## 1. Overview

FitFinder is a customizable shopping experience application built using AWS services. The primary goal of the project is to provide users with a personalized catalog of fashion items based on their individual measurements and style preferences. The solution leverages AWS Lambda functions for, a RDS database for persistent storage, and Amazon SQS for handling asynchronous arduous tasks (web scraping).

* **User Authentication & Authorization:**
  + Users can log in via token-based or username/password methods. Tokens are generated with a configurable expiration time.
* **Account Creation:**
  + New users are prompted to provide a username, password, and measurements (top size, pants waist, pants length, shoe size) along with the gender they wish to shop for.
* **Catalog Customization:**
  + The application provides a personalized catalog based on user details stored in the database.
* **Web Scraping:**
  + A dedicated Lambda function scrapes product information (e.g., titles, prices, links) from ASOS catalogs using BeautifulSoup. The scraper paginates until it detects a repeated product, ensuring that duplicate data is not processed.
* **Task Polling:**
  + Users can poll for the status of long-running web scraping tasks to monitor progress.

## 2. System Architecture

* **AWS Lambda Functions:**
  + Handle discrete tasks such as authentication (/auth), account creation (/make), catalog viewing (/view), web scraping (/scrape), and task polling (/poll).
* **API Gateway:**
  + Acts as the front door for client requests, routing them to the appropriate Lambda functions.
* **Amazon RDS:**
  + Stores all persistent data including user accounts, measurements, catalog items, and web scraping tasks.
* **Amazon SQS:**
  + Manages asynchronous tasks for web scraping by queuing requests and triggering corresponding Lambda functions.
* **Client Application:**
  + A command-line Python application that communicates with the API Gateway endpoints to provide a user interface for authentication, account management, catalog viewing, and other interactions.

### Diagram

A diagram of a diagram

Description automatically generated with medium confidence

## 3. API Specification

### 3.1 Authentication (/auth)

**HTTP Method: POST**

* **Purpose:**
  + Authenticate users using either a token or username/password.
* **Request Formats:** 
  + Token-based:
    - { "body": { "token": "your-token-here" } }
  + Username/Password:
    - { "body": { "username": "your-username", "password": "your-password", "duration": 30 } }
* **Response:** 
  + 200 OK: Returns a user ID (token-based) or a newly generated token (username/password-based).
  + 400 Bad Request: Missing credentials or invalid input format.
  + 401 Unauthorized: Invalid token, expired token, or incorrect username/password.
  + 500 Internal Server Error: Database or server-side error.

### 3.2 Account Creation (/make)

**HTTP Method: POST**

* **Purpose:**
  + Create a new user account.
* **Request Format:** 
  + { "body": {"username": "username1", "password": "password1", "top\_size": "L", "pants\_waist": "32", "pants\_length": "32", "shoe\_size": "10", "gender": "M"}}
* **Response:** 
  + 200 OK: Account created successfully.
  + 400 Bad Request: Missing or invalid parameters.
  + 500 Internal Server Error: Database or server-side error.
* **Note:**
  + User input is validated against the following:
  + Valid Top Sizes: {'XXS', 'XS', 'S', 'M', 'L', 'XL', 'XXL', '3XL'}
  + Pants Waist Range: 24 to 50
  + Pants Length Range: 26 to 40
  + Shoe Size Range: 6 to 15 (accepting half sizes)
  + Valid Genders: {'M', 'F', 'Other'}

### 3.3 Catalog Viewing (/view)

**HTTP Method: GET**

* **Purpose:**
  + Retrieve a personalized catalog based on the authenticated user's data.
* **Query String Parameters:** 
  + token (string): User's authentication token.
  + page (str): Page number of the catalog.
  + includeTops, includeShoes, includePants (flags: 1 or 0)(str): Filters for catalog items.
* **Response:** 
  + 200 OK: Returns a list of catalog items.
  + 400/401/500: Error message with appropriate status code.

### 3.4 Web Scraping (/scrape)

**HTTP Method: POST**

* **Purpose:**
  + Queue a scraping task for an ASOS catalog URL.
* **Request Format:** 
  + { "body": { "url": "https://www.asos.com/..." } }
* **Response:** 
  + 200 OK: Task queued successfully with a job ID.
  + 400/500: Error message.

### 3.5 Task Polling (/poll)

**HTTP Method: GET**

* **Purpose:**
  + Poll the status and progress of a web scraping task.
* **Query String:** 
  + task\_id (string): The ID of the task.
* **Response:** 
  + 200 OK: Returns task details (status and progress).
  + 400/401/500: Error message.

## 4. Database Schema

### 4.1 Users Table

CREATE TABLE users (

userid INT NOT NULL AUTO\_INCREMENT,

username VARCHAR(64) NOT NULL,

pwd VARCHAR(128) NOT NULL,

top\_size ENUM('XXS', 'XS', 'S', 'M', 'L', 'XL', 'XXL', '3XL') NOT NULL,

pants\_waist INT NOT NULL,

pants\_length INT NOT NULL,

shoe\_size FLOAT NOT NULL,

gender ENUM('M', 'F', 'Other') NOT NULL,

PRIMARY KEY (userid),

UNIQUE (username)

);

### 4.2 Items Table

CREATE TABLE items (

itemid INT NOT NULL AUTO\_INCREMENT,

item\_name VARCHAR(64) NOT NULL,

price VARCHAR(16) NOT NULL,

item\_gender ENUM('Men', 'Women', 'Unisex') NOT NULL,

PRIMARY KEY (itemid),

UNIQUE (item\_name)

);

ALTER TABLE items AUTO\_INCREMENT = 10001;

4.3 Sizes Table

CREATE TABLE sizes (

itemid INT NOT NULL,

size VARCHAR(128) NOT NULL,

FOREIGN KEY (itemid) REFERENCES items(itemid)

);

### 4.4 Colors Table

CREATE TABLE colors (

itemid INT NOT NULL,

color VARCHAR(64) NOT NULL,

photo\_url VARCHAR(256) NOT NULL,

FOREIGN KEY (itemid) REFERENCES items(itemid)

);

### 4.5 Scraping Tasks Table

CREATE TABLE scraping\_tasks (

taskid INT NOT NULL AUTO\_INCREMENT,

task\_url VARCHAR(512) NOT NULL,

task\_status VARCHAR(32) NOT NULL,

task\_progress VARCHAR(32) NOT NULL,

PRIMARY KEY (taskid)

);

### 4.6 Tokens Table

CREATE TABLE tokens (

token VARCHAR(36) NOT NULL,

userid INT NOT NULL,

expiration\_utc DATETIME NOT NULL,

PRIMARY KEY (token),

FOREIGN KEY (userid) REFERENCES users(userid)

);

## 5. Setup & Deployment Instructions

**Virtual Environment Setup**

* Create a Virtual Environment:
  + python3 -m venv venv
* Activate the Virtual Environment:
  + macOS/Linux:
    - source venv/bin/activate
  + Windows:
    - venv\Scripts\activate
* Install Dependencies:
  + pip3 install -r requirements.txt