- Responsive web apps adapt to different screen sizes and resolutions





App Development – Workflow

The general workflow to develop the web application in this example is as follows:

1. Create a wireframe mockup (or prototype) of the UI design.
2. Generate the backend using Amplify Studio or Amplify CLI
3. Create the frontend using a UI framework such as React
4. Connect the backend to the frontend of the app
5. Install UI packages such as Amplify UI and Cloudscape
6. Add and configure authentication
7. Configure UI components
8. Add GraphQL API and database
9. Update and deploy the GraphQL schema
10. Update the frontend with API queries
11. Create subscriptions to IoT devices
12. Add Forge API
13. Configure Data Exchange API

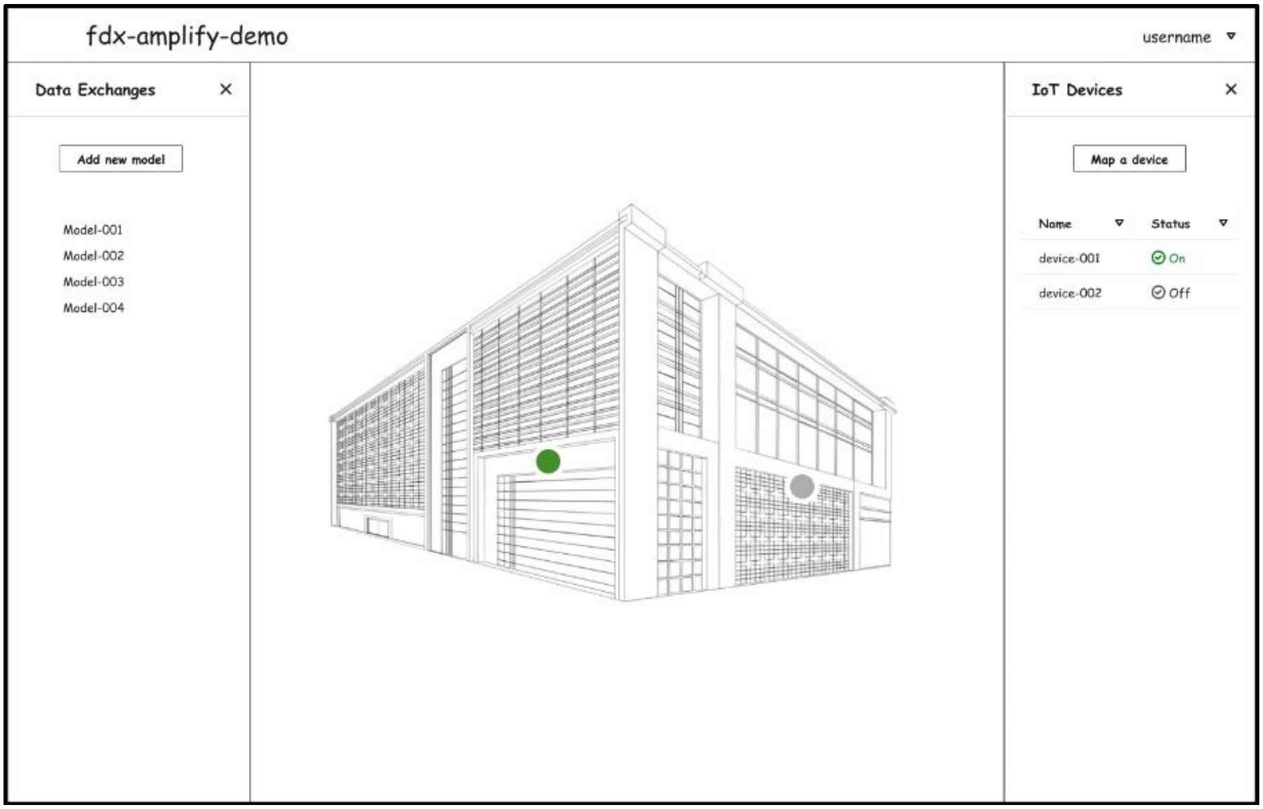
In this section we will dive deeper into setting up and integrating the backend of our app with React. We will create a web app with a GraphQL API that stores and retrieves items in a cloud database. In addition, we'll demonstrate how to authenticate users, communicate with our API, and integrate with Forge.

UI Prototype

There are generally two types of UI prototyping:

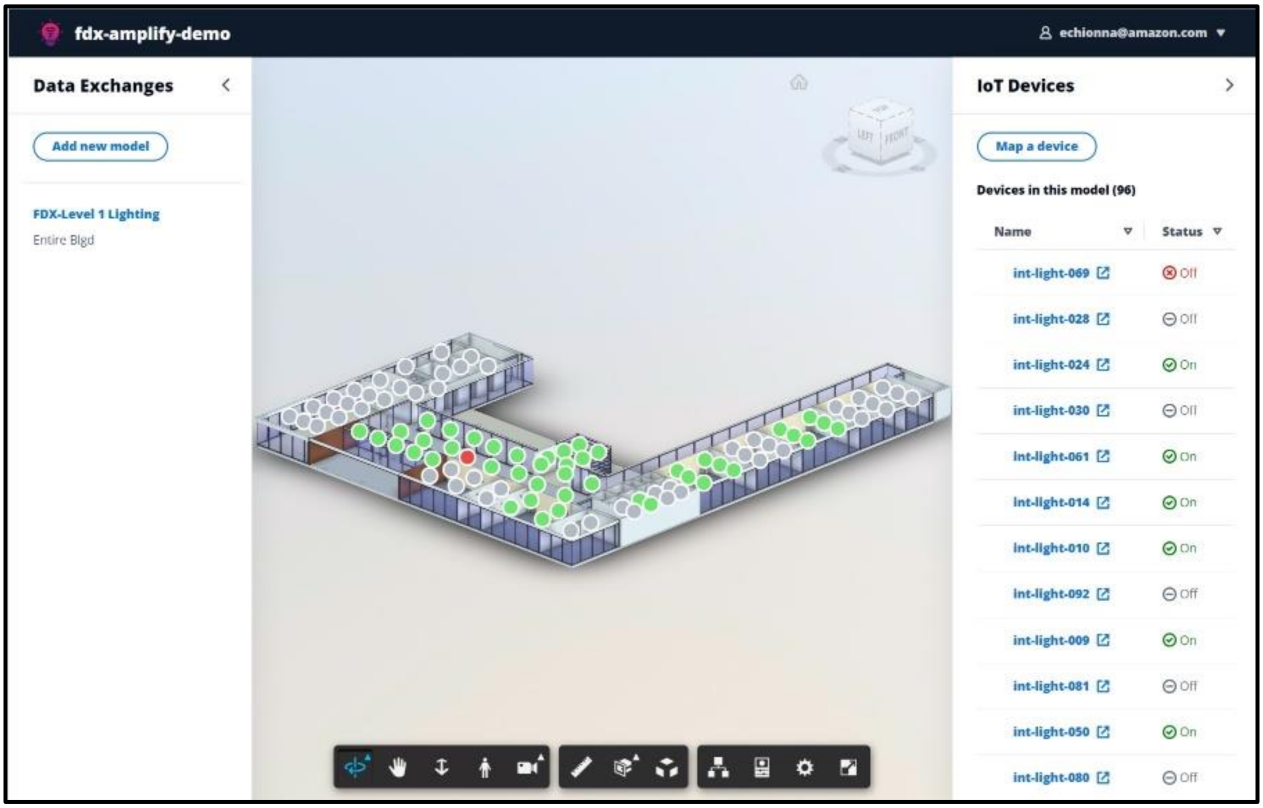
- **Low-fidelity prototype**: a rough representation of a design concept that helps designers validate the design early in the design process.
- **High-fidelity prototype**: an interactive prototype based on the low-fidelity prototype that simulates the real app's functionality and helps designers and/or end users to better understand the look and feel of the future app.

We created a UI prototype of the app using Figma to simulate main functions of the app, such as "Add new model", "Map a device", and viewing the model:



Name ▼	Status ▼
device-001	✓ On
device-002	✓ Off

Once the design achieves the desired user experience and satisfies all application requirements, we convert the prototype to a React app using Cloudscape React components. The Cloudscape design system helps facilitate a highly efficient workflow by providing a more seamless collaboration between designers and developers. Populating the converted low-fidelity UI prototype with Cloudscape components produces a high-fidelity version of the app:



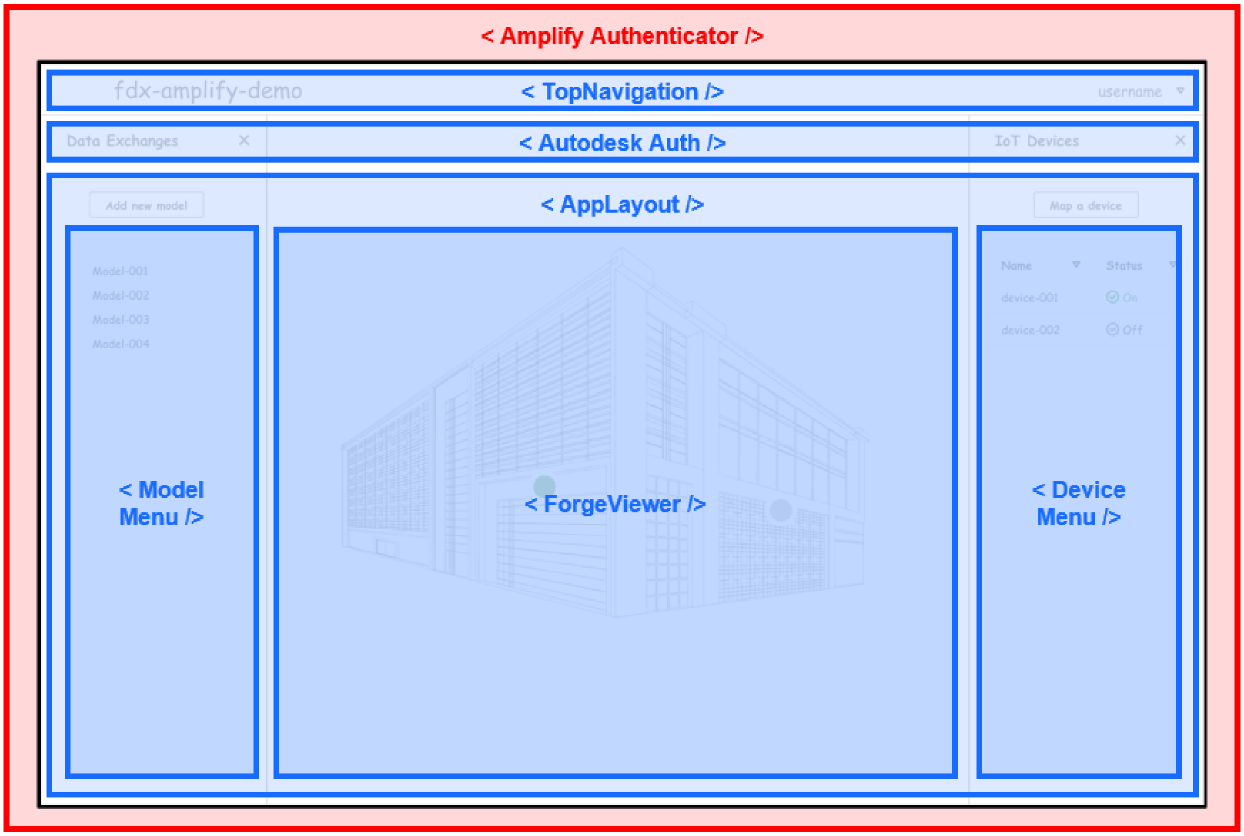
Devices in this model (96)

Name ▽	Status ▽
int-light-069	⊗ Off
int-light-028	⊖ Off
int-light-024	✓ On
int-light-030	⊖ Off
int-light-061	✓ On
int-light-014	✓ On
int-light-010	✓ On
int-light-092	⊖ Off
int-light-009	✓ On
int-light-081	⊖ Off
int-light-050	✓ On
int-light-080	⊖ Off

React UI Components

The frontend of our application leverages React components from two different libraries:

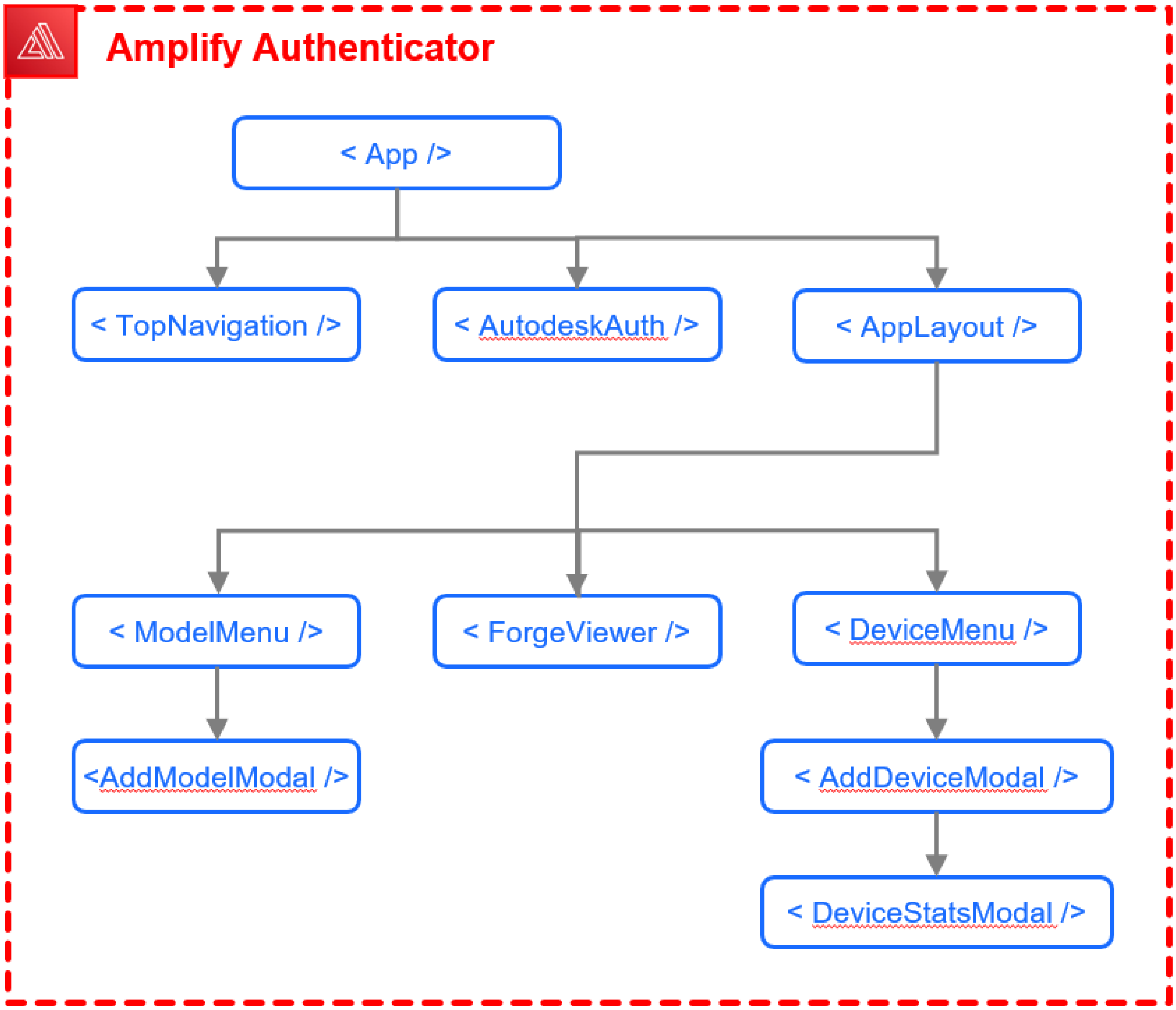
- AWS Amplify UI
- AWS Cloudscape



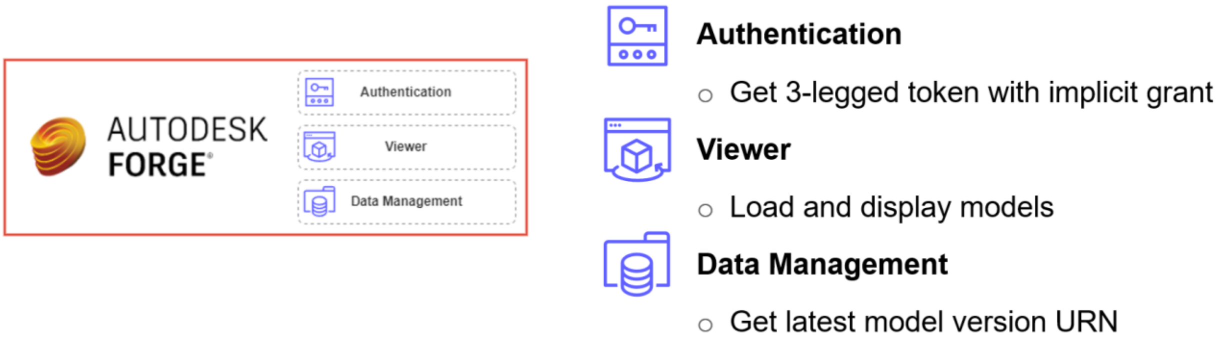
In Red is the Amplify Authenticator component from the AWS Amplify UI library, which wraps the entire React app and protects it with its fully automated authentication flow.

In Blue are the React components that use the Cloudscape design system library. These are components like the main App, the Top Navigation, the App Layout components. They are nested within each other, and make up the entire application and set the feel and look of the application.

The following diagram shows the hierarchical structure of our UI component framework:



Forge API



Authentication

- Get 3-legged token with implicit grant

Viewer

- Load and display models

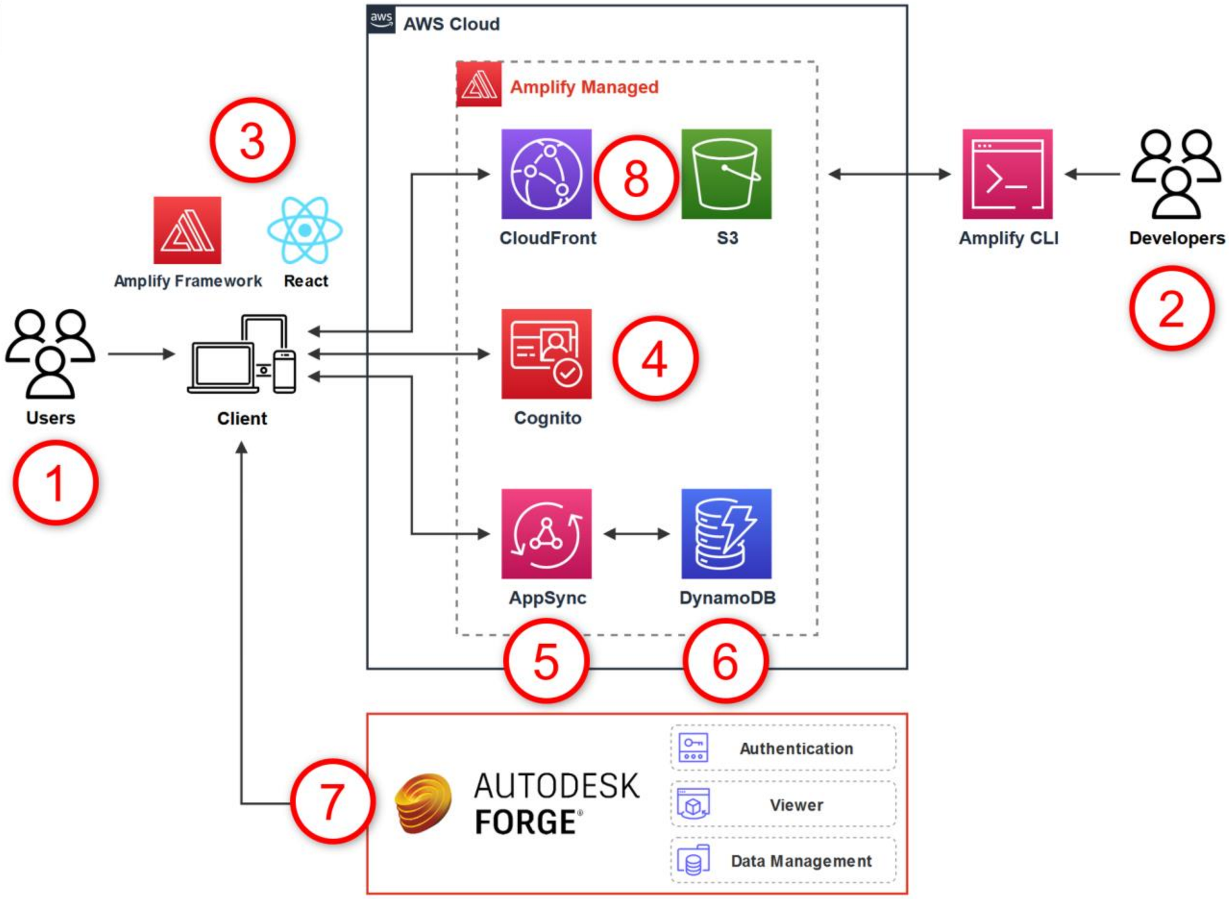
Data Management

- Get latest model version URN

Our Amplify app interacts with the Forge platform via 3 different APIs:

- **Authentication API**: used to generate a 3-legged access tokens and authorize our app to act on behalf of the user.
- **Viewer**: used to load and display a Revit models.
- **Data Management API**: used to retrieve the latest version of a model on BIM360 or ACC.

Application Architecture



The application architecture and development workflow is as follows:

1. Users (left) interact with the client application, and developers (right) work with AWS Amplify.
2. Developers install and configure Amplify CLI which initializes a new project and backend in the AWS cloud.
3. A new React app is then created and developer's setup is "Amplified" in the Frontend.
4. Amazon Cognito is added via Amplify CLI to manage user authentication.
5. A GraphQL API using AWS AppSync is added.
6. A database in DynamoDB is created.
7. Forge API is integrated into the architecture.
8. The application is then hosted and deployed using AWS Amplify, which uses CloudFront and S3. This step is performed at a later time and is outside the scope of this demonstration.

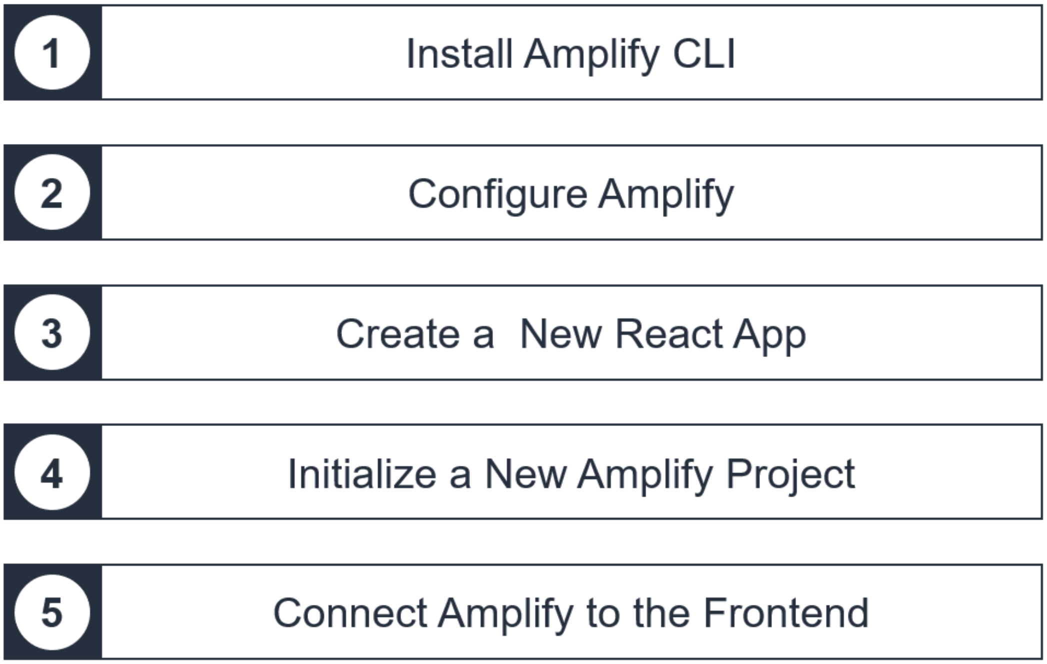
Prerequisites

Node.js v14.x or later
AWS Account
Npm v6.14.4 or later
Autodesk Account
Git v2.14.1 or later
Forge Account

The following items are needed to develop our application:

- Node.js
- Npm
- Git
- An AWS account. Note that there are no upfront charges or any term commitments, and signing up gives you immediate access to the AWS Free Tier.
- An Autodesk account to access models on the Autodesk Construction Cloud (ACC).
- A Forge account to access the API.

Steps to Create an Amplify Project



1 Install Amplify CLI

2 Configure Amplify

3 Create a New React App

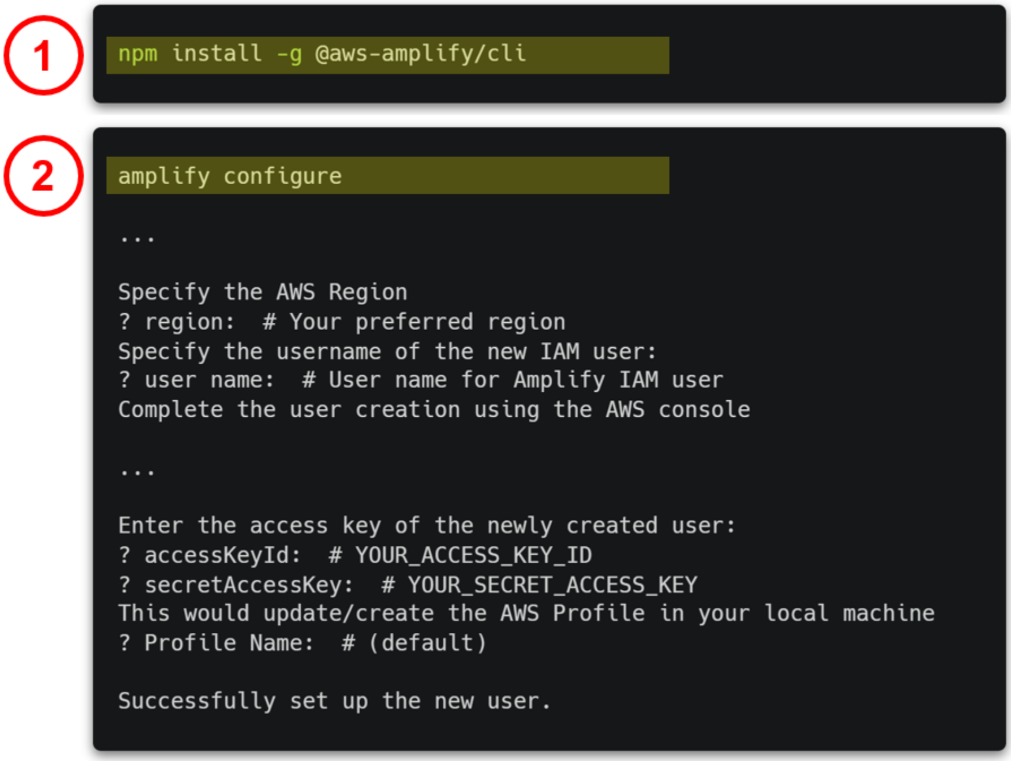
4 Initialize a New Amplify Project

5 Connect Amplify to the Frontend

Five steps are needed to create an Amplify project:

1. Install Amplify CLI
2. Configure Amplify
3. Create a New React App
4. Initialize a New Amplify Project
5. Connect Amplify to the Frontend

1. **Install Amplify CLI**
2. **Configure Amplify**



1

```
npm install -g @aws-amplify/cli
```

2

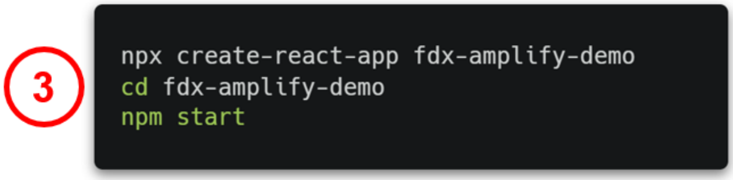
```
amplify configure

...

Specify the AWS Region
? region:  # Your preferred region
Specify the username of the new IAM user:
? user name:  # User name for Amplify IAM user
Complete the user creation using the AWS console

...

Enter the access key of the newly created user:
? accessKeyId:  # YOUR_ACCESS_KEY_ID
? secretAccessKey:  # YOUR_SECRET_ACCESS_KEY
This would update/create the AWS Profile in your local machine
? Profile Name:  # (default)

Successfully set up the new user.
```

Install and configure Amplify with just two lines of code:

```
npm install -g @aws-amplify/cli
amplify configure
```

The CLI will guide you through the process of creating a new IAM user that will be used by the Amplify CLI to add AWS services.

3. Create a New React App



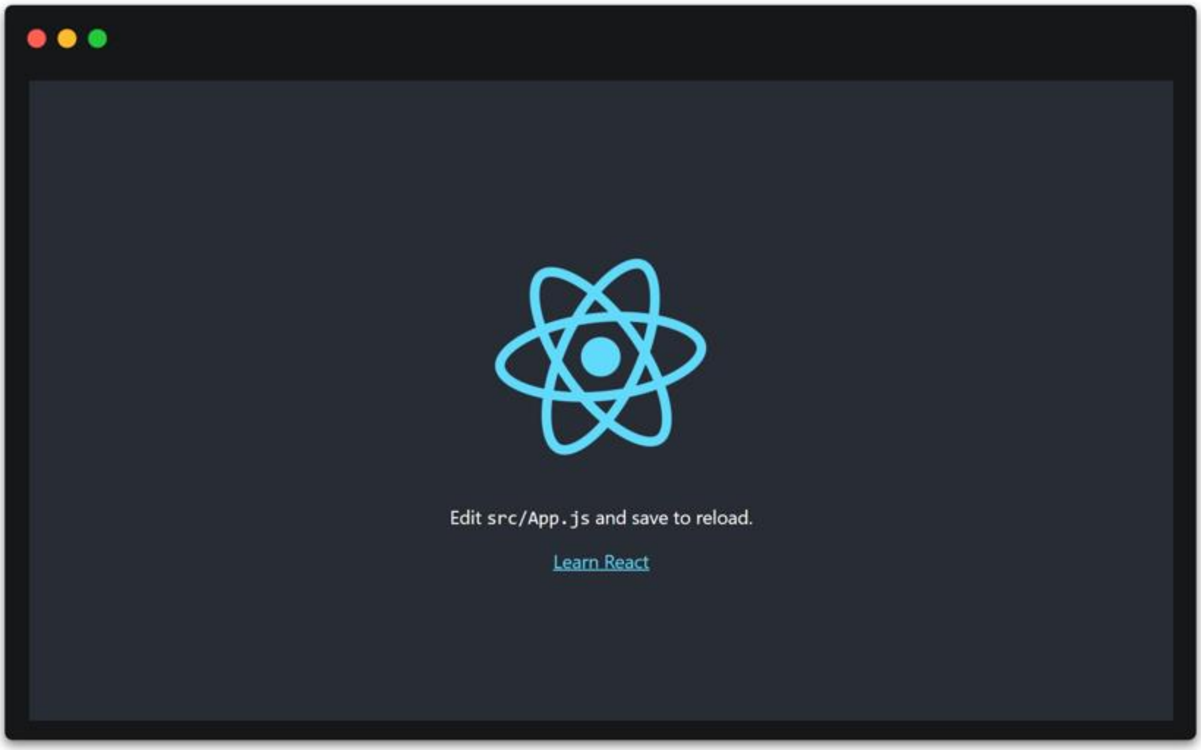
3

```
npx create-react-app fdx-amplify-demo
cd fdx-amplify-demo
npm start
```

Create a new React application with three lines of code:

```
npx create-react-app fdx-amplify-demo
cd fdx-amplify-demo
npm start
```

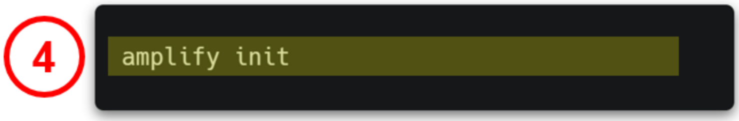
A new React app is created:



Edit src/App.js and save to reload.

Learn React

4. Initialize a New Amplify Project

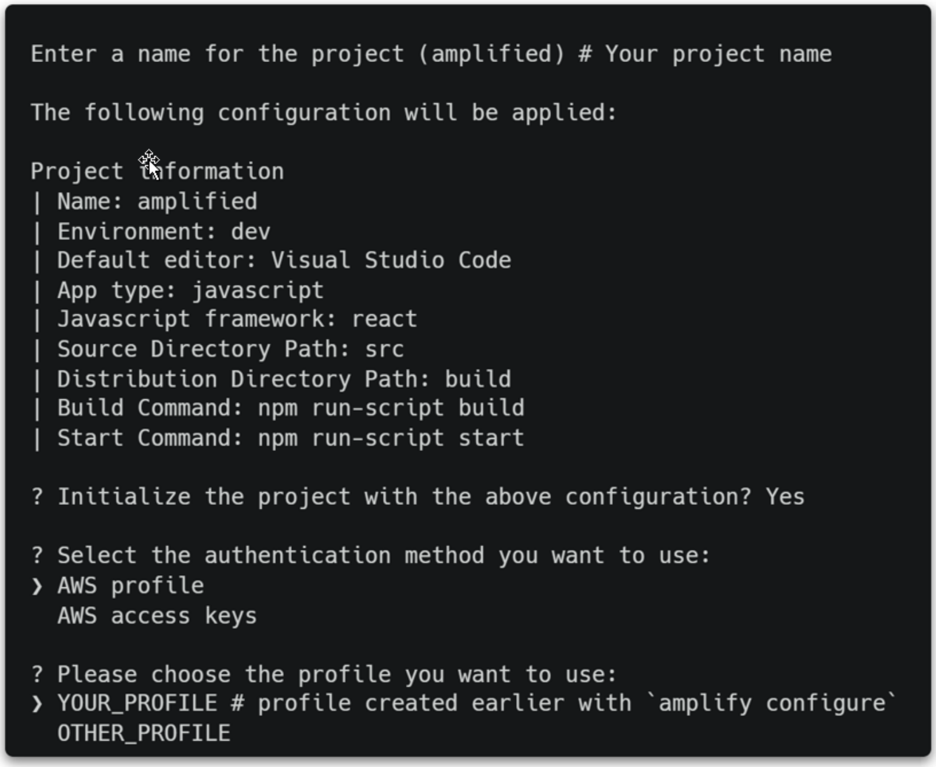


```
amplify init
```

With the frontend React app running, setup the Amplify backend with this one line of code:

```
amplify init
```

Amplify prompts you for the name and a few other configuration settings:



```
Enter a name for the project (amplified) # Your project name

The following configuration will be applied:

Project information
| Name: amplified
| Environment: dev
| Default editor: Visual Studio Code
| App type: javascript
| Javascript framework: react
| Source Directory Path: src
| Distribution Directory Path: build
| Build Command: npm run-script build
| Start Command: npm run-script start

? Initialize the project with the above configuration? Yes

? Select the authentication method you want to use:
❯ AWS profile
  AWS access keys

? Please choose the profile you want to use:
❯ YOUR_PROFILE # profile created earlier with `amplify configure`
  OTHER_PROFILE
```

Amplify Generated Files

When the Amplify project is initialized with Amplify CLI, Amplify generates code for a number of files automatically! These form the backend of your app:

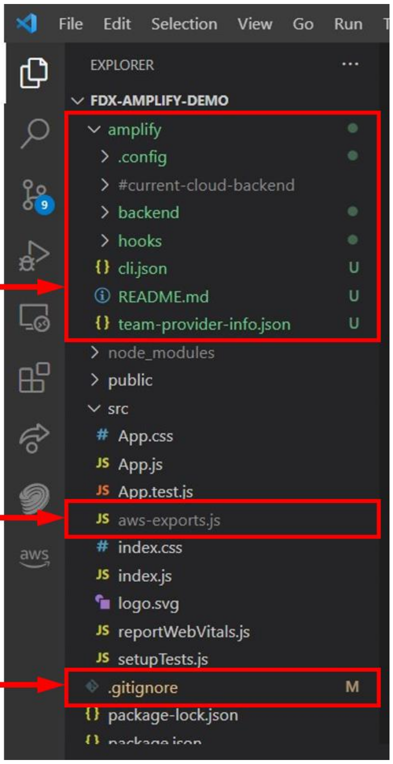
- Amplify backend definition
- Configuration for the backend services (**aws-export.js**)
- Updated git ignore file list (**.gitignore**)

Again, all of this code is generated automatically by Amplify!

creates top level directory called `amplify` with backend definition

`aws-export.js` holds configuration for the backend services

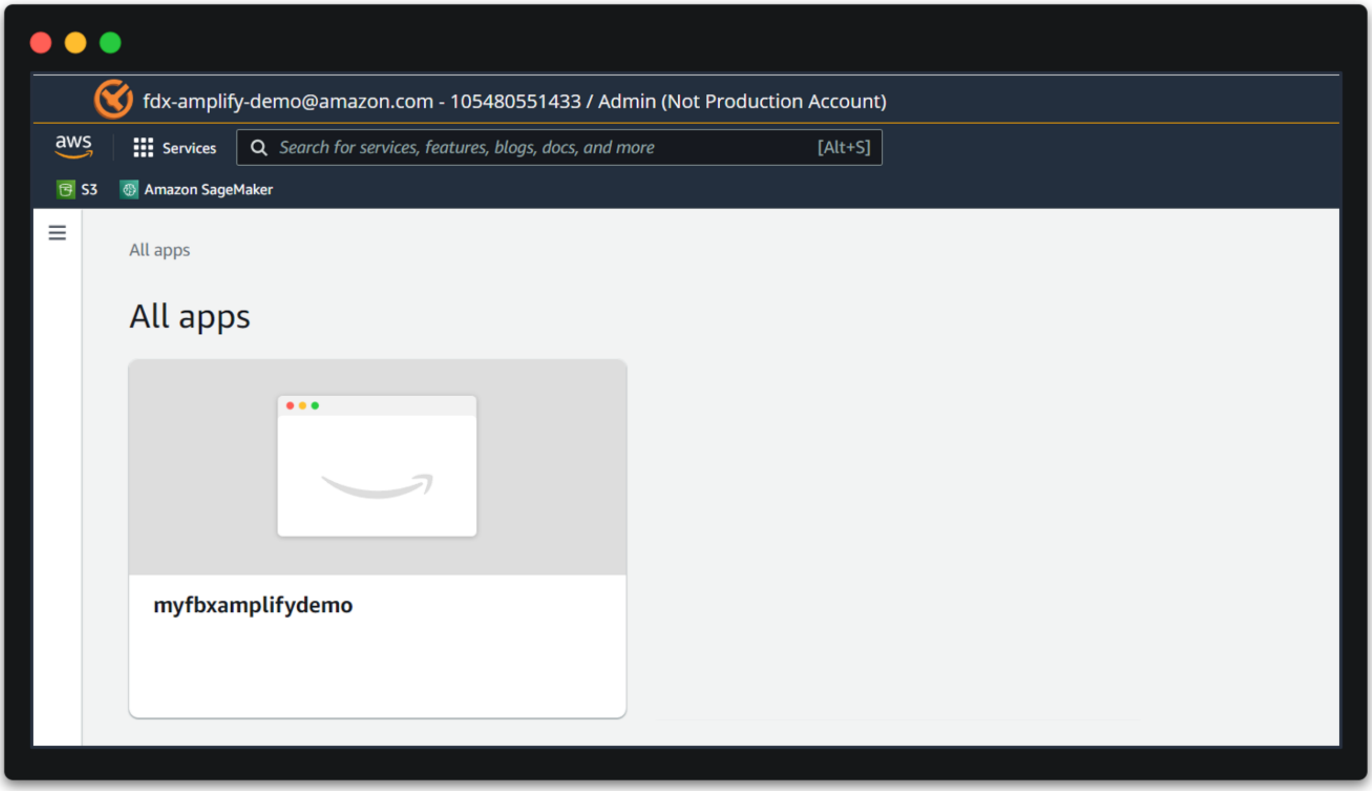
`.gitignore` adds some Amplify generated files to the ignore list



Amplify Generated Application in AWS Console

Amplify also creates a cloud project in the AWS Amplify Console that can be accessed by running the command “amplify console” in the command line:

```
amplify console
```



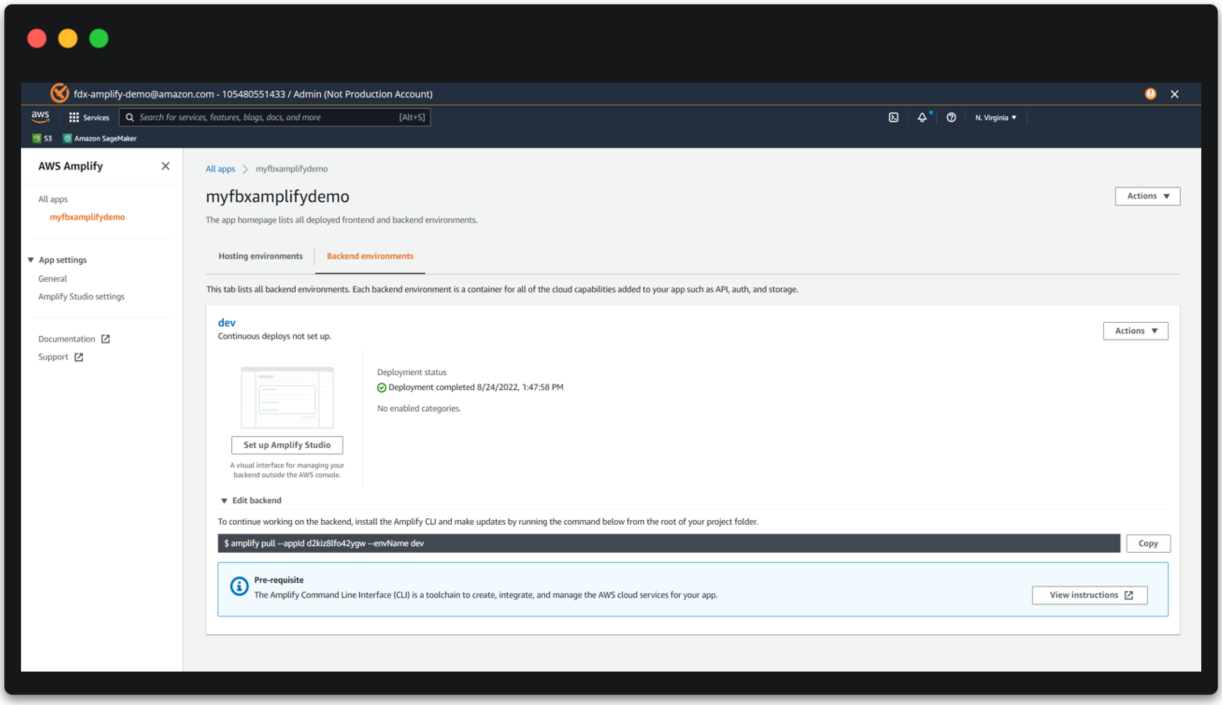
fdx-amplify-demo@amazon.com - 105480551433 / Admin (Not Production Account)

All apps

myfbxamplifydemo

Amplify Generated Backend in AWS Console

The Console provides a list of backend environments, deep links to provisioned resources per Amplify category, status of recent deployments, and instructions on how to promote, clone, pull, and delete backend resources.



The app homepage lists all deployed frontend and backend environments.

This tab lists all backend environments. Each backend environment is a container for all of the cloud capabilities added to your app such as API, auth, and storage.

Continuous deploys not set up.

A visual interface for managing your backend outside the AWS console.

Deployment status

Deployment completed 8/24/2022, 1:47:58 PM

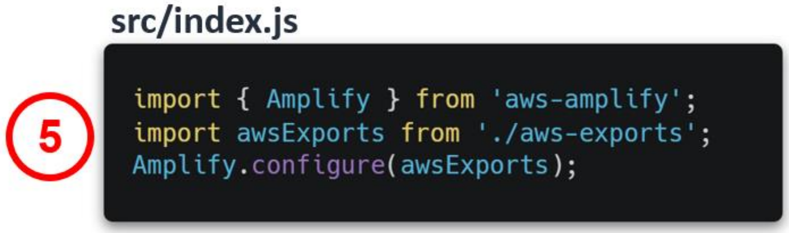
No enabled categories.

To continue working on the backend, install the Amplify CLI and make updates by running the command below from the root of your project folder.

```
$ amplify pull --appId d2kiz8lfo42ygw --envName dev
```

The Amplify Command Line Interface (CLI) is a toolchain to create, integrate, and manage the AWS cloud services for your app.

5. Connect Amplify to the Frontend



src/index.js

```
import { Amplify } from 'aws-amplify';
import awsExports from './aws-exports';
Amplify.configure(awsExports);
```

The last step to create the Amplify project is to connect Amplify to the frontend React app with these three lines of code added to index.js:

src/index.js

```
import { Amplify } from ‘aws-amplify’;
import awsExports from ‘./aws-exports’;
Amplify.configure(awsExports);
```

The code does the following by line number:

1. The first line imports Amplify.
2. The second line imports aws-export, which Amplify generated earlier.
3. The third line connects Amplify to the frontend.

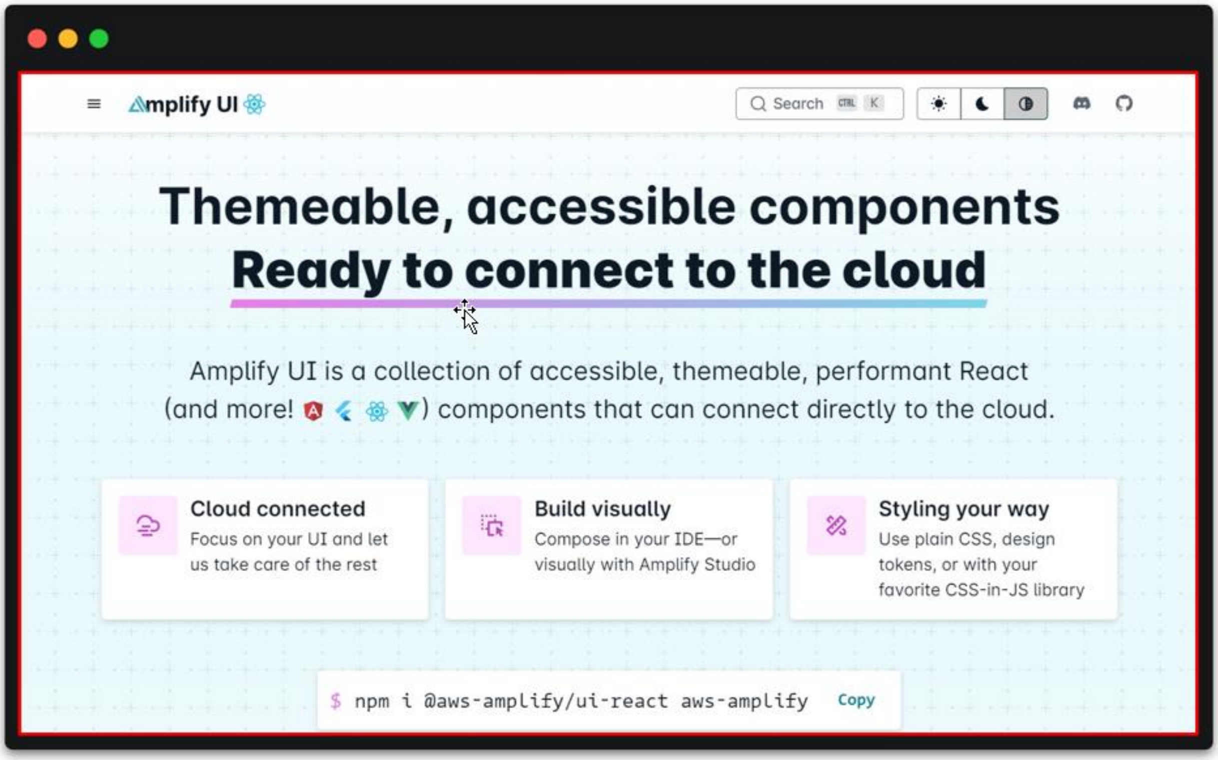
The client is now able to interact with the backend services through this frontend connection. In addition, future backend updates are automatically reflected through this connection.

App Development – Adding Services

Now that the React app is set up and amplify is initialized, we're ready to add new services such as Authentication, a GraphQL API, and Forge API.

Install Amplify UI Package

Amplify UI is a collection of React components that are cloud-connected:



Themeable, accessible components
Ready to connect to the cloud

Amplify UI is a collection of accessible, themeable, performant React (and more! A) components that can connect directly to the cloud.

Cloud connected
Focus on your UI and let us take care of the rest

Build visually
Compose in your IDE—or visually with Amplify Studio

Styling your way
Use plain CSS, design tokens, or with your favorite CSS-in-JS library

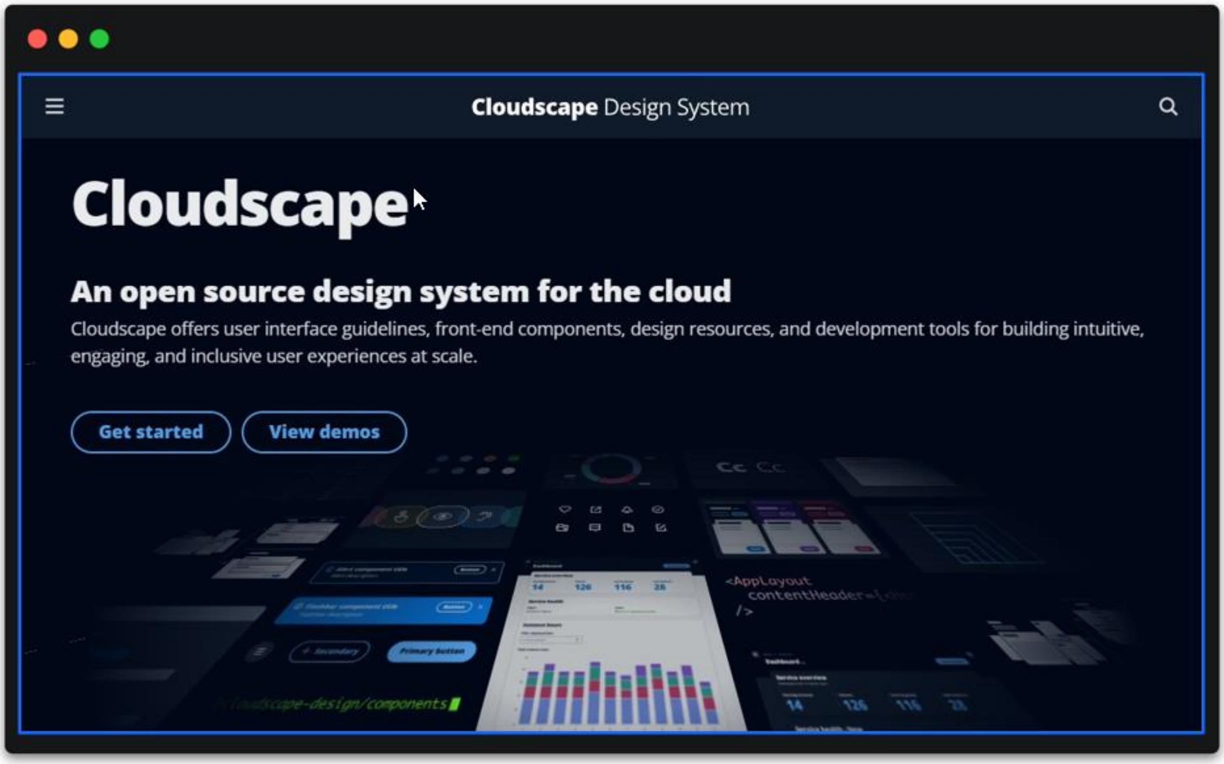
```
$ npm i @aws-amplify/ui-react aws-amplify
```

Install the Amplify UI package with:

```
npm install aws-amplify @aws-amplify/ui-react
```

Install Cloudscape Packages

https://cloudscape.design



Cloudscape

An open source design system for the cloud

Cloudscape offers user interface guidelines, front-end components, design resources, and development tools for building intuitive, engaging, and inclusive user experiences at scale.

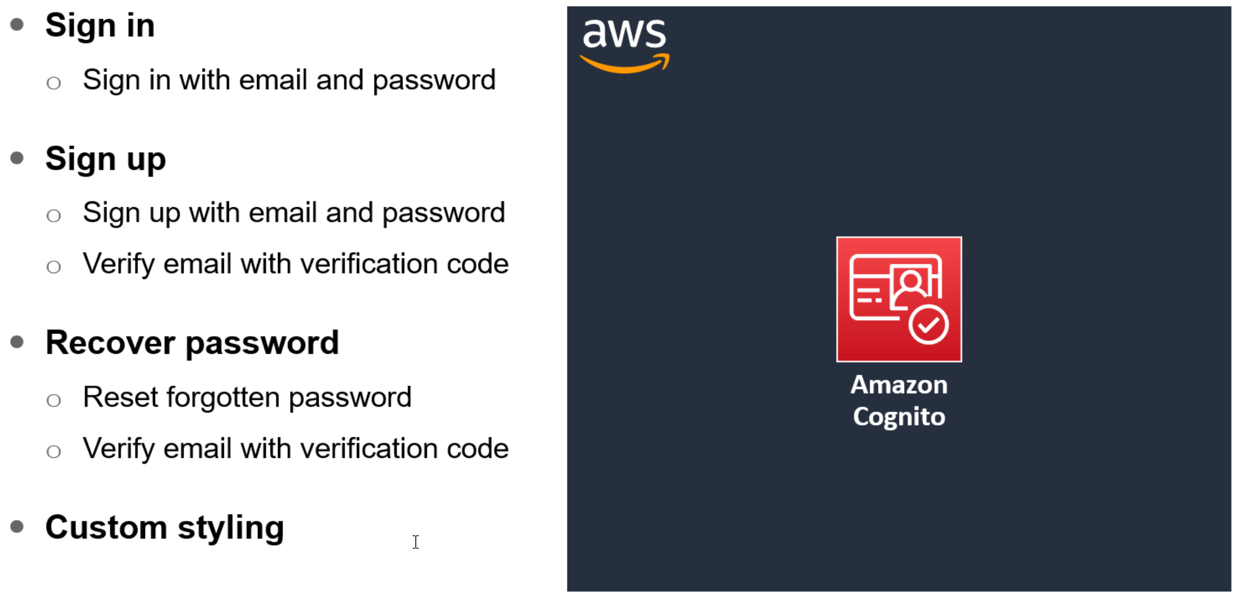
Install the following packages from Cloudscape:

- The **global-styles** package will give our app a nice look thanks to its CSS.
- The **components** package contains all the React components to build our user interface.
- The **design-tokens** package applies Cloudscape styles to the Amplify UI components.

```
npm install @cloudscape-design/global-styles
npm install @cloudscape-design/components
npm install @cloudscape-design/design-tokens
```

Add Authentication

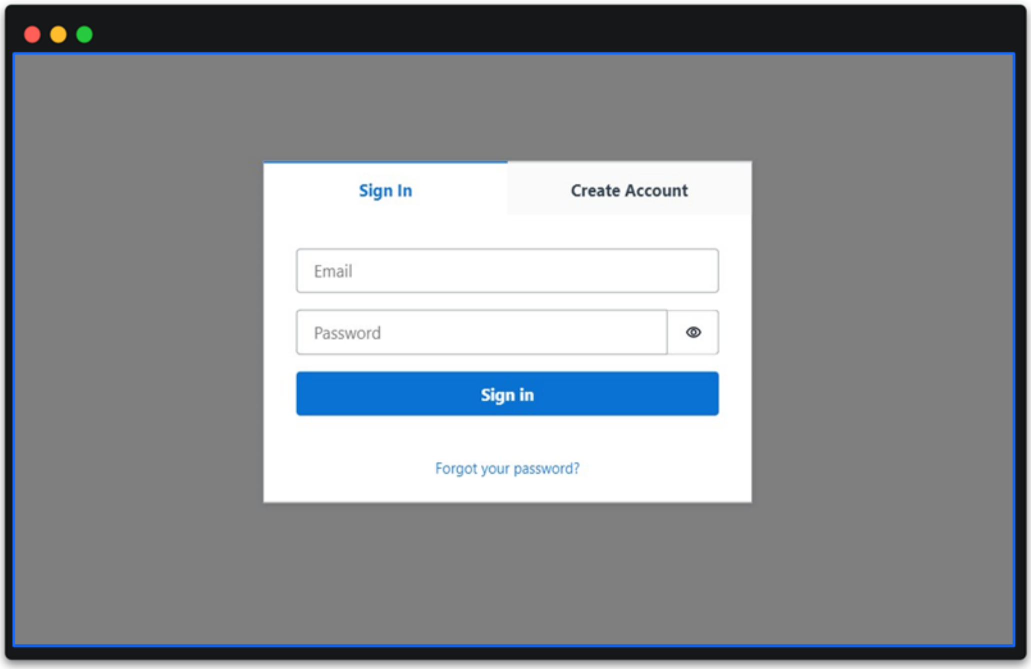
To add authentication to our app, the Amplify Framework uses Amazon Cognito as the main authentication provider. Amazon Cognito is a robust user directory service that handles user registration, authentication, account recovery & other operations. In this section, we’ll see how to add authentication to your application using Amazon Cognito and email/password login. The Amplify CLI together with the Amplify UI Authenticator component will create a fully automated authentication flow. This means that with a few lines of code, we will have a professional web application capable of sign in, sign up, password recovery, and even email verification when a user registers.



- **Sign in**
  - Sign in with email and password
- **Sign up**
  - Sign up with email and password
  - Verify email with verification code
- **Recover password**
  - Reset forgotten password
  - Verify email with verification code
- **Custom styling**

Add Authentication

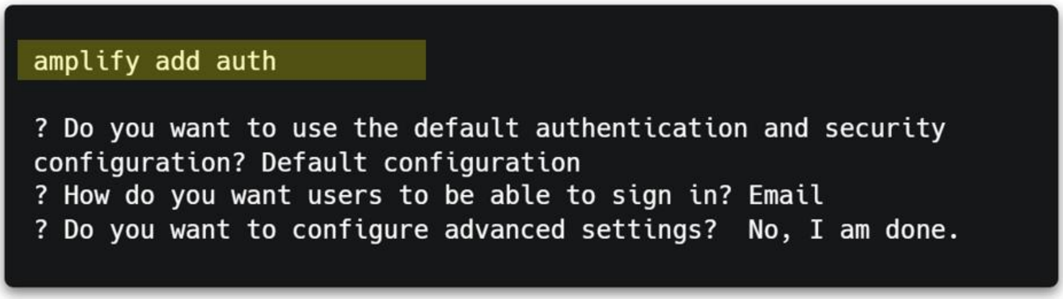
As a result of adding authentication, the application loads with a login dialog:



Create Authentication Service

To add authentication, we simply run “**amplify add auth**” in the command line. We then follow the prompts and choose **Email** as our preferred sign in method:

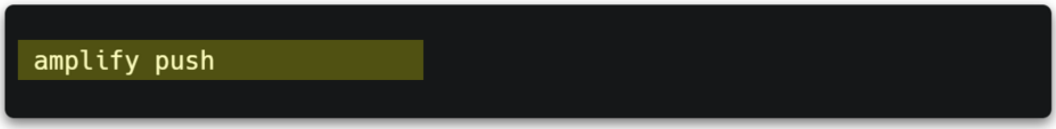
```
amplify add auth
```



```
amplify add auth

? Do you want to use the default authentication and security
configuration? Default configuration
? How do you want users to be able to sign in? Email
? Do you want to configure advanced settings?  No, I am done.
```

We then run the “**amplify push**” command to deploy the service to our backend:

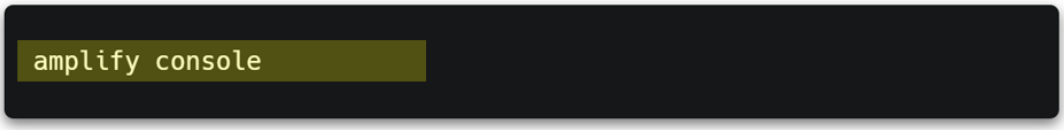
```
amplify push
```



```
amplify push
```

Now, the authentication service has been deployed, it’s ready to use. To view the deployed services in your project at any time, go to your Amplify Console by running the “**amplify console**” command.

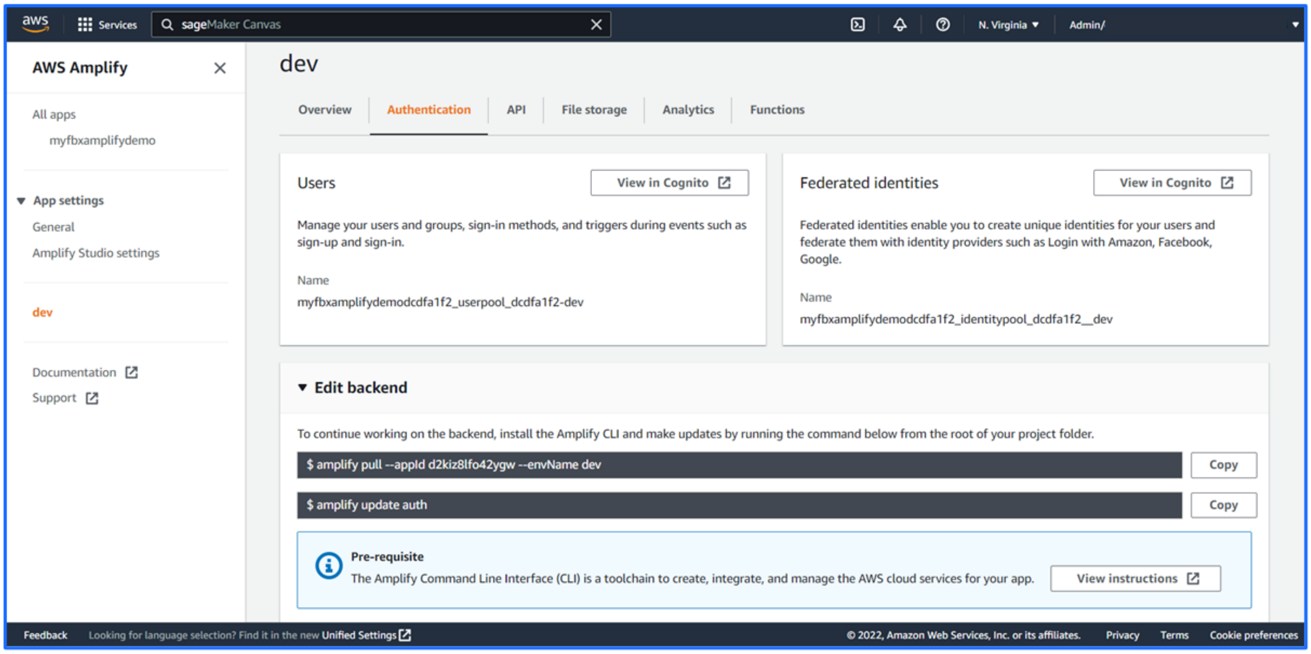
```
amplify console
```



```
amplify console
```

Create Authentication Service

From the AWS Console, we see that our authentication services have been deployed:



dev

Users

Manage your users and groups, sign-in methods, and triggers during events such as sign-up and sign-in.

Name
myfbxamplifydemodcdfa1f2_userpool_dcdfa1f2-dev

Federated identities

Federated identities enable you to create unique identities for your users and federate them with identity providers such as Login with Amazon, Facebook, Google.

Name
myfbxamplifydemodcdfa1f2_identitypool_dcdfa1f2__dev

Edit backend

To continue working on the backend, install the Amplify CLI and make updates by running the command below from the root of your project folder.

```
$ amplify pull --appId d2kiz8lfo42ygw --envName dev
```

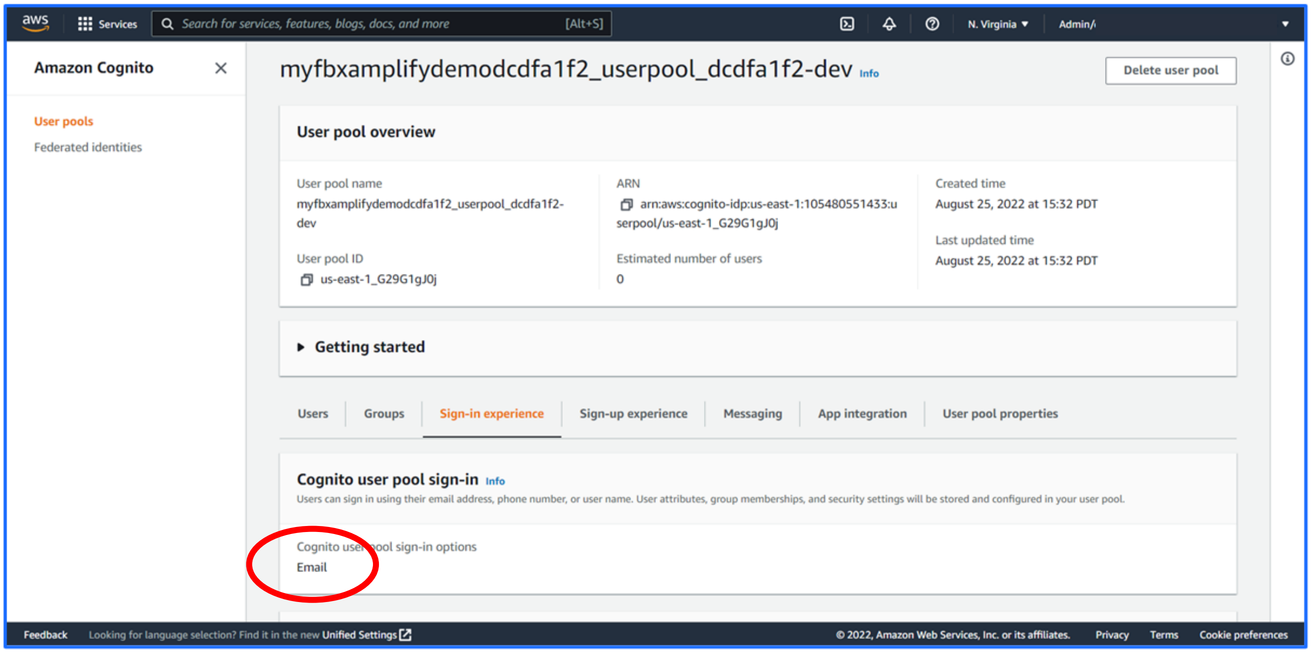
```
$ amplify update auth
```

Pre-requisite
The Amplify Command Line Interface (CLI) is a toolchain to create, integrate, and manage the AWS cloud services for your app.

Looking for language selection? Find it in the new Unified Settings

© 2022, Amazon Web Services, Inc. or its affiliates.

If we then navigate to the Amazon Cognito service page, we see that there's a newly created user pool to manage all our users. And if we take a look at the Sign-in Experience tab, Email is set as our sign-in method, which we specified earlier in the command line:



myfbxamplifydemodcdfa1f2_userpool_dcdfa1f2-dev

User pool overview

User pool name
myfbxamplifydemodcdfa1f2_userpool_dcdfa1f2-dev

User pool ID
us-east-1_G29G1gJ0j

ARN
arn:aws:cognito-idp:us-east-1:105480551433:userpool/us-east-1_G29G1gJ0j

Estimated number of users
0

Created time
August 25, 2022 at 15:32 PDT

Last updated time
August 25, 2022 at 15:32 PDT

Cognito user pool sign-in

Users can sign in using their email address, phone number, or user name. User attributes, group memberships, and security settings will be stored and configured in your user pool.

Cognito user pool sign-in options
Email

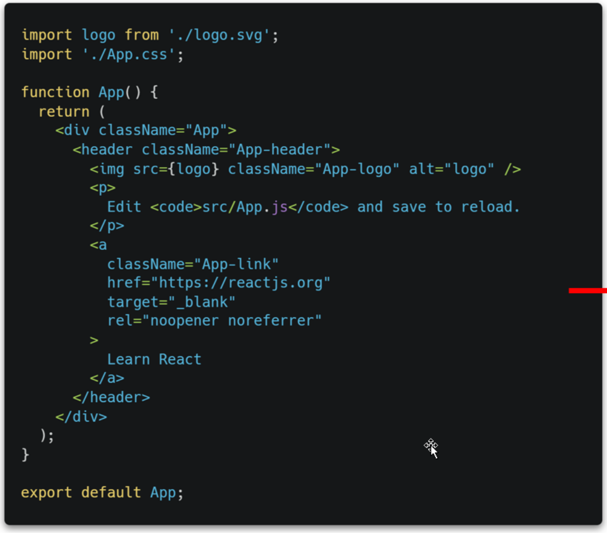
Looking for language selection? Find it in the new Unified Settings

© 2022, Amazon Web Services, Inc. or its affiliates.

Create Login UI

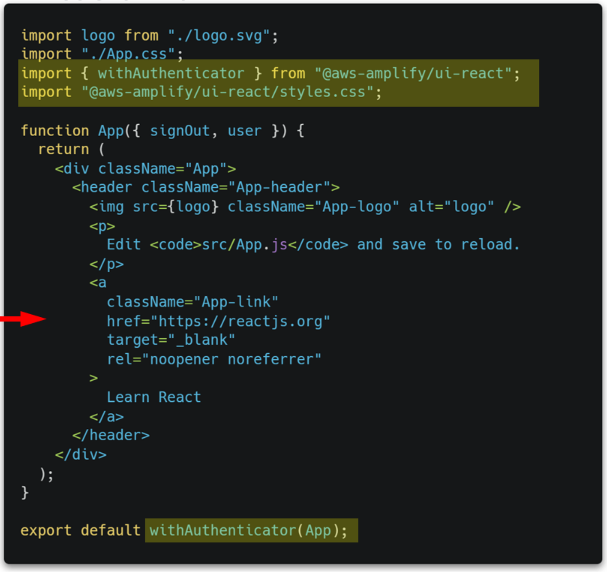
We add the fully automated authentication flow to with three lines of code using the Authenticator component:

src/app.js (before)



```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

src/app.js (after)



```
import logo from "./logo.svg";
import "./App.css";
import { withAuthenticator } from "@aws-amplify/ui-react";
import "@aws-amplify/ui-react/styles.css";

function App({ signOut, user }) {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default withAuthenticator(App);
```

Import the **withAuthenticator** component from **aws-amplify/ui-react**:

```
import { withAuthenticator } from “@aws-amplify/ui-react”;
```

Import its styles in css:

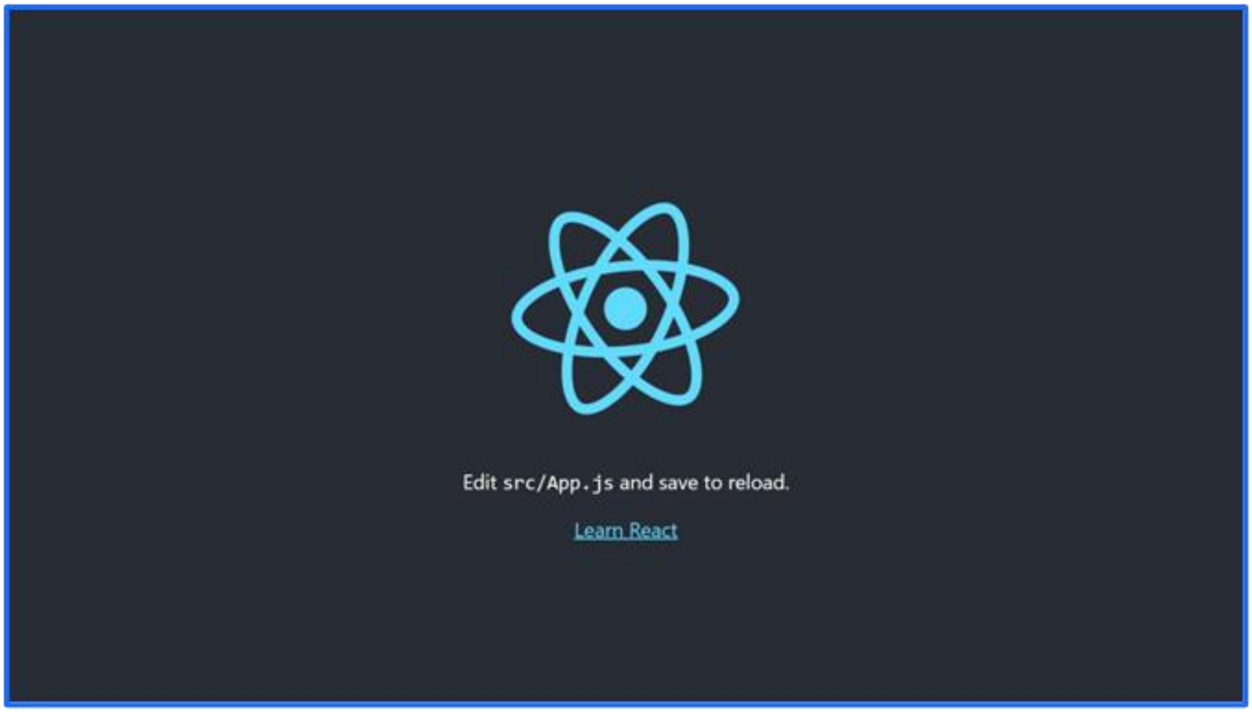
```
import “@aws-amplify/ui-react/style.css”;
```

Wrap the **App** function with the Authenticator component:

```
export default withAuthenticator(App);
```

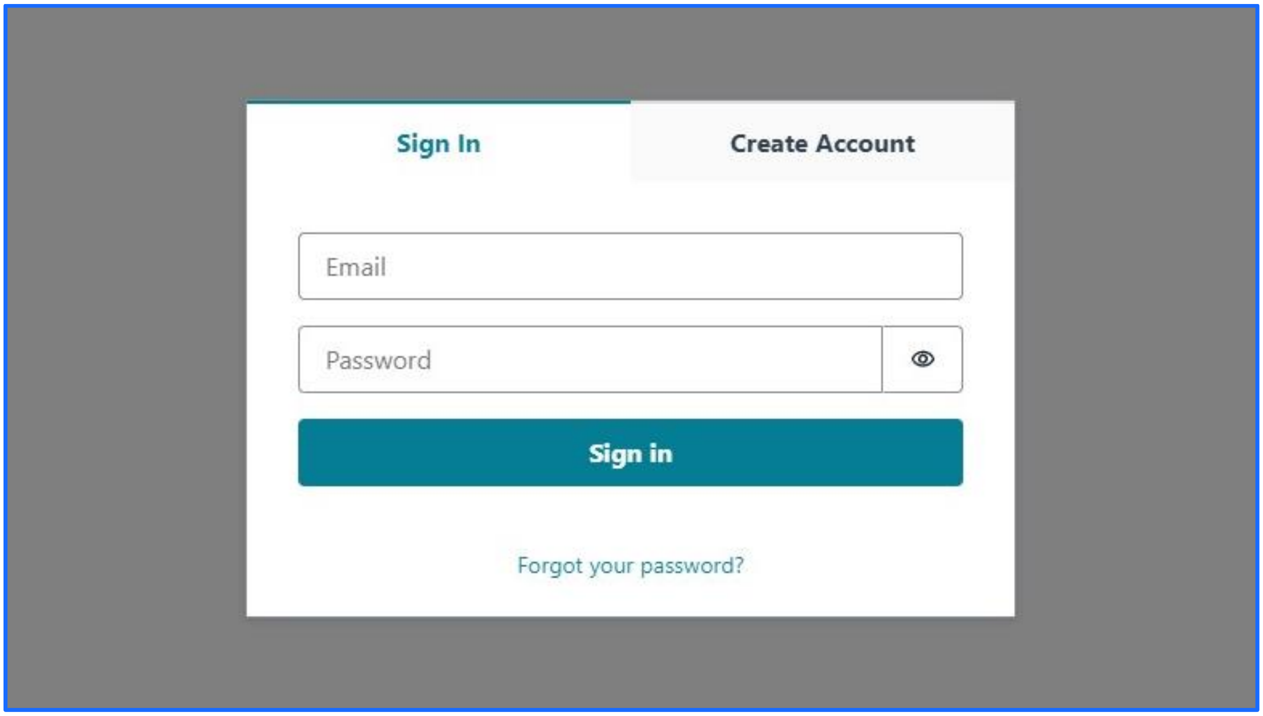
The Authenticator works seamlessly with Amplify CLI to ***automatically*** work with the backend. No extra configuration is needed!

Before:



Edit src/App.js and save to reload.

After configuring the Authenticator with the three lines of code, the login dialog is implemented:



Login UI Custom Theme

https://ui.docs.amplify.aws/react/theming

With a fully automated authentication flow in place, we can customize its look to match our brand and app identity. Since we are using Cloudscape as our primary design system, we apply the same primary color from Cloudscape to the Amplify UI components.

First, import **ThemeProvider** from Amplify UI, and Cloudscape design-tokens in the **src/index.js** file:

```
import { ThemeProvider } from “@aws-amplify/ui-react”;
import * as awsui from “@cloudscape-design/design-tokens”;
```

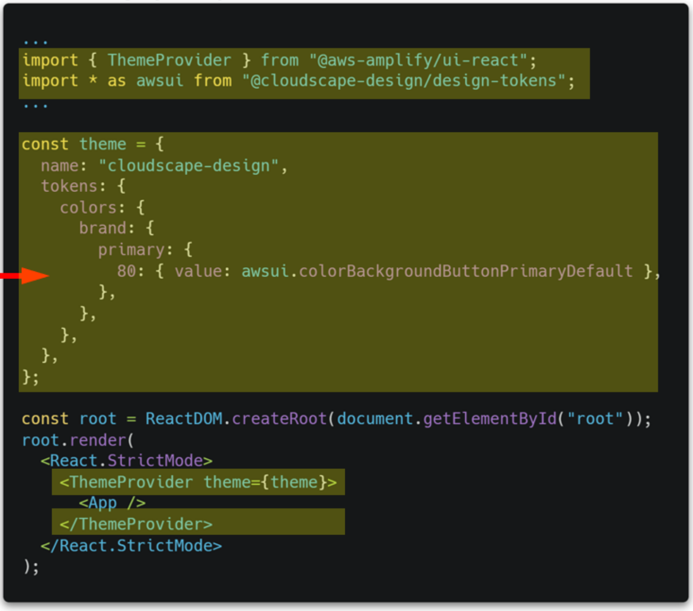
Next, create a custom theme object such that the Amplify UI primary color is the same as the Cloudscape primary color:

```
const theme = {
   name: “cloudscape-design”,
    tokens: {
      colors: {
       brand: {
         primary: {
          80: { value: awsui.colorBackgroundButtonPrimaryDefault },
         },
       },
      },
    },
   },
};
```

To apply the custom theme to all Amplify UI components, we pass the new theme object to the Amplify UI <ThemeProvider> component, which wraps the main <App> component:

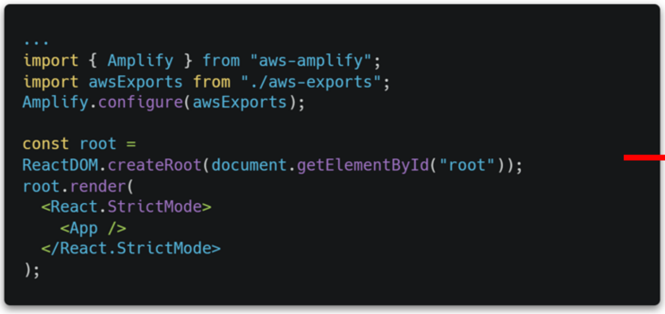
```
<ThemeProvider theme={ theme }>
    <App />
</ThemeProvider>
```

src/index.js (after)



```
...
import { ThemeProvider } from "@aws-amplify/ui-react";
import * as awsui from "@cloudscape-design/design-tokens";
...

const theme = {
  name: "cloudscape-design",
  tokens: {
    colors: {
      brand: {
        primary: {
          80: { value: awsui.colorBackgroundButtonPrimaryDefault },
        },
      },
    },
  },
};

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <ThemeProvider theme={theme}>
      <App />
    </ThemeProvider>
  </React.StrictMode>
);
```

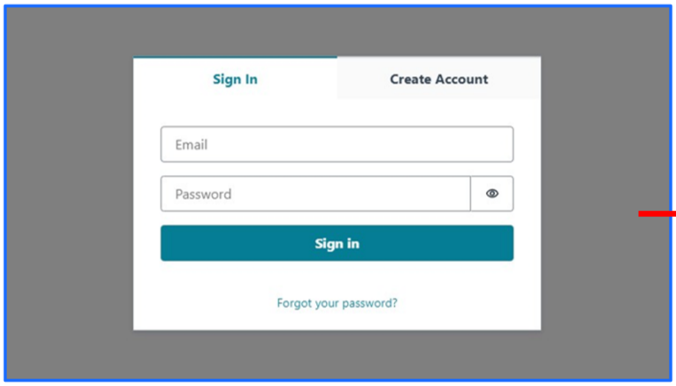
src/index.js (before)



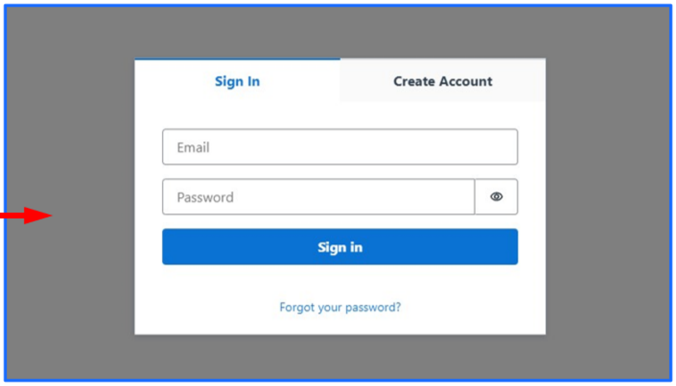
```
...
import { Amplify } from "aws-amplify";
import awsExports from "./aws-exports";
Amplify.configure(awsExports);

const root =
ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

After a screen refresh of the app, all primary colors including the tab font, the button background, and the “Forgot your password” link color match our React app primary color:

before



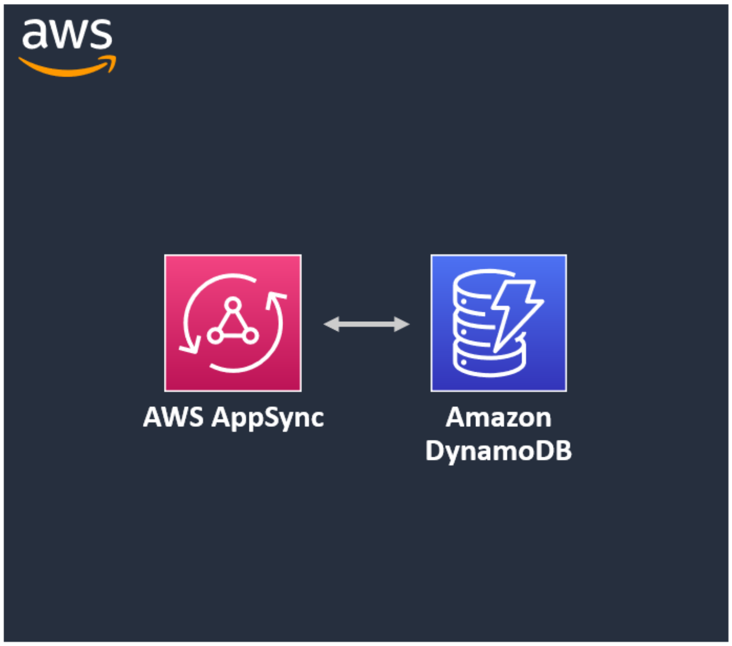
after



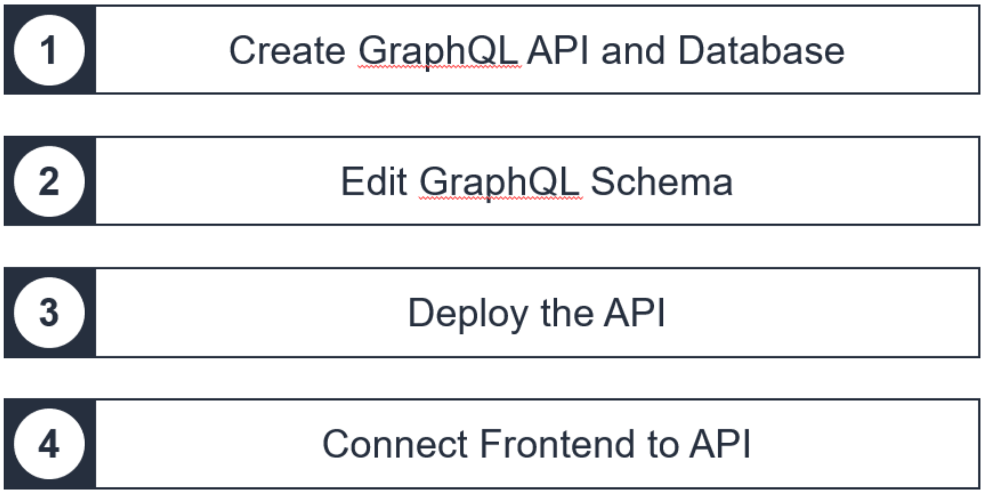
Add GraphQL API Using Amplify

To add functionality to the application, for example when the user clicks on a Data Exchange or IoT device, we add an API to fetch models, create new ones, and subscribe to real-time updates. Amplify uses AWS AppSync to create a GraphQL API, and Amazon DynamoDB to create a NoSQL database.

- **Queries**
  - Get list of models
  - Get list of devices
- **Mutations**
  - Create a new model
  - Create a new device
- **Subscriptions**
  - Get real-time notifications when a device is updated



Steps to Add an API



1 Create GraphQL API and Database

2 Edit GraphQL Schema

3 Deploy the API

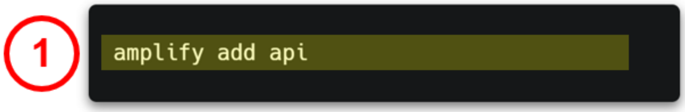
4 Connect Frontend to API

Five steps are needed to create an Amplify project:

1. Create the GraphQL API and Database
2. Edit GraphQL Schema
3. Deploy the API
4. Connect Frontend to API

1. Create GraphQL API and Database

With just one line of code, create the api:

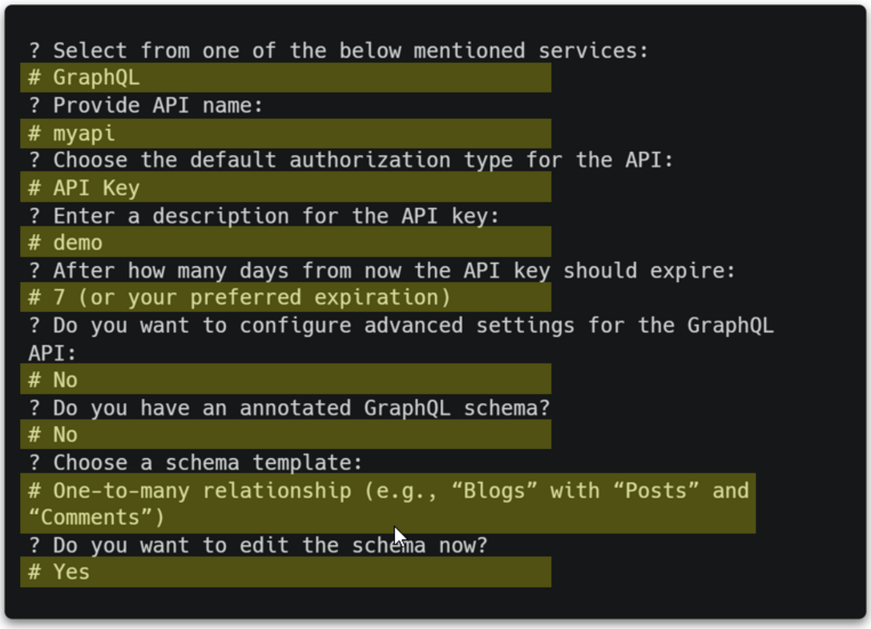


1

```
amplify add api
```

```
amplify add api
```

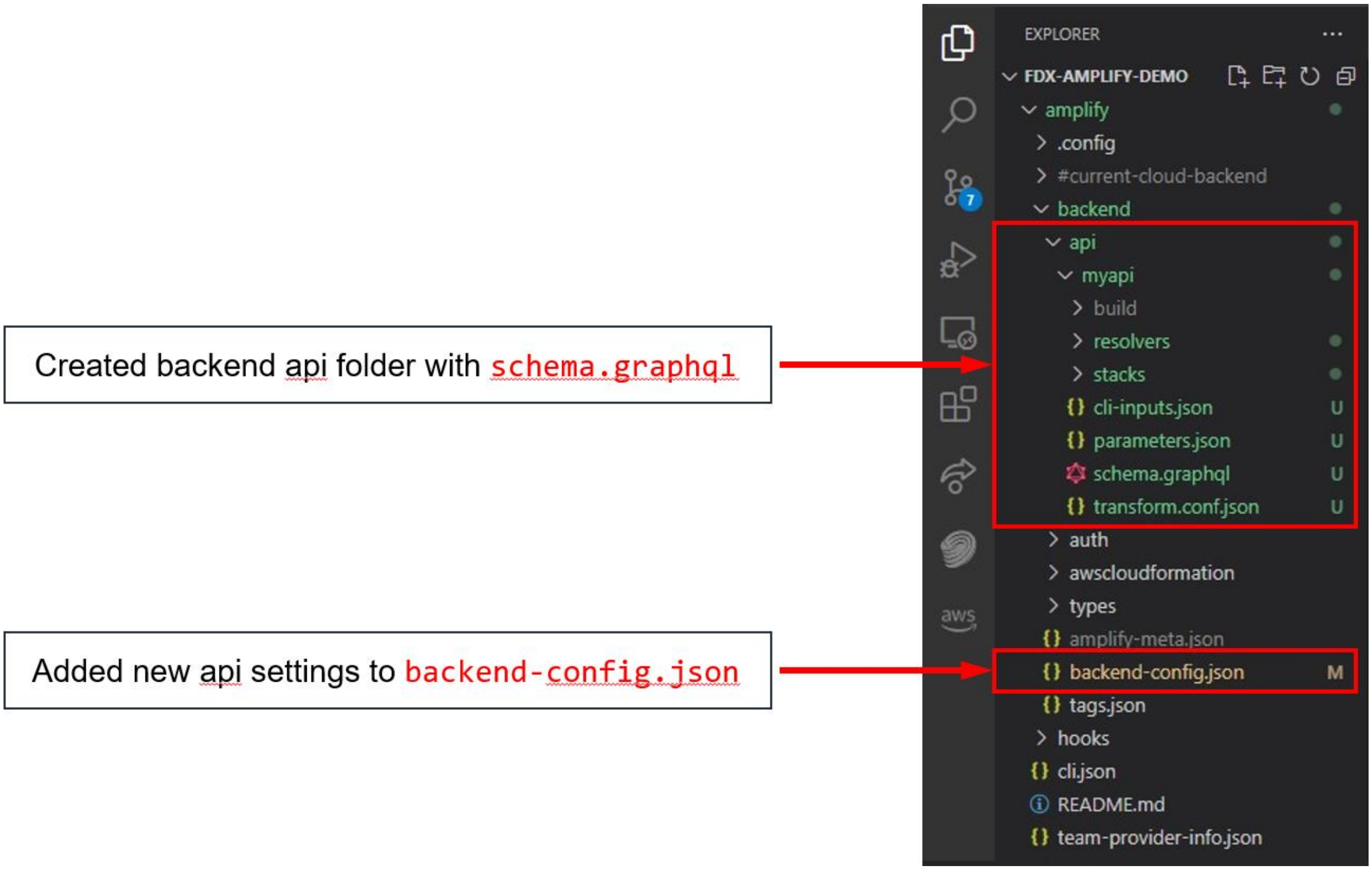
At the Amplify CLI prompt, select GraphQL, give it a name, and enter other values in the wizard. At the end, choose “Yes” to edit the Edit schema:



```
? Select from one of the below mentioned services:
# GraphQL
? Provide API name:
# myapi
? Choose the default authorization type for the API:
# API Key
? Enter a description for the API key:
# demo
? After how many days from now the API key should expire:
# 7 (or your preferred expiration)
? Do you want to configure advanced settings for the GraphQL
API:
# No
? Do you have an annotated GraphQL schema?
# No
? Choose a schema template:
# One-to-many relationship (e.g., “Blogs” with “Posts” and
“Comments”)
? Do you want to edit the schema now?
# Yes
```

Amplify Generated Files

When the GraphQL API is added to the application, Amplify does the heavy lifting by generating all the backend code! Here, it created a dedicated "api" folder and updated the **backend-configuration.json** file:

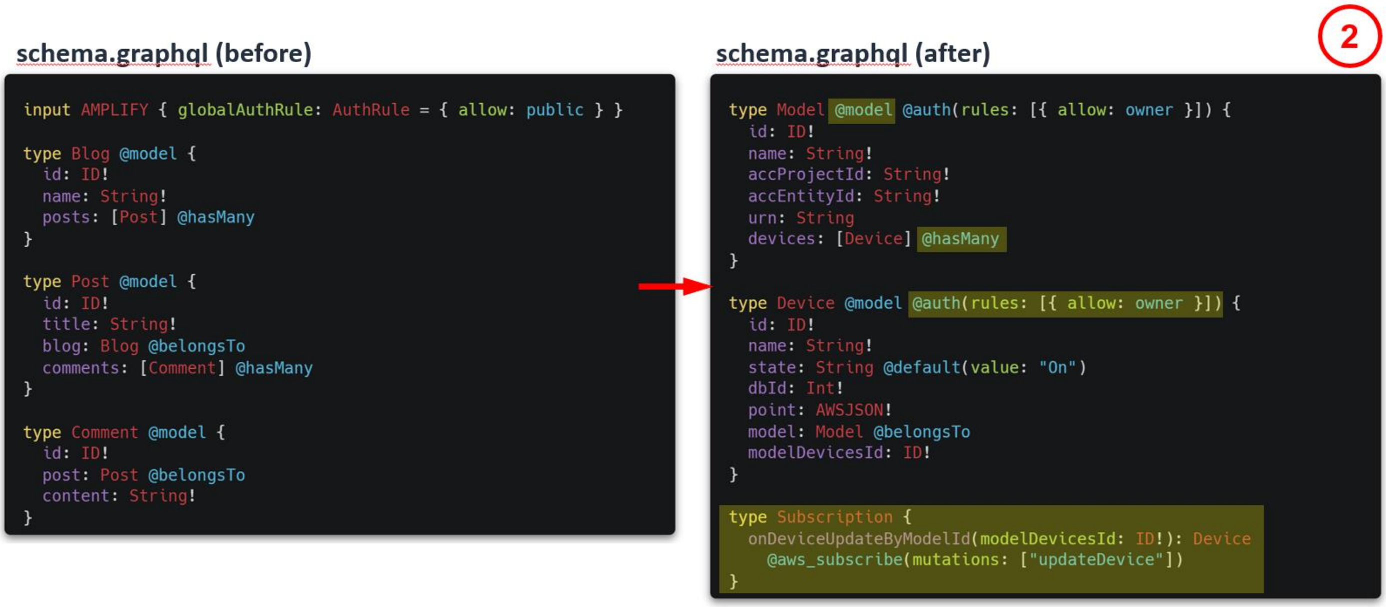


Created backend api folder with `schema.graphql`

Added new api settings to `backend-config.json`

2. Edit GraphQL Schema

The template schema is modified to have two types, Models and Devices, such that one model can have many devices.



2

schema.graphql (before)

```
input AMPLIFY { globalAuthRule: AuthRule = { allow: public } }

type Blog @model {
  id: ID!
  name: String!
  posts: [Post] @hasMany
}

type Post @model {
  id: ID!
  title: String!
  blog: Blog @belongsTo
  comments: [Comment] @hasMany
}

type Comment @model {
  id: ID!
  post: Post @belongsTo
  content: String!
}
```

schema.graphql (after)

```
type Model @model @auth(rules: [{ allow: owner }]) {
  id: ID!
  name: String!
  accProjectId: String!
  accEntityId: String!
  urn: String
  devices: [Device] @hasMany
}

type Device @model @auth(rules: [{ allow: owner }]) {
  id: ID!
  name: String!
  state: String @default(value: "On")
  dbId: Int!
  point: AWSJSON!
  model: Model @belongsTo
  modelDevicesId: ID!
}

type Subscription {
  onDeviceUpdateByModelId(modelDevicesId: ID!): Device
    @aws_subscribe(mutations: ["updateDevice"])
}
```

The “**@model**” directive informs Amplify to do the following:

- Scaffold out the necessary database in DynamoDB
- Create the schema for CRUD and list operations
- Create GraphQL resolvers needed to make everything work together

The “**@hasMany**” sets the one-to-many relationship between Models and Devices.

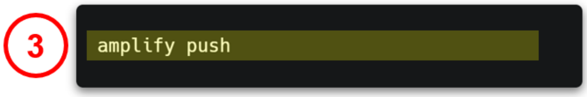
The “**@auth**” enables custom authorization rules.

Last, a custom subscription is added to receive real-time updates every time a device is updated:

```
type Subscription {
     onDeviceUpdateByModelId(modelDevicesId: ID!): Device
             @aws_subscribe(mutations: [“updateDevice”])
}
```

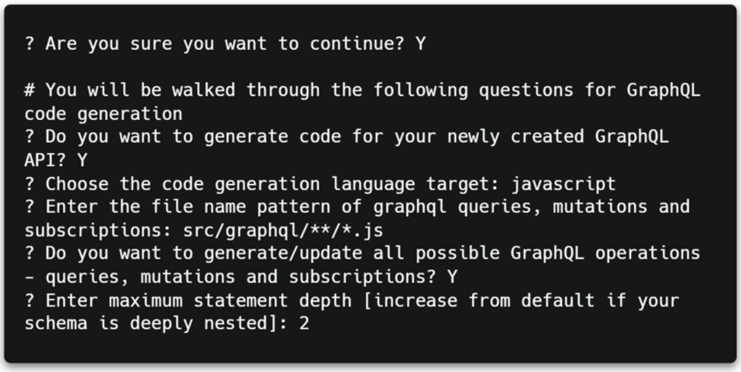
3. Deploy the API

Execute “amplify push” to deploy this API to the backend:



3

```
amplify push
```



```
? Are you sure you want to continue? Y

# You will be walked through the following questions for GraphQL
code generation
? Do you want to generate code for your newly created GraphQL
API? Y
? Choose the code generation language target: javascript
? Enter the file name pattern of graphql queries, mutations and
subscriptions: src/graphql/**/*.js
? Do you want to generate/update all possible GraphQL operations
- queries, mutations and subscriptions? Y
? Enter maximum statement depth [increase from default if your
schema is deeply nested]: 2
```

```
amplify push
```

Amplify CLI prompts to configure GraphQL.

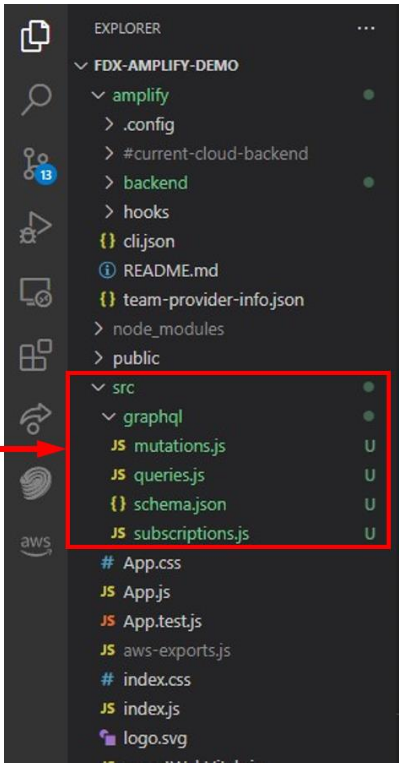
The API is now live and we can start interacting with it!

Amplify Generated Files

Based on the **schema.graphql** configuration above, Amplify has generated all the code for the Frontend of the app in the following files:

```
src/graphql/mutations.js
src/graphql/queries.js
src/graphql/schema.json
src/graphql/subscriptions.js
```

Generated code for mutations, queries, and subscriptions



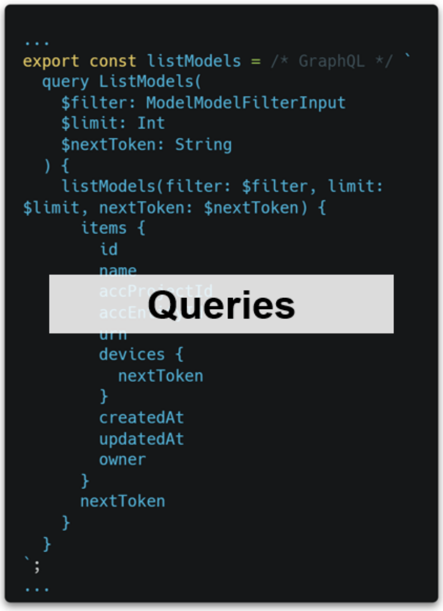
```
EXPLORER                          ...
∨ FDX-AMPLIFY-DEMO
  ∨ amplify
    > .config
    > #current-cloud-backend
    > backend
    > hooks
    {} cli.json
    ⓘ README.md
    {} team-provider-info.json
  > node_modules
  > public
  ∨ src
    ∨ graphql
      JS mutations.js             U
      JS queries.js               U
      {} schema.json              U
      JS subscriptions.js         U
    # App.css
    JS App.js
    JS App.test.js
    JS aws-exports.js
    # index.css
    JS index.js
    logo.svg
```

A Look Inside the Generated Code

The code generated by Amplify includes the following operations for Models and Devices:

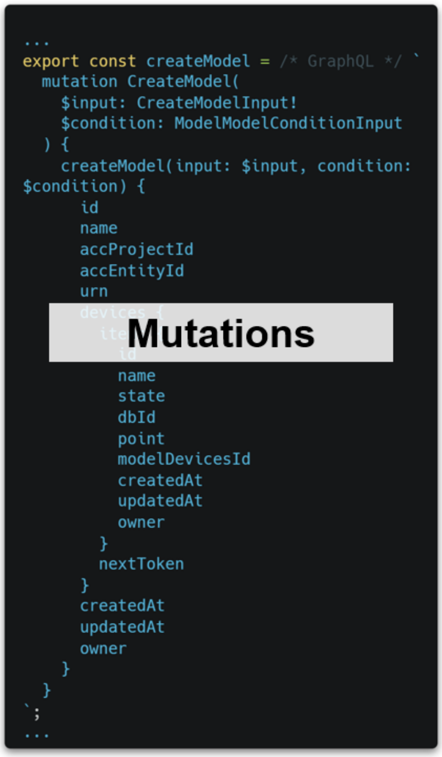
- Create
- Read
- Update
- Delete
- List

Code for subscribing to real-time updates is also generated!



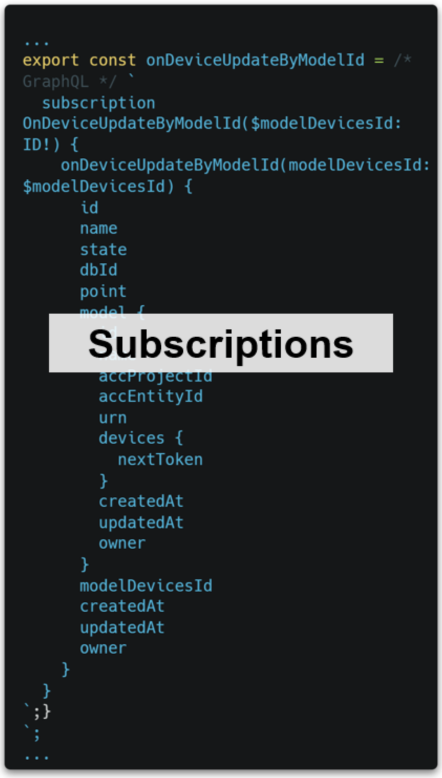
Queries

```
...
export const listModels = /* GraphQL */ `
  query ListModels(
    $filter: ModelModelFilterInput
    $limit: Int
    $nextToken: String
  ) {
    listModels(filter: $filter, limit:
$limit, nextToken: $nextToken) {
      items {
        id
        name
        accProjectId
        accEntityId
        urn
        devices {
          nextToken
        }
        createdAt
        updatedAt
        owner
      }
      nextToken
    }
  }
`;
...
```



Mutations

```
...
export const createModel = /* GraphQL */ `
  mutation CreateModel(
    $input: CreateModelInput!
    $condition: ModelModelConditionInput
  ) {
    createModel(input: $input, condition:
$condition) {
      id
      name
      accProjectId
      accEntityId
      urn
      devices {
        items {
          id
          name
          state
          dbId
          point
          modelDevicesId
          createdAt
          updatedAt
          owner
        }
        nextToken
      }
      createdAt
      updatedAt
      owner
    }
  }
`;
...
```



Subscriptions

```
...
export const onDeviceUpdateByModelId = /*
GraphQL */ `
  subscription
OnDeviceUpdateByModelId($modelDevicesId:
ID!) {
    onDeviceUpdateByModelId(modelDevicesId:
$modelDevicesId) {
      id
      name
      state
      dbId
      point
      model {
        id
        name
        accProjectId
        accEntityId
        urn
        devices {
          nextToken
        }
        createdAt
        updatedAt
        owner
      }
      modelDevicesId
      createdAt
      updatedAt
      owner
    }
  }
`;}
`;
...
```

4. Connect Frontend to API - Queries

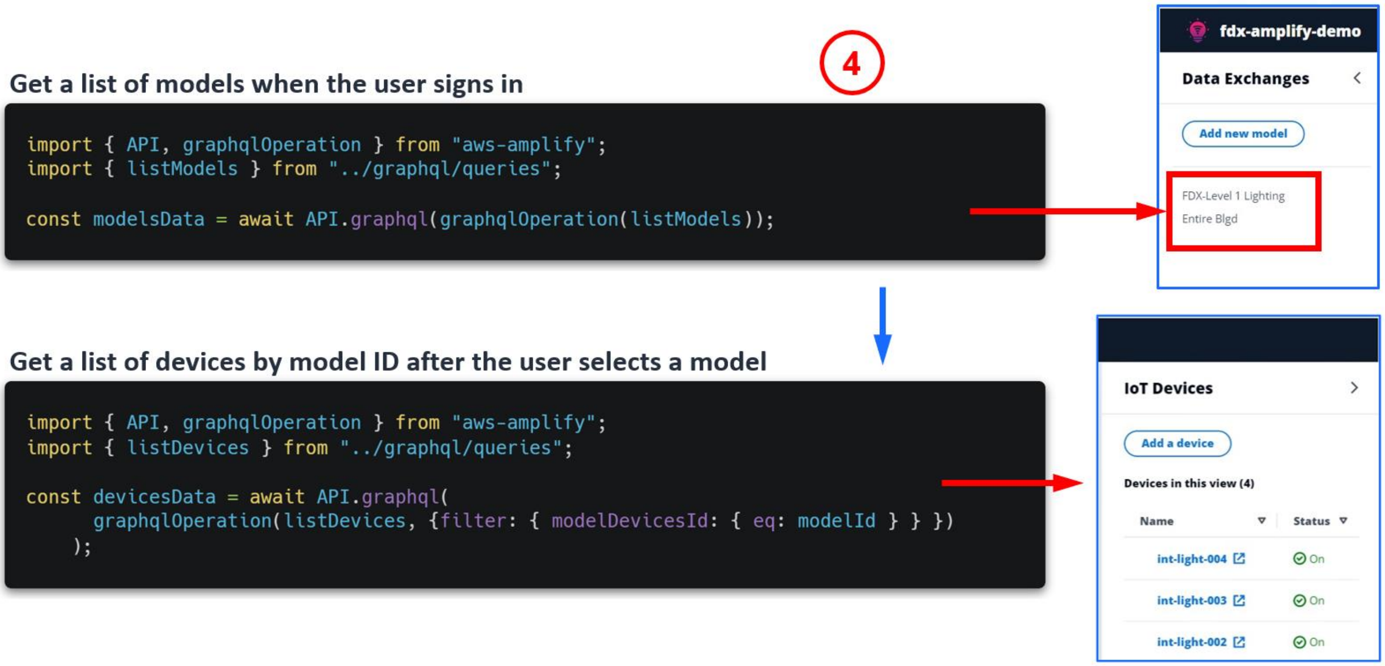
Next, a few lines of code are needed to implement the generated query code from the previous step into the Frontend.

A list of models is fetched when a user signs in. This is used to populate the UI with a list of models in the “Data Exchanges” panel. The following is the code-behind:

```
import { API, graphqlOperation } from “aws-amplify”;
import { listModels } from “../graphql/queries”;

const modelsData = await API.graphql(graphqlOperation(listModels));
```

The “Add a device” button gets a list of devices by model ID after the user selects a model. This is used to populate the “IoT devices” panel with a list of devices based on the selected model. The following is the code-behind:

```
import { API, graphqlOperation } from “aws-amplify”;
import { listDevices } from “../graphql/queries”;

const devicesData = await API.graphql(
   graphqlOperation(listDevices, { filter: { modelDevicesId: {
      eq: modelId } } })
);
```



Get a list of models when the user signs in

```
import { API, graphqlOperation } from "aws-amplify";
import { listModels } from "../graphql/queries";

const modelsData = await API.graphql(graphqlOperation(listModels));
```

Get a list of devices by model ID after the user selects a model

```
import { API, graphqlOperation } from "aws-amplify";
import { listDevices } from "../graphql/queries";

const devicesData = await API.graphql(
      graphqlOperation(listDevices, {filter: { modelDevicesId: { eq: modelId } } })
    );
```

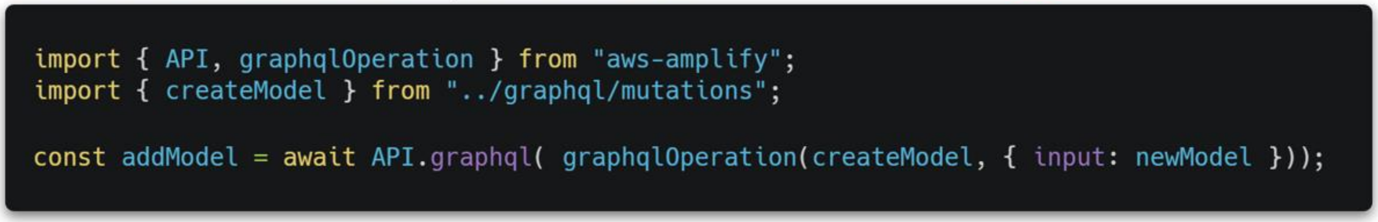
4. Connect Frontend to API - Mutations

When the user clicks on the “Add new model” button, a new model is created and recorded into the database. The code-behind is as follows:

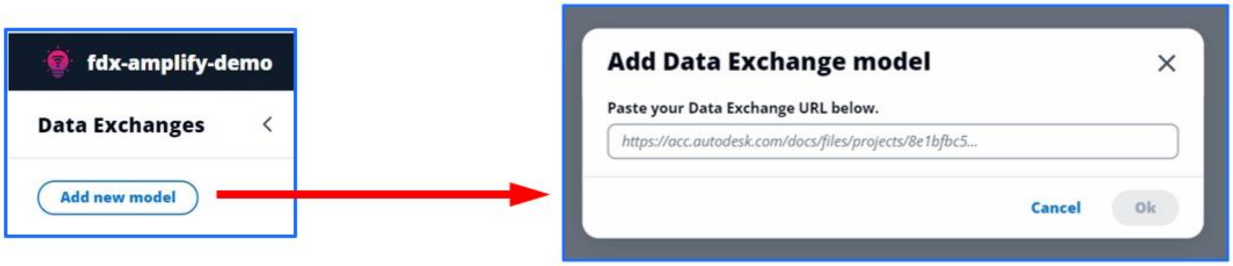
```
import { API, graphqlOperation } from “aws-amplify”;
import { createModel } from “../graphql/mutations”;

const addModel = await API.graphql(
   graphqlOperation(createModel, { input: newModel }));
```

Create new model with only 3 lines of code

4



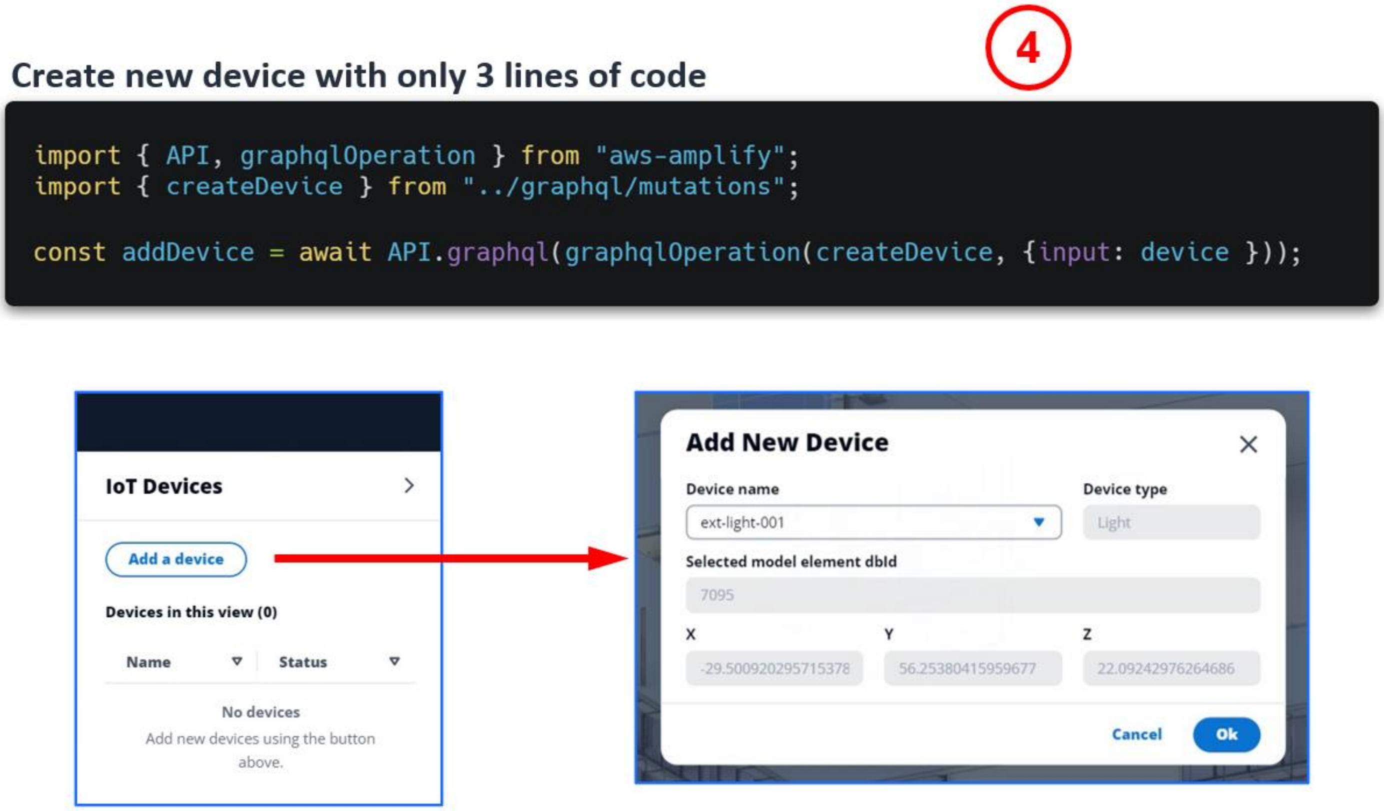
```
import { API, graphqlOperation } from "aws-amplify";
import { createModel } from "../graphql/mutations";

const addModel = await API.graphql( graphqlOperation(createModel, { input: newModel }));
```


fdx-amplify-demo
Data Exchanges
Add new model
Add Data Exchange model
Paste your Data Exchange URL below.
https://acc.autodesk.com/docs/files/projects/8e1bfbc5...
Cancel
Ok

4. Connect Frontend to API - Mutations

Similarly, for the “Add a device” button:

```
import { API, graphqlOperation } from “aws-amplify”;
import { createModel } from “../graphql/mutations”;

const addDevice = await API.graphql(
   graphqlOperation(createDevice, { input: device }));
```



Create new device with only 3 lines of code

4

```
import { API, graphqlOperation } from "aws-amplify";
import { createDevice } from "../graphql/mutations";

const addDevice = await API.graphql(graphqlOperation(createDevice, {input: device }));
```

IoT Devices

Devices in this view (0)

No devices

Add new devices using the button above.

Add New Device

Device name: ext-light-001

Device type: Light

Selected model element dbId: 7095

X: -29.500920295715378

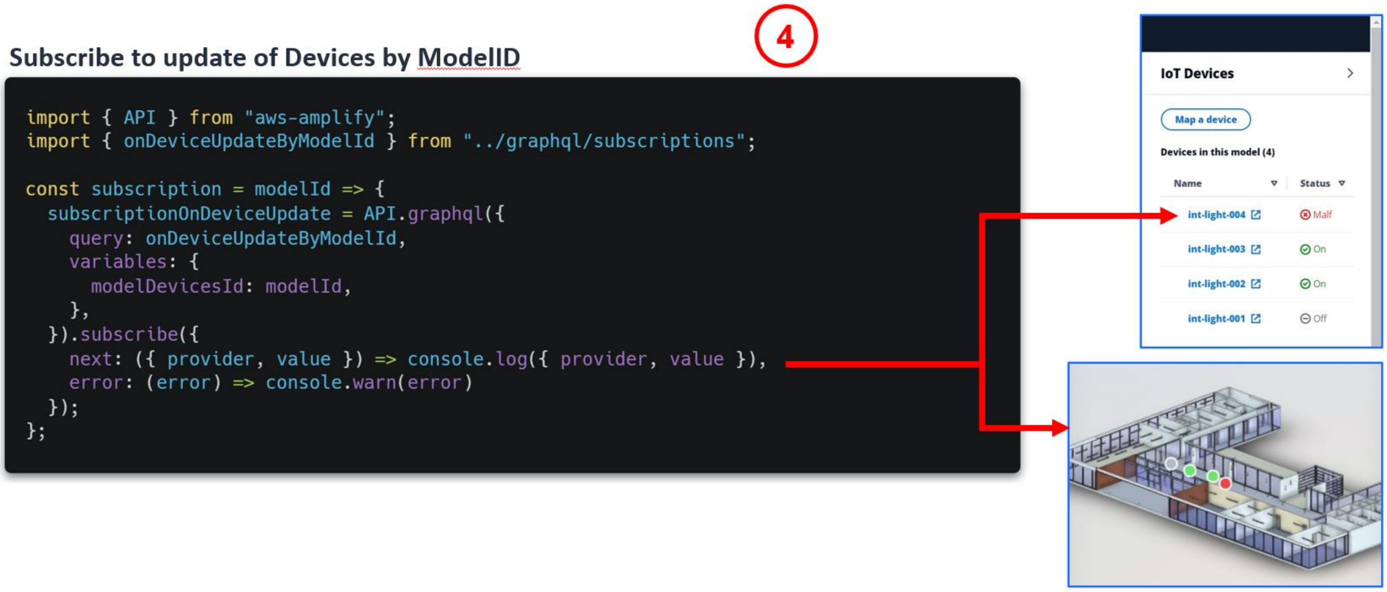
Y: 56.25380415959677

Z: 22.09242976264686

4. Connect Frontend to API - Subscriptions

To listen to real-time device state changes, register a subscription from the client with the code below. This updates both the device list and the viewer in the UI:

```
import { API } from “aws-amplify”;
import { onDeviceUpdatebyModelId } from “../graphql/subscriptions”;

const subscription = modelId => {
   subscriptionOnDeviceUpdate = API.graphql({
      query: onDeviceUpdateByModelId,
      variables: {
         modelDevicesId: modelId,
      },
   }).subscribe({
      next: ({provider, value }) => console.log({ provider, value }),
      error: (error) => console.warn(error)
   });
};
```



Subscribe to update of Devices by ModelID

```
import { API } from "aws-amplify";
import { onDeviceUpdateByModelId } from "../graphql/subscriptions";

const subscription = modelId => {
  subscriptionOnDeviceUpdate = API.graphql({
    query: onDeviceUpdateByModelId,
    variables: {
      modelDevicesId: modelId,
    },
  }).subscribe({
    next: ({ provider, value }) => console.log({ provider, value }),
    error: (error) => console.warn(error)
  });
};
```

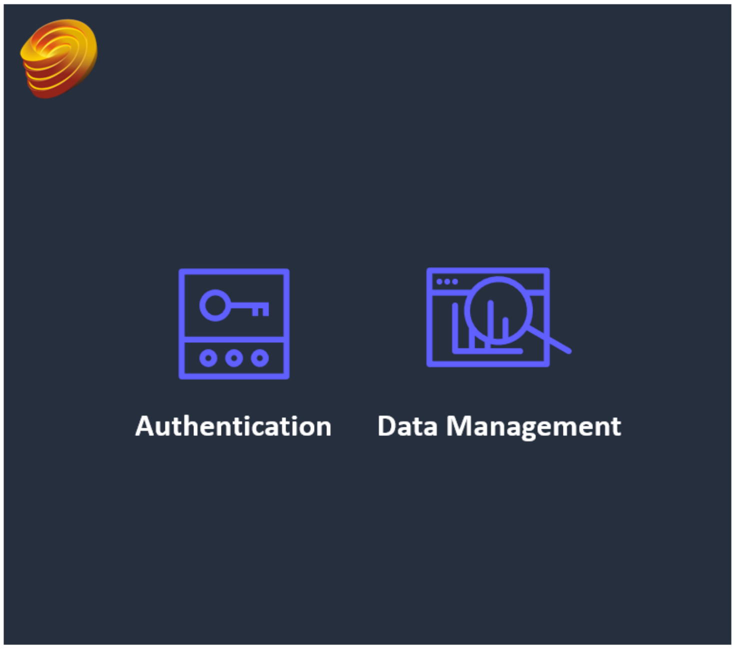
IoT Devices

Devices in this model (4)

Name	Status
int-light-004	Malf
int-light-003	On
int-light-002	On
int-light-001	Off

Add Forge API

For backend functionality of the app, two components of Forge API are referenced:



Authentication API

- Gets a 3-legged token with implicit grant

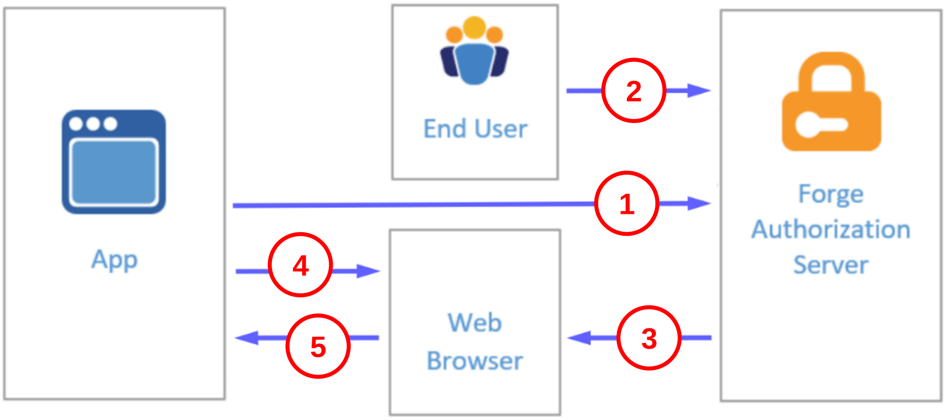
Data Management API

- Gets the latest Data Exchange model version
- Gets the Data Exchange model name

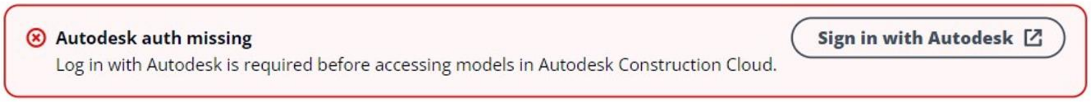
Get 3-Legged Token

When a user logs into the Amplify app, they are first authenticated using their AWS Account. To be able to fetch Data Exchanges, the Amplify app needs authentication to the users Forge account (through their Autodesk account).This is done by implementing the **Autodesk Authorization Web Flow** using the 3-legged token with implicit grant:

https://forge.autodesk.com/en/docs/oauth/v2/tutorials/get-3-legged-token-implicit



1. App directs end user's browser to the Autodesk sign-in page:

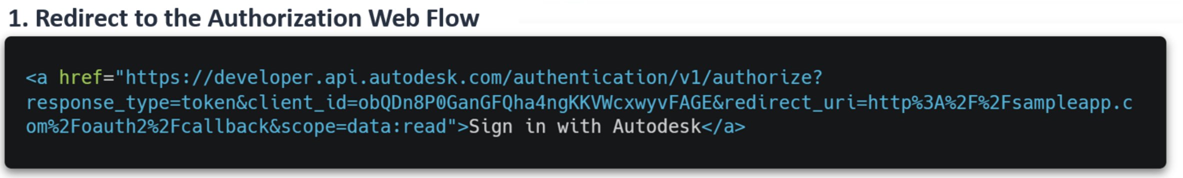


Autodesk auth missing

Log in with Autodesk is required before accessing models in Autodesk Construction Cloud.

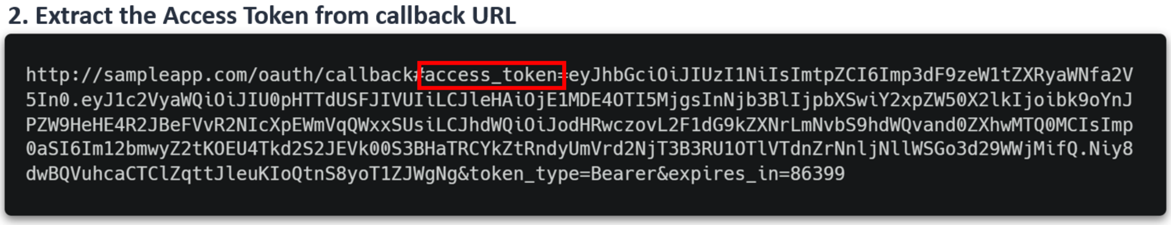
2. User logs in with their Autodesk credentials.
3. Forge Authorization Server redirects user to original URL with appended access token.
4. Upon a page reload, access token is extracted with JavaScript.
5. Access token is passed to the app.

As a result, the end user's Autodesk credentials are never exposed to the web app. In addition, the access token is used to make any subsequent API calls to Forge.



1. Redirect to the Authorization Web Flow

```
<a href="https://developer.api.autodesk.com/authentication/v1/authorize?
response_type=token&client_id=obQDn8P0GanGFQha4ngKKVWcxwyvFAGE&redirect_uri=http%3A%2F%2Fsampleapp.c
om%2Foauth2%2Fcallback&scope=data:read">Sign in with Autodesk</a>
```



2. Extract the Access Token from callback URL

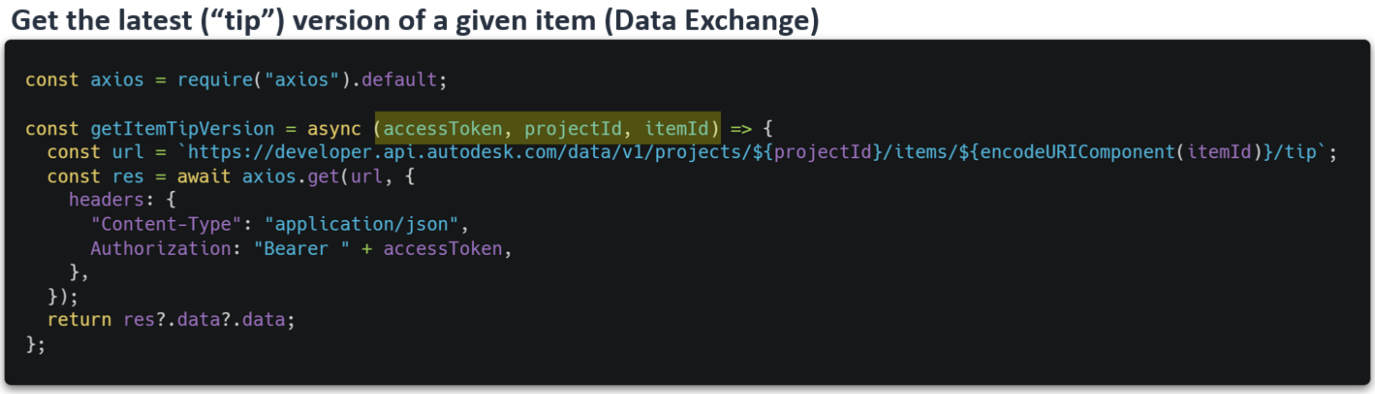
```
http://sampleapp.com/oauth/callback#access_token=eyJhbGciOiJIUzI1NiIsImtpZCI6Imp3dF9zeW1tZXRyaWNfa2V
5In0.eyJ1c2VyaWQiOiJIU0pHTTdUSFJIVUIiLCJleHAiOjE1MDE4OTI5MjgsInNjb3BlIjpbXSwiY2xpZW50X2lkIjoibk9oYnJ
PZW9HeHE4R2JBeFVvR2NIcXpEWmVqQWxxSUsiLCJhdWQiOiJodHRwczovL2F1dG9kZXNrLmNvbS9hdWQvand0ZXhwMTQ0MCIsImp
0aSI6Im12bmwyZ2tKOEU4Tkd2S2JEVk00S3BHaTRCYkZtRndyUmVrd2NjT3B3RU1OTlVTdnZrNnljNllWSGo3d29WWjMifQ.Niy8
dwBQVuhcaCTClZqttJleuKIoQtnS8yoT1ZJWgNg&token_type=Bearer&expires_in=86399
```

Get Data Exchange Info

To add a new model to the app, the user pastes the Data Exchange URL that was shared by the Architect:

```
https://acc.autodesk.com/docs/files/projects/8e1bfbc5-7807-4190-90ba-
429cb434cd9c?folderUrn=urn%3Aadsk.wipprod%3Afs.folder%3Aco.cFlLi3uETZWniC1
FJUibvg&entityId=urn%3Aadsk.wipprod%3Adm.lineage%3AZE5IZ5jYTuevlNQMmJ5igw&
viewModel=detail&moduleId=folders
```

The URL is parsed for the **ProjectID** and **EntityID** values. In conjunction with the access token, the Data Management API is called to retrieve the latest version and name of the model referenced in the URL.

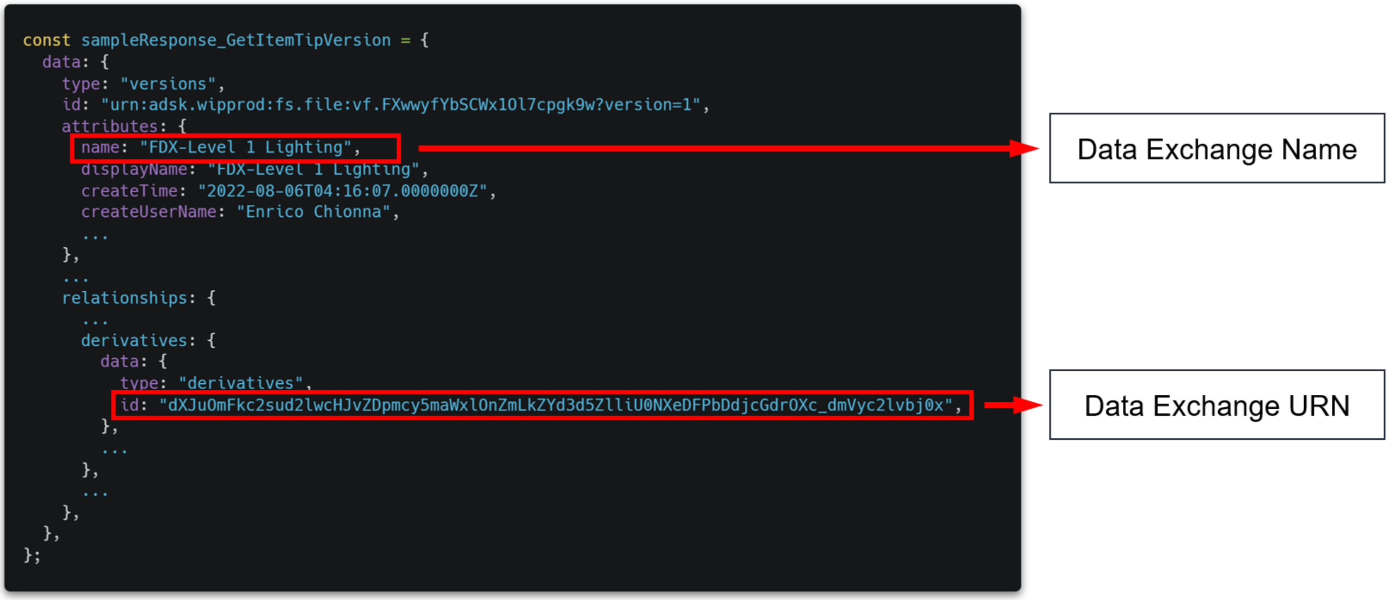


Get the latest ("tip") version of a given item (Data Exchange)

```
const axios = require("axios").default;

const getItemTipVersion = async (accessToken, projectId, itemId) => {
  const url = `https://developer.api.autodesk.com/data/v1/projects/${projectId}/items/${encodeURIComponent(itemId)}/tip`;
  const res = await axios.get(url, {
    headers: {
      "Content-Type": "application/json",
      Authorization: "Bearer " + accessToken,
    },
  });
  return res?.data?.data;
};
```

Get Data Exchange Info – Sample Response

This is a sample API response from Forge after getting the “Tip” or latest version of an item on ACC:



```
const sampleResponse_GetItemTipVersion = {
  data: {
    type: "versions",
    id: "urn:adsk.wipprod:fs.file:vf.FXwwyfYbSCWx1Ol7cpgk9w?version=1",
    attributes: {
      name: "FDX-Level 1 Lighting",
      displayName: "FDX-Level 1 Lighting",
      createTime: "2022-08-06T04:16:07.0000000Z",
      createUserName: "Enrico Chionna",
      ...
    },
    ...
    relationships: {
      ...
      derivatives: {
        data: {
          type: "derivatives",
          id: "dXJuOmFkc2sud2lwcHJvZDpmcy5maWxlOnZmLkZYd3d5ZlliU0NXeDFPbDdjcGdrOXc_dmVyc2lvbj0x",
        },
        ...
      },
      ...
    },
  },
};
```

Add Forge Viewer & Sprites

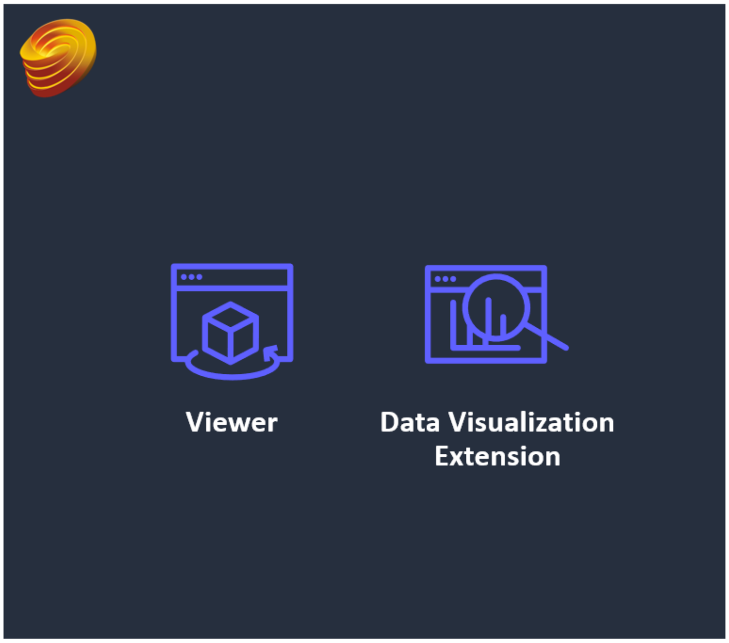
For the app’s frontend, we implement two components:

Forge Viewer

- Loads models in the viewport of the app.

Data Visualization Extension

- Adds sprites to the model representing IoT devices

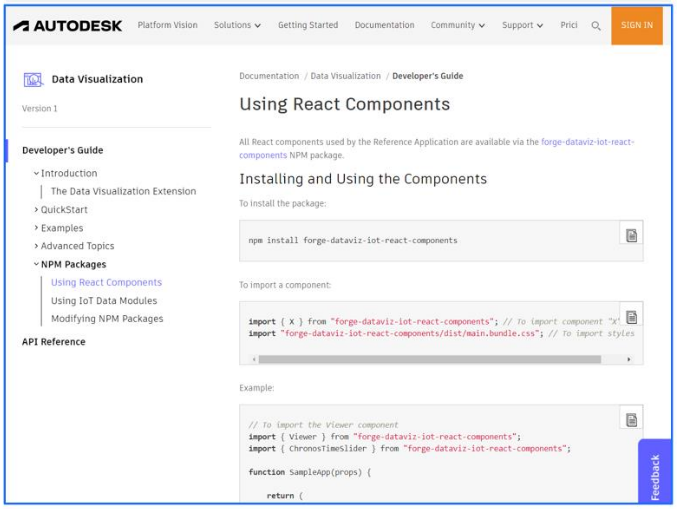


Add Forge Viewer & Sprites

This link points to the Forge Data Visualization library that contains step-by-step instructions on how to implement the viewer in your application and how to show IoT devices using the extension:

https://forge.autodesk.com/en/docs/dataviz/v1/developers_guide/introduction

Viewer



Documentation / Data Visualization / Developer's Guide

Using React Components

All React components used by the Reference Application are available via the forge-dataviz-iot-react-components NPM package.

Installing and Using the Components

To install the package:

```
npm install forge-dataviz-iot-react-components
```

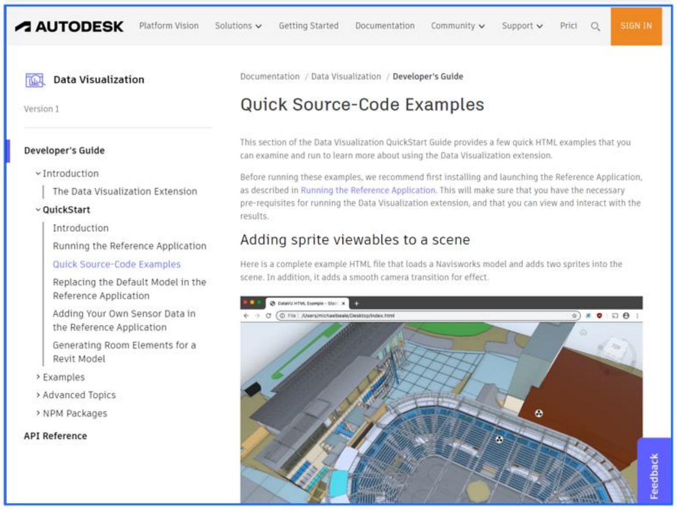
To import a component:

```
import { X } from "forge-dataviz-iot-react-components"; // To import component "X"
import "forge-dataviz-iot-react-components/dist/main.bundle.css"; // To import styles
```

Example:

```
// To import the Viewer component
import { Viewer } from "forge-dataviz-iot-react-components";
import { ChronosTimeSlider } from "forge-dataviz-iot-react-components";

function SampleApp(props) {

    return (
```

Data Visualization Extension (Sprites)


AUTODESK
Platform Vision
Solutions
Getting Started
Documentation
Community
Support
Prici
SIGN IN
Data Visualization
Version 1
Developer's Guide
Introduction
The Data Visualization Extension
QuickStart
Introduction
Running the Reference Application
Quick Source-Code Examples
Replacing the Default Model in the Reference Application
Adding Your Own Sensor Data in the Reference Application
Generating Room Elements for a Revit Model
Examples
Advanced Topics
NPM Packages
API Reference
Feedback

Documentation / Data Visualization / Developer's Guide

Quick Source-Code Examples

This section of the Data Visualization QuickStart Guide provides a few quick HTML examples that you can examine and run to learn more about using the Data Visualization extension.

Before running these examples, we recommend first installing and launching the Reference Application, as described in Running the Reference Application. This will make sure that you have the necessary pre-requisites for running the Data Visualization extension, and that you can view and interact with the results.

Adding sprite viewables to a scene

Here is a complete example HTML file that loads a Navisworks model and adds two sprites into the scene. In addition, it adds a smooth camera transition for effect.

Autodesk Costs

Costs are considered from the perspective of the author of models and Data Exchanges, such as an Architect, and the consumer, such as a Vendor.

Architect (author):

- Incurs costs associated with Revit and ACC licensing to design in Revit and host the models in the Autodesk cloud.
- Incurs no Forge costs.

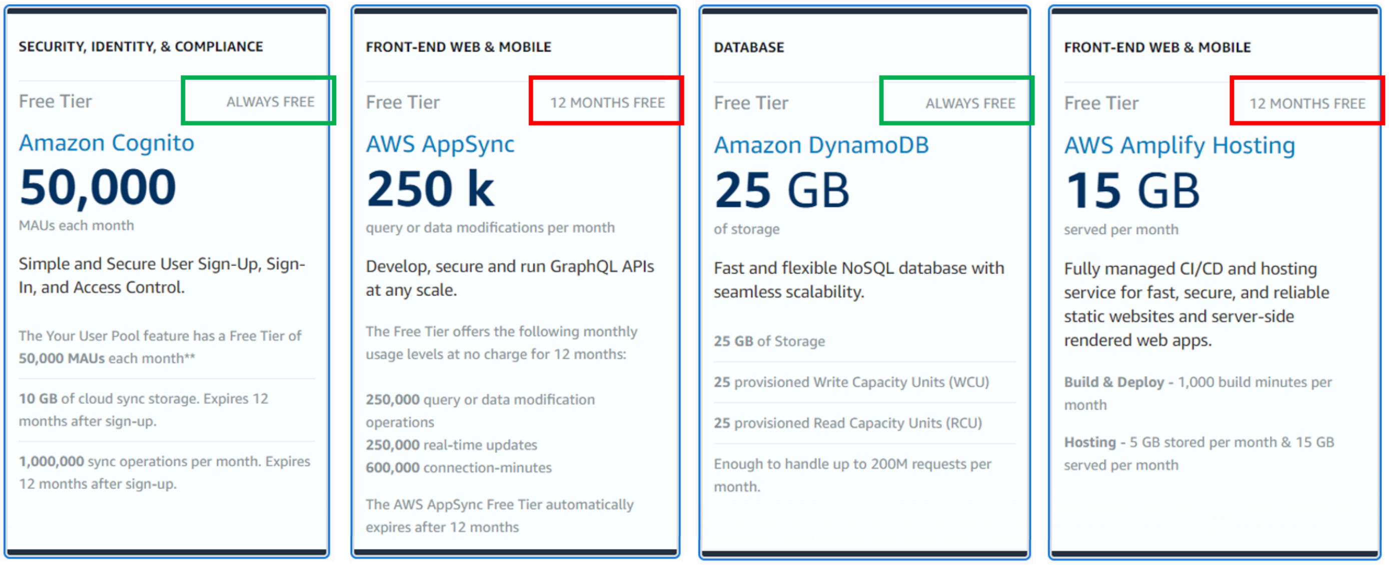
Vendor (consumer):

- Incurs no costs Revit or ACC costs, as the Vendor does not author or store models.
- A Forge account is needed and Forge costs are not incurred by the example application described in this guide.

Revit		ACC / Docs		Forge	
Architect:	PAID	**Architect:**	PAID	**Architect:**	FREE
Vendor:	FREE	**Vendor:**	FREE	**Vendor:**	FREE

AWS Costs

Only the vendor will need an AWS account. The services used by this app are managed by AWS Amplify, and **there is no cost for using the Amplify Framework**. AWS costs are only incurred by the underlying AWS services used, which could also be free if running on the Free Tier. Visit https://aws.amazon.com/free or see below for details.



SECURITY, IDENTITY, & COMPLIANCE

Free Tier ALWAYS FREE

Amazon Cognito

50,000

MAUs each month

Simple and Secure User Sign-Up, Sign-In, and Access Control.

The Your User Pool feature has a Free Tier of **50,000 MAUs** each month**

10 GB of cloud sync storage. Expires 12 months after sign-up.

1,000,000 sync operations per month. Expires 12 months after sign-up.

FRONT-END WEB & MOBILE

Free Tier 12 MONTHS FREE

AWS AppSync

250 k

query or data modifications per month

Develop, secure and run GraphQL APIs at any scale.

The Free Tier offers the following monthly usage levels at no charge for 12 months:

250,000 query or data modification operations
250,000 real-time updates
600,000 connection-minutes

The AWS AppSync Free Tier automatically expires after 12 months

DATABASE

Free Tier ALWAYS FREE

Amazon DynamoDB

25 GB

of storage

Fast and flexible NoSQL database with seamless scalability.

25 GB of Storage

25 provisioned Write Capacity Units (WCU)

25 provisioned Read Capacity Units (RCU)

Enough to handle up to 200M requests per month.

FRONT-END WEB & MOBILE

Free Tier 12 MONTHS FREE

AWS Amplify Hosting

15 GB

served per month

Fully managed CI/CD and hosting service for fast, secure, and reliable static websites and server-side rendered web apps.

Build & Deploy - 1,000 build minutes per month

Hosting - 5 GB stored per month & 15 GB served per month

Resources and References

- Autodesk Forge:
  - https://forge.autodesk.com
  -
- Autodesk Data Exchange:
  - https://forge.autodesk.com/blog/data-exchange-released-forge-data-exchange-apis-now-available-public-beta
  - https://forge.autodesk.com/en/docs/fdx/v1/developers_guide/fd_overview
  - https://forge.autodesk.com/en/docs/data/v2/reference/http/projects-project_id-items-item_id-tip-GET

- Autodesk Authentication:
  - https://forge.autodesk.com/en/docs/oauth/v2/tutorials/get-3-legged-token-implicit

- Autodesk Forge Viewer:
  - https://forge.autodesk.com/en/docs/viewer/v7/developers_guide/overview
  - https://forge.autodesk.com/en/docs/dataviz/v1/developers_guide/introduction

- Amplify Framework Documentation
  - https://ui.docs.amplify.aws

- Amplify UI:
  - https://ui.docs.amplify.aws

- Figma:
  - https://www.figma.com

- React:
  - https://reactjs.org

- GraphQL:
  - https://graphql.org
  - https://aws.amazon.com/graphql
  - https://aws.amazon.com/graphql/graphql-dynamodb-data-modeling

- Cloudscape:
  - https://cloudscape.design

- AWS Free Tier:
  - https://aws.amazon.com/free