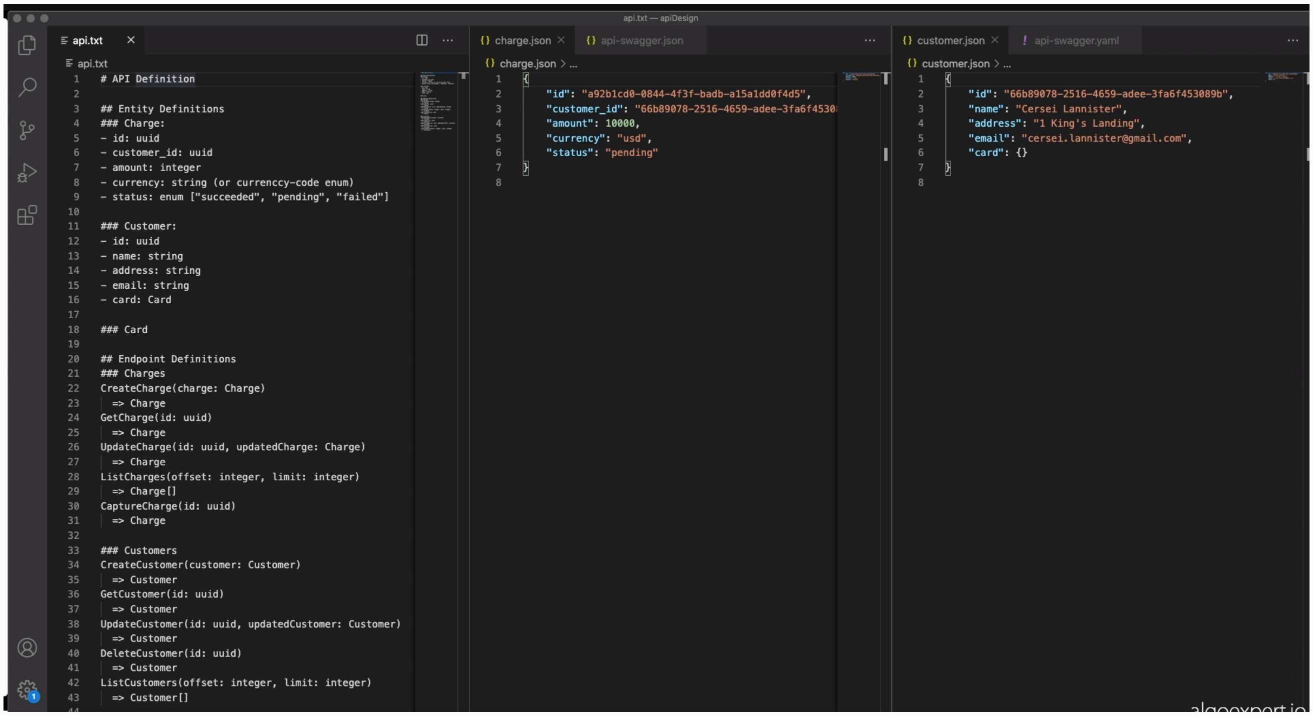


api.txt — apiDesign

api.txt

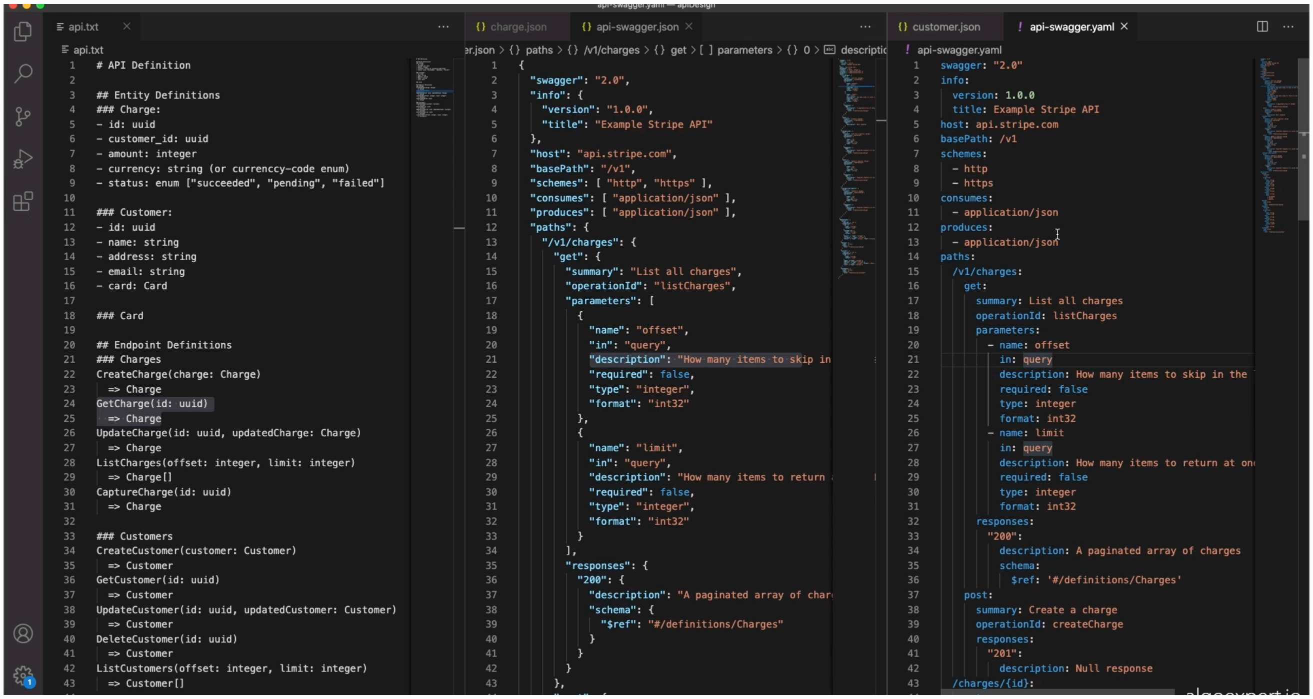
```
# API Definition

## Entity Definitions
### Charge:
- id: uuid
- customer_id: uuid
- amount: integer
- currency: string (or currenccy-code enum)
- status: enum ["succeeded", "pending", "failed"]

### Customer:
- id: uuid
- name: string
- address: string
- email: string
- card: Card

### Card

## Endpoint Definitions
### Charges
CreateCharge(charge: Charge)
  => Charge
GetCharge(id: uuid)
  => Charge
UpdateCharge(id: uuid, updatedCharge: Charge)
  => Charge
ListCharges(offset: integer, limit: integer)
  => Charge[]
CaptureCharge(id: uuid)
  => Charge

### Customers
CreateCustomer(customer: Customer)
  => Customer
GetCustomer(id: uuid)
  => Customer
UpdateCustomer(id: uuid, updatedCustomer: Customer)
  => Customer
DeleteCustomer(id: uuid)
  => Customer
ListCustomers(offset: integer, limit: integer)
  => Customer[]
```

charge.json > ...

```
{
    "id": "a92b1cd0-0844-4f3f-badb-a15a1dd0f4d5",
    "customer_id": "66b89078-2516-4659-adee-3fa6f4530
    "amount": 10000,
    "currency": "usd",
    "status": "pending"
}
```

customer.json > ...

```
{
    "id": "66b89078-2516-4659-adee-3fa6f453089b",
    "name": "Cersei Lannister",
    "address": "1 King's Landing",
    "email": "cersei.lannister@gmail.com",
    "card": {}
}
```



api-swagger.yaml — apiDesign

api.txt

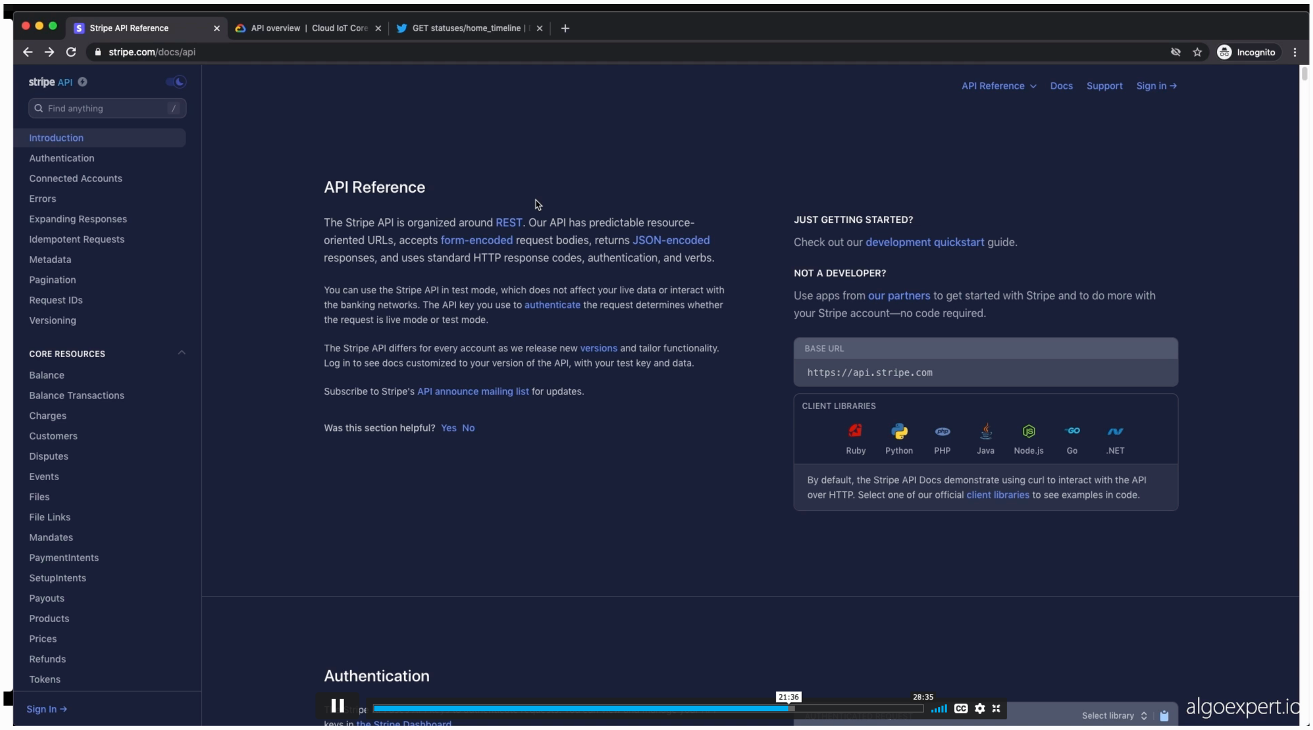
```
# API Definition

## Entity Definitions
### Charge:
- id: uuid
- customer_id: uuid
- amount: integer
- currency: string (or currenccy-code enum)
- status: enum ["succeeded", "pending", "failed"]

### Customer:
- id: uuid
- name: string
- address: string
- email: string
- card: Card

### Card

## Endpoint Definitions
### Charges
CreateCharge(charge: Charge)
  => Charge
GetCharge(id: uuid)
  => Charge
UpdateCharge(id: uuid, updatedCharge: Charge)
  => Charge
ListCharges(offset: integer, limit: integer)
  => Charge[]
CaptureCharge(id: uuid)
  => Charge

### Customers
CreateCustomer(customer: Customer)
  => Customer
GetCustomer(id: uuid)
  => Customer
UpdateCustomer(id: uuid, updatedCustomer: Customer)
  => Customer
DeleteCustomer(id: uuid)
  => Customer
ListCustomers(offset: integer, limit: integer)
  => Customer[]
```

er.json > {} paths > {} /v1/charges > {} get > [ ] parameters > {} 0 > descriptic

```
{
  "swagger": "2.0",
  "info": {
    "version": "1.0.0",
    "title": "Example Stripe API"
  },
  "host": "api.stripe.com",
  "basePath": "/v1",
  "schemes": [ "http", "https" ],
  "consumes": [ "application/json" ],
  "produces": [ "application/json" ],
  "paths": {
    "/v1/charges": {
      "get": {
        "summary": "List all charges",
        "operationId": "listCharges",
        "parameters": [
          {
            "name": "offset",
            "in": "query",
            "description": "How many items to skip in
            "required": false,
            "type": "integer",
            "format": "int32"
          },
          {
            "name": "limit",
            "in": "query",
            "description": "How many items to return
            "required": false,
            "type": "integer",
            "format": "int32"
          }
        ],
        "responses": {
          "200": {
            "description": "A paginated array of char
            "schema": {
              "$ref": "#/definitions/Charges"
            }
          }
        }
      },
```

api-swagger.yaml

```
swagger: "2.0"
info:
  version: 1.0.0
  title: Example Stripe API
host: api.stripe.com
basePath: /v1
schemes:
  - http
  - https
consumes:
  - application/json
produces:
  - application/json
paths:
  /v1/charges:
    get:
      summary: List all charges
      operationId: listCharges
      parameters:
        - name: offset
          in: query
          description: How many items to skip in the
          required: false
          type: integer
          format: int32
        - name: limit
          in: query
          description: How many items to return at on
          required: false
          type: integer
          format: int32
      responses:
        "200":
          description: A paginated array of charges
          schema:
            $ref: '#/definitions/Charges'
    post:
      summary: Create a charge
      operationId: createCharge
      responses:
        "201":
          description: Null response
  /charges/{id}:
```



API Reference

The Stripe API is organized around REST. Our API has predictable resource-oriented URLs, accepts form-encoded request bodies, returns JSON-encoded responses, and uses standard HTTP response codes, authentication, and verbs.

You can use the Stripe API in test mode, which does not affect your live data or interact with the banking networks. The API key you use to authenticate the request determines whether the request is live mode or test mode.

The Stripe API differs for every account as we release new versions and tailor functionality. Log in to see docs customized to your version of the API, with your test key and data.

Subscribe to Stripe's API announce mailing list for updates.

Was this section helpful? Yes No

JUST GETTING STARTED?

Check out our development quickstart guide.

NOT A DEVELOPER?

Use apps from our partners to get started with Stripe and to do more with your Stripe account—no code required.

BASE URL

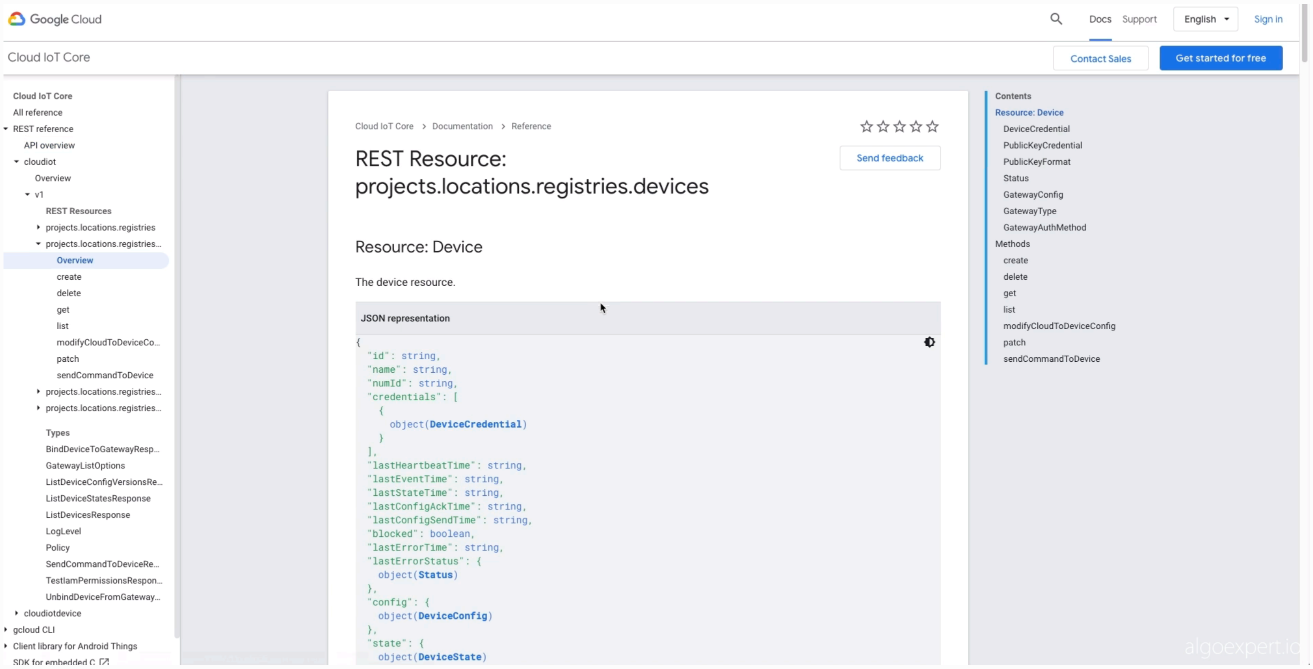
```
https://api.stripe.com
```

CLIENT LIBRARIES

By default, the Stripe API Docs demonstrate using curl to interact with the API over HTTP. Select one of our official client libraries to see examples in code.

Authentication

keys in the Stripe Dashboard



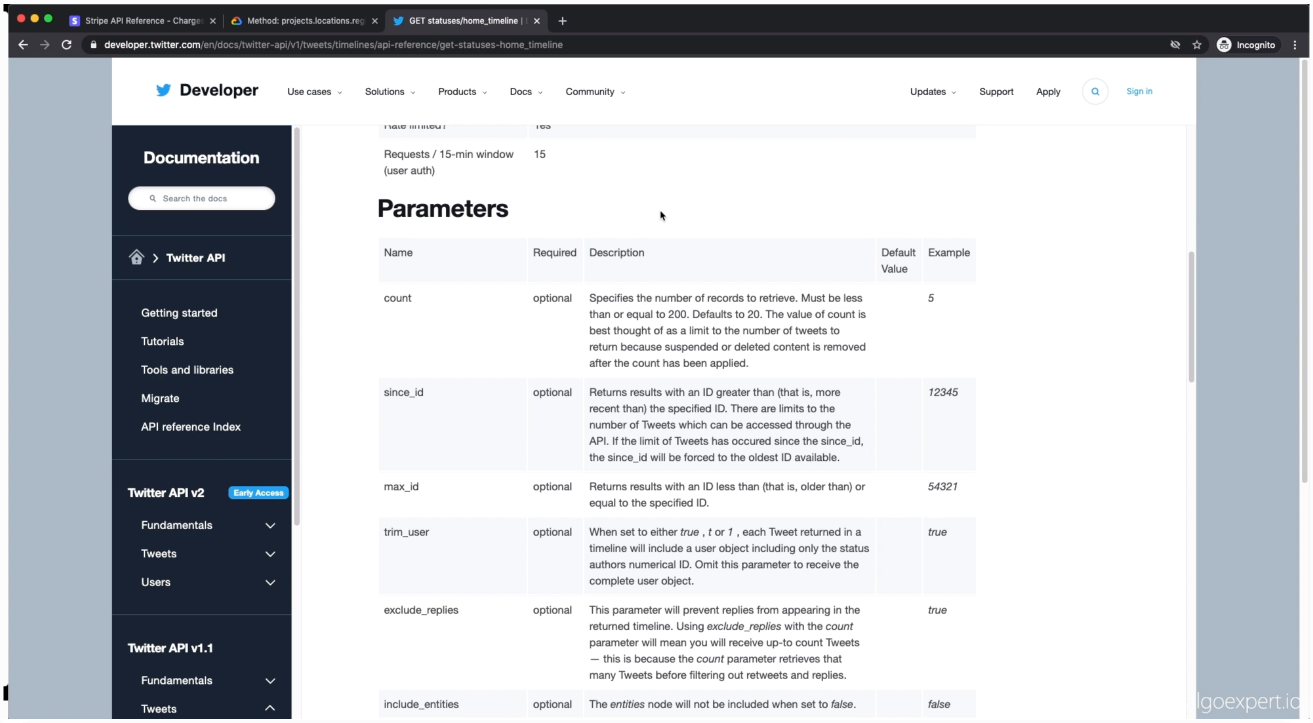
REST Resource: projects.locations.registries.devices

Resource: Device

The device resource.

JSON representation

```
{
  "id": string,
  "name": string,
  "numId": string,
  "credentials": [
    {
      object(DeviceCredential)
    }
  ],
  "lastHeartbeatTime": string,
  "lastEventTime": string,
  "lastStateTime": string,
  "lastConfigAckTime": string,
  "lastConfigSendTime": string,
  "blocked": boolean,
  "lastErrorTime": string,
  "lastErrorStatus": {
    object(Status)
  },
  "config": {
    object(DeviceConfig)
  },
  "state": {
    object(DeviceState)
```



Requests / 15-min window (user auth)	15

Parameters

Name	Required	Description	Default Value	Example
count	optional	Specifies the number of records to retrieve. Must be less than or equal to 200. Defaults to 20. The value of count is best thought of as a limit to the number of tweets to return because suspended or deleted content is removed after the count has been applied.		*5*
since_id	optional	Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.		*12345*
max_id	optional	Returns results with an ID less than (that is, older than) or equal to the specified ID.		*54321*
trim_user	optional	When set to either *true* , *t* or *1* , each Tweet returned in a timeline will include a user object including only the status authors numerical ID. Omit this parameter to receive the complete user object.		*true*
exclude_replies	optional	This parameter will prevent replies from appearing in the returned timeline. Using *exclude_replies* with the *count* parameter will mean you will receive up-to count Tweets — this is because the *count* parameter retrieves that many Tweets before filtering out retweets and replies.		*true*
include_entities	optional	The *entities* node will not be included when set to *false*.		*false*

4 Prerequisites

HTTP

The **H**yper**T**ext **T**ransfer **P**rotocol is a very common network protocol implemented on top of TCP. Clients make HTTP requests, and servers respond with a response.

Requests typically have the following schema:

```
host: string (example: algoexpert.io)
port: integer (example: 80 or 443)
method: string (example: GET, PUT, POST, DELETE, OPTIONS or PATCH)
headers:  pair list (example: "Content-Type" => "application/json")
body: opaque sequence of bytes
```

Responses typically have the following schema:

```
status code: integer (example: 200, 401)
headers:  pair list (example: "Content-Length" => 1238)
body: opaque sequence of bytes
```

JSON

A file format heavily used in APIs and configuration. Stands for **J**ava**S**cript **O**bject **N**otation. Example:

```
{
   "version": 1.0,
   "name": "AlgoExpert Configuration"
}
```

YAML

A file format mostly used in configuration. Example:

```
version: 1.0
name: AlgoExpert Configuration
```

ACL

Short for **Access-Control List**. This term is often used to refer to a permissioning model: which users in a system can perform which operations. For instance, APIs often come with ACLs defining which users can delete, edit, or view certain entities.

2 Key Terms

Pagination

When a network request potentially warrants a really large response, the relevant API might be designed to return only a single **page** of that response (i.e., a limited portion of the response), accompanied by an identifier or token for the client to request the next page if desired.

Pagination is often used when designing **List** endpoints. For instance, an endpoint to list videos on the YouTube Trending page could return a huge list of videos. This wouldn't perform very well on mobile devices due to the lower network speeds and simply wouldn't be optimal, since most users will only ever scroll through the first ten or twenty videos. So, the API could be designed to respond with only the first few videos of that list; in this case, we would say that the API response is **paginated**.

CRUD Operations

Stands for **Create**, **Read**, **Update**, **Delete** Operations. These four operations often serve as the bedrock of a functioning system and therefore find themselves at the core of many APIs. The term **CRUD** is very likely to come up during an API-design interview.