

**EÖTVÖS LORÁND UNIVERSITY FACULTY OF
INFORMATICS DEPARTMENT OF PROGRAMMING
LANGUAGES AND COMPILERS**



AthMeet Fitness Social Media Application

László Ildikó
Supervisor

Na'il Garba
Computer Science BSc

Table of Contents

1. Introduction
 - a. Motivation
 - b. Thesis Structure
2. User Documentation
 - a. Project Description
 - b. Bottom Tabs
 - i. Feed
 - ii. Profiles
 - iii. Search
 - iv. AthleteFinder
 - c. Sign In and Sign Up
 - d. Messages
3. Developer Documentation
 - a. React Native
 - b. Expo
 - c. Backend
 - i. AWS and Amplify Command Line Interface
 - ii. GraphQL
 - iii. DynamoDB
 - d. Class Diagrams
 - e. Test Cases
4. Bibliography

Introduction:

There are only a few platforms that focus on fitness in a social way, and as a person who spends 8 hours per week in the gym, I decided to develop an application that encourages people to share their passion for fitness. The fitness lifestyle is something many developers strive to improve with their own applications. People have an innate desire to improve their bodies both mentally and physically, however, the latter is often overlooked due to laziness and lack of drive. Often, people are overwhelmed by their lack of knowledge which ultimately ends up leading them to unhealthy lifestyles. The few platforms that focus on fitness in a social way are mainly in the form of online forums, so a social media application would allow this sector to grow. A social media style application that encourages people to educate themselves, as well as facilitates the partaking of physical activities will be something that can influence people into leading better lifestyles for themselves.

AthMeet is intended to be an application designed to help those interested in fitness to socialize with other people who have similar interests and goals. Users of the application will be welcomed with posts primarily pertaining to fitness and a healthy lifestyle. These users will be able to interact with each other, share educational posts, and even connect with users who are looking for partners in their activities.

This application will be designed in a way that will be familiar to users of popular social media applications such as Twitter, Instagram, and Facebook. Following a similar style and layout will reduce the make the application feel natural since users already have an understanding of the basic principles of the app's design. AthMeet is built using React Native, a Javascript framework built by Facebook, which provides an efficient and simple framework for developing user interfaces for mobile applications. The backend of the application makes use of Amazon Web Services (AWS), a cloud computing platform produced and maintained by Amazon. The use of a cloud platform greatly reduces the amount of work needed to maintain the database for the application, and AWS provides services that are highly compatible with React Native.

AthMeet incorporates the standard features that comprise any other social media application, as well as its key differentiator, the AthleteFinder. The application allows for the posting and sharing of content, the following of user profiles, and private messaging.

AthMeet features a feed that shows users the posts of accounts they are following, and allows them to share posts with other accounts. There is an explore page that features posts from profiles that the user is not following, which will introduce users to new content that they may find interesting. The AthletesMeet tab is a specialized feature that finds people who the user can potentially perform activities with. It uses an algorithm that checks for mutual interests and goals and pairs users with those it calculates to have a high compatibility with.

User Documentation:

Project Description:

The main objective of this project is to create an easily navigable social media app, that can also serve as a meeting ground for athletes. Users will be able to create a profile that displays their athletic interests and information about their training, including their level of progression in their sport, as well as where they primarily train at. This information is readily available to all users in order to facilitate the meeting process, and provides key information when it comes to the AthleteFinder.

AthMeet at its core is still a social media application, so a feed of posts from a user's subscribed profiles is on the front page. Each post contains text and can feature an image of the user's choice, and it can be upvoted as well as shared to other users in the private messages.

AthMeet is created through React Native, a framework that offers developers tools for creating mobile applications with JavaScript at its core. React Native is developer friendly and greatly simplifies the development process of applications by giving templates of common application layouts such as the Bottom Tab Navigator that is used for AthMeet. React Native also allows applications to be used across multiple platforms including iOS, Android, and web browsers.

This project makes use of cloud services so as to avoid purchasing and upkeeping servers. Firebase is a Google hosted cloud service that provides all the necessary storage and functions for this project at no additional cost.

Bottom Tabs

AthMeet uses a bottom tab navigation style so as to facilitate the user experience. Many major social media applications such as Reddit, Twitter, and Instagram use this feature, and users will recognize the format immediately. The advantage of using this navigation style is that it simplifies the application into distinct

categories. For AthMeet, the bottom tabs are Home, Search, AthleteFinder, and Profile, and the Home tab is the default page after logging in.

The Home tab will house the main feed containing posts made by users. The user can scroll down to continue reading posts that are pulled from the server. Featured at the top of the Home tab is the private messages button, which when pressed will navigate users to the private messages screen. This screen contains a list of all ongoing conversations that the user is engaged in, and pressing on a conversation will take the user to the screen of the conversation.

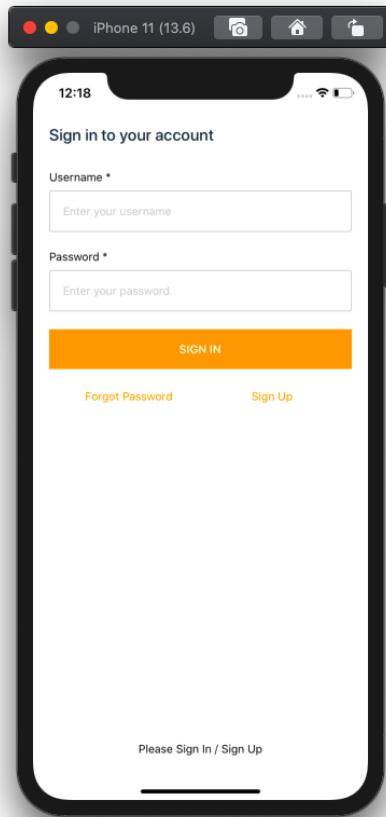
The Search tab will offer the user the option to search for other profiles by typing in their username or profile name.

The AthleteFinder tab will be a unique feature of AthMeet which caters to those looking for partners in particular sports and activities. In this tab, users will be shown a list of profiles with matching interests. This tab takes into account the different personal details that users save to their profile including main and side sports, level of progression in their sports, and their selected gyms. The Athlete finder is meant to connect people who may have similar training regimens, so the level of progression is important here. Furthermore, the page features a filter button that navigates to another screen where the user will be able to change what they are looking for from the AthleteFinder, and this may be a change in sport, level, or gym.

The Profile tab shows the user their own profile in the way it would be shown to other users. This includes their main profile attributes, their following and followers, and a list of posts they have posted. On this page there is an edit button that will allow the user to quickly make changes to their profile. The button will navigate to another screen where the user may change their name, profile picture, gym, sports, and level.

Sign In and Sign Up

Upon launching AthMeet, users are greeted with the Sign In screen, and are prompted to enter a username and password.



Sign In screen

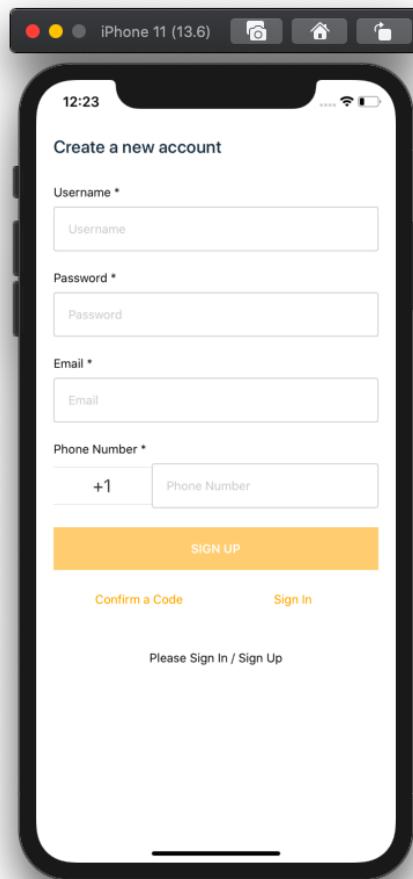
For existing users, entering their credentials and pressing the Sign In button will send their credentials to AWS Cognito user pool for verification. If the verification is successful, the user will be sent to the AthMeet home screen, and be shown their feed. If verification fails, the user may choose to reset their password by pressing the Forgot Password button.

For new users, pressing the Sign Up button will navigate them to the new user registration screen, where they will be prompted for an email, username, password, and phone number in order to create a new account in the user pool.

For existing users, entering their credentials and pressing the Sign In button will send their credentials to AWS Cognito user pool for verification. If the verification is successful, the user will be sent to the AthMeet home screen, and be shown their feed.

If verification fails or the user has forgotten their password, they may choose to reset their password by pressing the Forgot Password button.

For new users, pressing the Sign Up button will navigate them to the new user registration screen, where they will be prompted for an email, username, password, and phone number in order to create a new account in the user pool.



Sign Up screen

By creating an account, a new user is registered in the cognito user pool, and upon its first sign in, the account will then be registered into the AthMeet database.

Developer Documentation:

Tools used

Reactnative

Expo

Yarn

Nodejs

Android studio

Developer Documentation

a. Frontend

i. React Native

ii. Expo

b. Backend

i. AWS and Amplify CLI

ii. GraphQL

iii. DynamoDB

c. Class Diagrams

d. Test Cases

3. Developer Documentation

3.b Backend

i. AWS and Amplify Command Line Interface

The backend for AthMeet relies on Amazon Web Services (AWS) which provides us with a cloud computing platform and API service that can be easily integrated with React Native applications. This application uses GraphQL API which works with AWS AppSync to link with Amazon DynamoDB and AWS Lambda. User authentication services such as the sign in and sign out processes are automatically handled through AWS Authentication, and is managed through Cognito User Pools.

Amplify Command Line Interface (CLI) is the toolchain that is installed onto AthMeet in order to install and manage the various AWS cloud services that are used.

Amplify is used every time a database change occurs and needs to be pushed or pulled between the DynamoDB and the project. It is also responsible for generating new GraphQL code when changes occur in the schema.

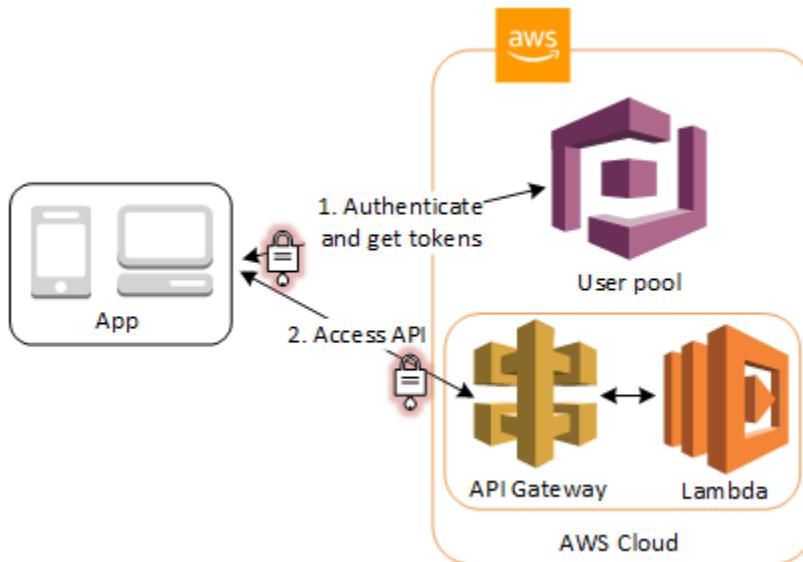
Amplify and AWS AppSync work together to manage the backend through the AWS Management Console that can be accessed through a web browser. The console provides a user-friendly interface that allows us to change the schema, functions and queries, and manage various API settings for our project. It also provides an interface for Cognito and DynamoDB, giving us full control of the backend.

AWS makes implementing the entire authentication process very simple. First, we install Amplify CLI by calling “amplify init” in the console. Once Amplify is installed, calling “amplify add auth” will install all the necessary dependencies from ‘aws-amplify’ onto the project. This step only adds in the necessary files needed to connect to the Cognito user pool. To connect with our database we need to add the GraphQL API by calling “amplify add api” and configuring our API settings.

This command provides us with the option to generate a GraphQL schema for the project. This can then be modified into a schema that suits the needs of the project.

ii. Authentication

Authentication is also done through AWS. Upon a user sign in request, AWS sends the user credentials to the Cognito user pool for verification and authentication. Once the authentication is complete the API is connected to the AWS API cloud gateway, and the app can then interact with the database.



For authentication, the App.tsx file must be slightly modified in order to allow AWS to wrap the application. We do this by configuring Amplify and the API, and then exporting the default App function by passing it through the AWS withAuthenticator() function

```

16  Amplify.configure(AMPLIFY_CONFIG)
17  API.configure(AMPLIFY_CONFIG)
18
19
20
21
22 > function App() { ...
23 }
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59

```

App.tsx- Amplify and API configuration and withAuthenticator() wrapping.

Once authenticated by Cognito, the Cognito user information is stored in the userInfo constant. In the case of a newly created user, the application must create a new user profile in the database in the App() function.

```

22  function App() {
23    const isLoadingComplete = useCachedResources();
24    const colorScheme = useColorScheme();
25
26    const saveUserToDB = async (user:createUserInput) =>{
27      console.log(user);
28      await API.graphql(graphqlOperation(createUser, {input: user}))
29    }
30
31    useEffect(() => {
32      const updateUser = async () => {
33        // Get current authenticated user
34        const userInfo = await Auth.currentAuthenticatedUser({ bypassCache: true });
35
36        if(userInfo) {
37          // Check if user already exists in database
38          // If it doesn't, create the user in the database
39          const userData = await API.graphql(graphqlOperation(getUser, { id: userInfo.attributes.sub }));
40          console.log(userData)
41          if(!userData.getUser) {
42            const user = {
43              id: userInfo.attributes.sub,
44              username: userInfo.username,
45              name: userInfo.username,
46              email: userInfo.attributes.email,
47              image: ''
48            }
49            await saveUserToDB(user);
50          } else {
51            console.log('User already exists');
52          }
53        }
54      }
55      updateUser();
56    }, [])
57
58
59

```

App() function with user authentication and creation

The user's database userID will be matched with the Cognito userID found in the userInfo.attributes.sub, and the new user variable is initialized with all the other necessary attributes that have already been registered in Cognito. The user variable is then created in the database.

iii. GraphQL

GraphQL is the data query language for the API used by AthMeet. GraphQL allows for the client to request the exact data required, which makes development of queries and management of data much simpler. The schema allows us to create queries that fetch detailed information, and doesn't require additional content filtration. While GraphQL doesn't excel at file uploading, this application does not make significant use of file uploads aside from pictures, so there is no real drawback to using GraphQL.

```

amplify > backend > api > dev > schema.graphql
  1 type User @model {
  2   id: ID!
  3   username: String!
  4   name: String!
  5   email: String!
  6   image: String
  7   posts: [Post] @connection(keyName: "byUser", fields: ["id"])
  8   following: [User]
  9   followers: [User]
 10   mainGym: String
 11   mainSport: String
 12   level: String
 13   chatRoomUser: [ChatRoomUser] @connection(keyName: "byUser", fields: ["id"])
 14 }
 15
 16 type Post @model @key(name: "byUser", fields: ["userID"]) {
 17   id: ID!
 18   content: String!
 19   userID: ID!
 20   user: User @connection(fields: ["userID"])
 21   image: String
 22   likes: [Like] @connection(keyName: "byPost", fields: ["id"])
 23   comments: [Comment] @connection(keyName: "byPost", fields: ["id"])
 24 }
 25
 26 type Comment @model @key(name: "byPost", fields: ["postID", "content"]) {
 27   id: ID!
 28   postID: ID!
 29   post: Post! @connection(fields: ["postID"])
 30   content: String!
 31 }
 32
 33 type Like
 34   @model(queries:null)
 35   @key(name: "byUser", fields: ["userID", "postID"])
 36   @key(name: "byPost", fields: ["postID", "userID"]) {
 37     id: ID!
 38     userID: ID!
 39     postID: ID!
 40     user: User! @connection(fields: ["userID"])
 41     post: Post! @connection(fields: ["postID"])
 42   }
 43
 44 type ChatRoomUser
 45   @model
 46   @key(name: "byUser", fields: ["userID", "chatRoomID"])
 47   @key(name: "byChatRoom", fields: ["chatRoomID", "userID"]) {
 48     id: ID!
 49     userID: ID!
 50     chatRoomID: ID!
 51     user: User @connection(fields: ["userID"])
 52     chatRoom: ChatRoom @connection(fields: ["chatRoomID"])
 53   }
 54
 55 type ChatRoom @model {
 56   id: ID!
 57   chatRoomUsers: [ChatRoomUser] @connection(keyName: "byChatRoom", fields: ["id"])
 58   messages: [Message] @connection(keyName: "byChatRoom", fields: ["id"])
 59   lastMessageID: ID!
 60   lastMessage: Message @connection(fields: ["lastMessageID"])
 61 }
 62
 63 type Message
 64   @model
 65   @key(
 66     name: "byChatRoom",
 67     fields: ["chatRoomID", "createdAt"],
 68     queryField: "messagesByChatRoom"
 69   ) {
 70     id: ID!
 71     createdAt: String!
 72     content: String!
 73     userID: ID!
 74     chatRoomID: ID!
 75     user: User @connection(fields: ["userID"])
 76     chatRoom: ChatRoom @connection(fields: ["chatRoomID"])
 77     post: Post
 78   }

```

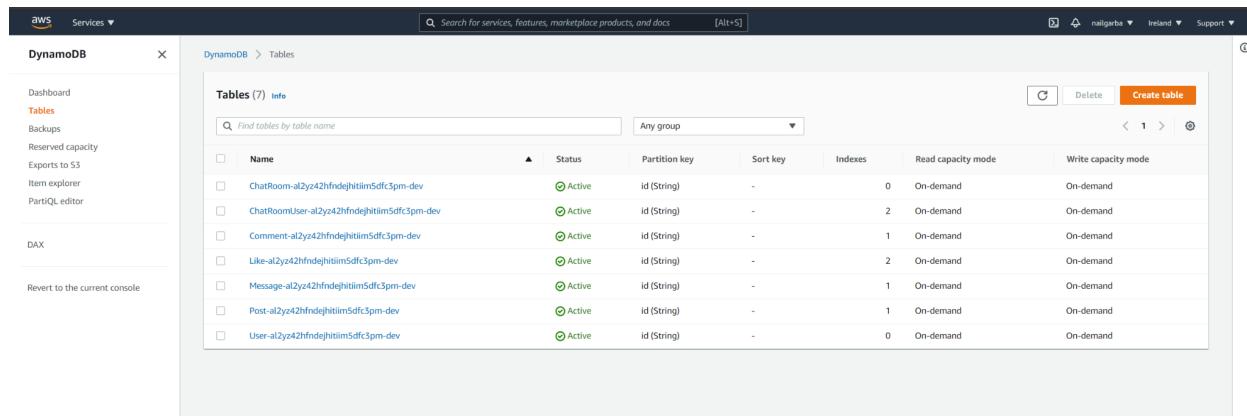
Amplify GraphQL schema

The User table contains information on the users' profile information, and is linked to every other table.

Once “amplify push” is called in the console, AWS will upload the schema to the cloud and create the appropriate DynamoDB tables for the schema. This will also generate the necessary mutation, queries, and subscription code in the graphql folder. Once the files are imported they can be used freely in the source code to modify the database when using the application.

iv. DynamoDB

DynamoDB is the database provided by AWS which houses all AthMeet content, and holds tables that store users, posts, likes, and comments. With each post or update in the application, a query or mutation is made through the GraphQL API which then transfers to the database. Each item in the tables has a unique ID that is generated by DynamoDb, and the tables are linked together using these IDs according to the schema. We can view and edit the tables and their individual elements through the AWS console.



A screenshot of the AWS DynamoDB console. The left sidebar shows navigation links for Dashboard, Tables (which is selected), Backups, Reserved capacity, Exports to S3, Item explorer, and PartiQL editor. Below that is a DAX section with a 'Revert to the current console' link. The main content area is titled 'Tables (7) Info'. It features a search bar and a dropdown menu for 'Any group'. A table lists seven tables: ChatRoom, ChatRoomUser, Comment, Like, Message, Post, and User, each with its status (Active), partition key (id), sort key, indexes, and read/write capacity modes (all set to On-demand).

Name	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacity mode
ChatRoom-a1zy242hfndejhitiim5dfc3pm-dev	Active	id (String)	-	0	On-demand	On-demand
ChatRoomUser-a1zy242hfndejhitiim5dfc3pm-dev	Active	id (String)	-	2	On-demand	On-demand
Comment-a1zy242hfndejhitiim5dfc3pm-dev	Active	id (String)	-	1	On-demand	On-demand
Like-a1zy242hfndejhitiim5dfc3pm-dev	Active	id (String)	-	2	On-demand	On-demand
Message-a1zy242hfndejhitiim5dfc3pm-dev	Active	id (String)	-	1	On-demand	On-demand
Post-a1zy242hfndejhitiim5dfc3pm-dev	Active	id (String)	-	1	On-demand	On-demand
User-a1zy242hfndejhitiim5dfc3pm-dev	Active	id (String)	-	0	On-demand	On-demand

List of tables in AWS Console

The screenshot shows the AWS DynamoDB console with the 'Tables' section selected. A table named 'Post-al2yz42hfndejhiim5dfc3pm-dev' is displayed. The 'General information' section shows the partition key as 'id (String)', sort key as ' - ', capacity mode as 'On-demand', and table status as 'Active' with 'No active alarms'. The 'Items summary' section indicates there are 4 items in the table, each 829 bytes in size, with an average item size of 207.25 bytes. The 'Items preview' section shows 4 items with the following details:

	id	_typename	content	createdAt	image	updatedAt	userID
6b5baef4-b...	6b5baef4-b729-4740-b192-5d5d78fcda0	Post	Cat	2021-05-1...	https://c.fil...	2021-05-1...	22842985-8fa6-4f80-b113-98674b1b3f52
3721d46a-...	3721d46a-...	Post	Another po...	2021-05-1...	<empty>	2021-05-1...	22842985-8fa6-4f80-b113-98674b1b3f52
6597013f-f...	6597013f-f...	Post	Goh	2021-05-1...	aa	2021-05-1...	22842985-8fa6-4f80-b113-98674b1b3f52
a59aa56-f...	a59aa56-f...	Post	Testing pos...	2021-05-1...	aa	2021-05-1...	22842985-8fa6-4f80-b113-98674b1b3f52

Post Table in AWS Console

The screenshot shows the 'Item editor' for a specific item in the 'Post' table. The item has the following attributes:

Attribute name	Value	Type
id - Partition key	6b5baef4-b729-4740-b192-5d5d78fcda0	String
_typename	Post	String
content	Cat	String
image	https://c.files.bbc.co.uk/12A9B/production/_111434467_gettyimages-1143489763.jpg	String
userID	22842985-8fa6-4f80-b113-98674b1b3f52	String
updatedAt	2021-05-19T13:02:04.760Z	String
createdAt	2021-05-19T13:02:04.760Z	String

Buttons at the bottom include 'Cancel' and 'Save changes'.

Individual Post from Post Table in AWS Console

TEST CASES

// remove click on, make less wordy

Login/registration Authentication

Messages

Bottom Tab Navigator		
Test	Goal	Outcome
Click on Home Tab	Navigates to Home Tab screen	
Click on Search Tab	Navigates to Search Tab screen	
Click on AthleteFinder Tab	Navigates to AthleteFinder Tab screen	
Click on Profile Tab	Navigates to Profile Tab screen	

Home Tab		
Test	Goal	Outcome
Click on Messages Icon on the header	Navigates to private messages screen	
Click on New Post button	Navigates to New Post screen	
Click on post's username or name	Navigates to users profile	
Click on Like button	Changes like button color, and adds like to the like count	

Click on share	Brings up tab showing users to share to	
Click on comment	Navigates to the comments screen	

New Post Screen		
Test	Goal	Outcome
Click on content text input box	Brings up keyboard	
Click on image text input box	Brings up keyboard	
Click on Post button	Posts new post, and navigates back to home screen	
Click on Back button	Returns to home screen	

Search Tab		
Test	Goal	Outcome
Click on search bar	Brings up keyboard	
Typing a username and pressing enter	Navigates to profile with matching username	

AthleteFinder Tab		
Test	Goal	Outcome
Click on filter button	Navigates to filter settings screen	
Click on users name or username	Navigates to user's profile	

Profile Tab		
Test	Goal	Outcome
Click on edit button	Navigates to profile settings screen	
Click on following	Navigates to screen showing users followed by the profile	
Click on followers	Navigates to screen showing users following the profile	

4. Bibliography

<https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html>

<https://www.instamobile.io/mobile-development/react-native-aws-amplify/>