HTTP + REST + Firebase

Some of examples and definitions are from Firebase docs - https://firebase.google.com/docs, or great web docs https://developer.mozilla.org/ or Wikipedia.

1.
What is HTTP?

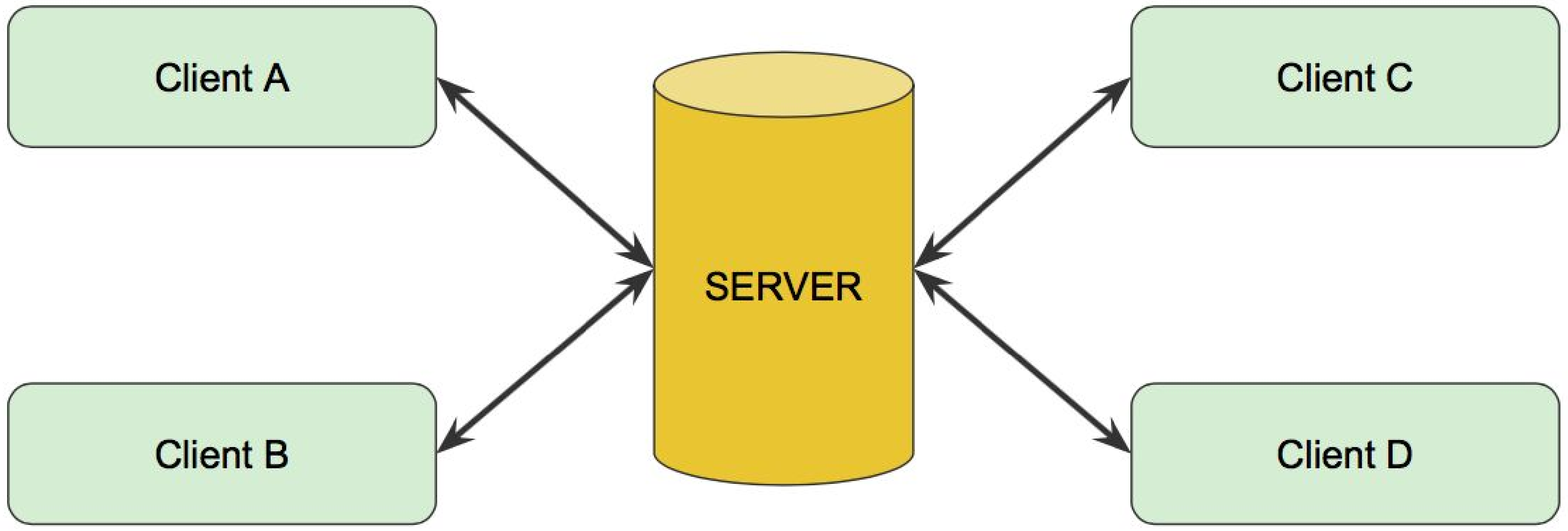
What is HTTP?
Hypertext Transfer Protocol

HTTP is an protocol that is a high-level interface that we use to make request over the Internet.
Its definition presumes an underlying and reliable transport layer protocol - Transmission Control Protocol (TCP) and physical infrastructure..

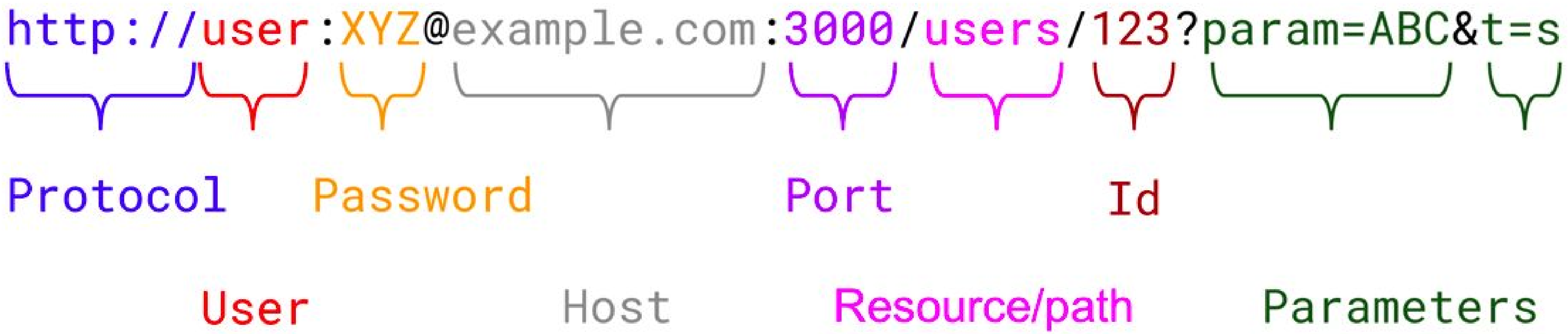
HTTP is:

- text
- stateless
- slient-server architecture

What is HTTP?
Client-server architecture



What is HTTP?
URL



What is HTTP?

Parts of URL

Required:
a. Protocol (**http://**)
b. Host (**api.com**)
c. Resource (**/users**)
d. Object / id (**/users/123**)

Optional:
a. User and password (http://**user:pass**@api.com/)
b. Parameters (/users**?s=word&limit=100**)
c. Version (/**v2**/users)
d. Port (http://localhost:**3000)**

What is HTTP?
HTTP request

The client and server communicate by **plain-text** messages.

The **client sends requests** to the server and the **server sends response.**

Request contains:

1. A request line (e.g. GET users/123 HTTP/1.1)
2. Request header fields
3. An empty line
4. An optional message body

What is HTTP?
HTTP status codes

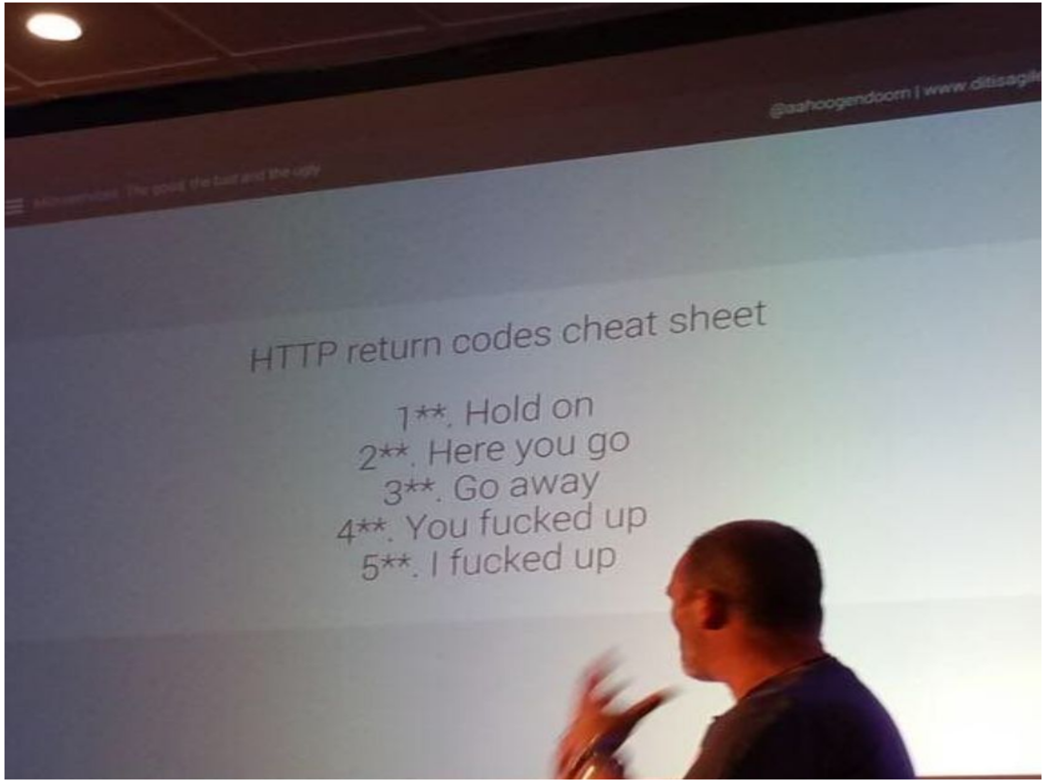
Status codes are issued by a server in response to a browser's request made to the server.

1xx - informational
2xx - success
3xx - redirections
4xx - client error
5xx - server error

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

What is HTTP?

HTTP status codes cheat sheet



Microservices. The good, the bad and the ugly

@aahoogendoorn | www.ditisagile

HTTP return codes cheat sheet

1**. Hold on
2**. Here you go
3**. Go away
4**. You fucked up
5**. I fucked up

What is HTTP?
Every enterprise-grade software has cats inside ;)

https://http.cat/

https://httpstatusdogs.com/

What is HTTP?
Headers

HTTP header fields are components of the header section of HTTP request and response.

They define the operating parameters of an HTTP transaction.

Headers can be checked here -> https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

What is HTTP?
Message body

HTTP Message Body is the data bytes transmitted in an HTTP transaction message immediately following the headers.

What is HTTP?
Sample HTTP response

```
HTTP/1.1 200 OK
Date: Sun, 10 Oct 2010 23:26:07 GMT
Server: Apache/2.2.8 (Ubuntu) mod_ssl/2.2.8 OpenSSL/0.9.8g
Last-Modified: Sun, 26 Sep 2010 22:04:35 GMT
Accept-Ranges: bytes
Content-Length: 13
Connection: close
Content-Type: text/html

Hello world!
```

2.
REST API

REST API
API - application programming interface

In general terms, it is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.

REST API
REST - Representational State Transfer

A way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.

REST has:

- Unified interface
- Stateless communication
- Client-server architecture
- Hipermedia
- HATEOAS

“ Task 0

Find an install Chrome plugin called JSON Viewer

REST API

Checking weather for Lublin through REST API

HOST: http://api.openweathermap.org

RESOURCE: forecast

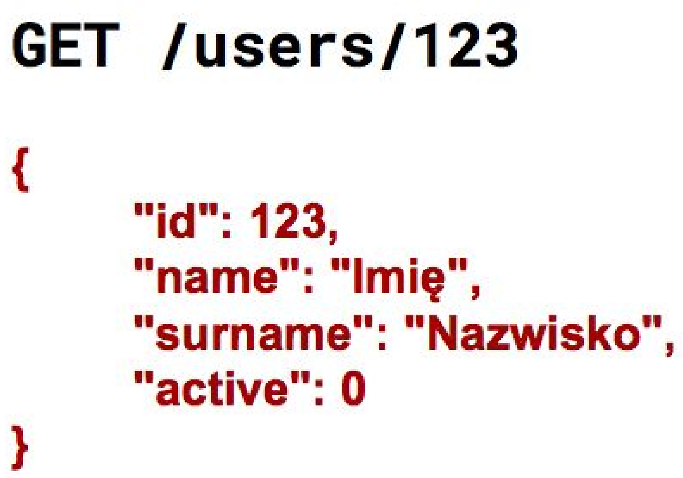
VERSION: 2.5

CITY ID: 765876

API Key: 0b3d75e5a49f2a267f054a0a60bed6f3

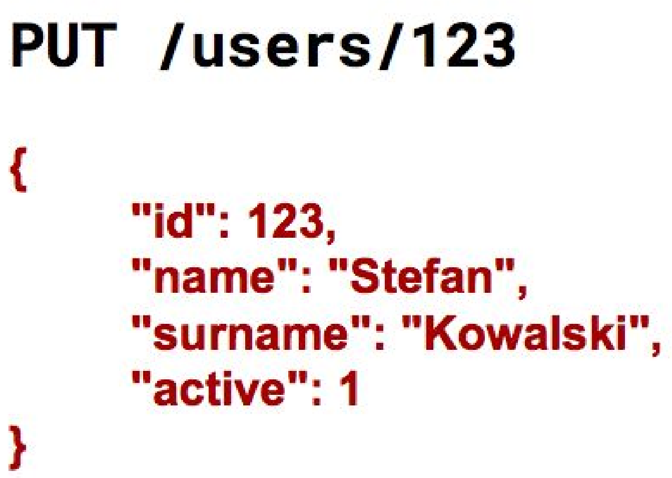
http://api.openweathermap.org/data/2.5/forecast?id=765876&APPID=0b3d75e5a49f2a267f054a0a60bed6f3

REST API
Resources representation



```
GET /users/123

{
        "id": 123,
        "name": "Imię",
        "surname": "Nazwisko",
        "active": 0
}
```



```
PUT /users/123

{
        "id": 123,
        "name": "Stefan",
        "surname": "Kowalski",
        "active": 1
}
```

REST API
Hypermedia As The Engine Of Application State

```
GET /users/123

{
        "id": 123,
        "name": "Imię",
        "surname": "Nazwisko",
        "active": 0,
        "links": {
           "self": "http://example.com/users/123",
           "parent": "http://example.com/users"
         },
}
```

REST API
JSON - JavaScript Object Notation

JSON is an standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types. It is a very common data format used for asynchronous browser–server communication.

JSON data is JS object literal, with keys surrounded by double quotes and it allows data types:

- number
- string
- boolean
- array
- object
- null

REST API
JSON - JavaScript Object Notation

```
{
    "account": -1221.43,
    "name": "Imię",
    "data": [1, 2, 3],
    "parent": { ... },
    "active": false,
    "emptyValue": null
}
```

“ *Task 1*

Find an install Chrome plugin called Advanced REST Client

Task 2

Using http://openweathermap.org/ check actual temperature in Gdańsk in Celsius degrees

0b3d75e5a49f2a267f054a0a60bed6f3

REST API
Example API

http://omdbapi.com/

https://randomuser.me/

Task 3

Using GET check what is under the endpoint https://ad-snadbox.firebaseio.com/get-endpoint.json

REST API
Basic methods

GET - geting resources
POST - creating new resources
PUT - updating whole resource
PATCH - updating part of resource
DELETE - delete resource

REST API

Basic methods on Firebase REST API example

https://firebase.google.com/docs/reference/rest/database/

REST API
Basic methods

1. **GET** /users/YOUR_NAME.json
2. **POST** /users/YOUR_NAME.json
```
{ "name": "Mateusz", "surname": "Choma", "phone": "+48" }
```
3. **PUT** /users/YOUR_NAME/**_ID_**.json
```
{ "name": "Mateusz", "surname": "Choma", "active": 0,
"phone": "+48" }
```
4. **PATCH** /users/YOUR_NAME/**_ID_**.json
```
{ "active": 1 }
```
5. **DELETE** /users/YOUR_NAME/**_ID_**.json

“ ***Task 4***

Try to use methods from previous side on this API https://ad-snadbox.firebaseio.com/

Task 5

Make all this requests using FETCH in REACT.

** triggered by button clicks*
*** editable body in textarea*

Task 6 BEER CONTEST!

“

This API sends mails - https://emaillabs-sender.herokuapp.com/
Use POST to send mail to chomamateusz@gmail.com

Fastest WIN!

Task 6 BEER CONTEST!

“

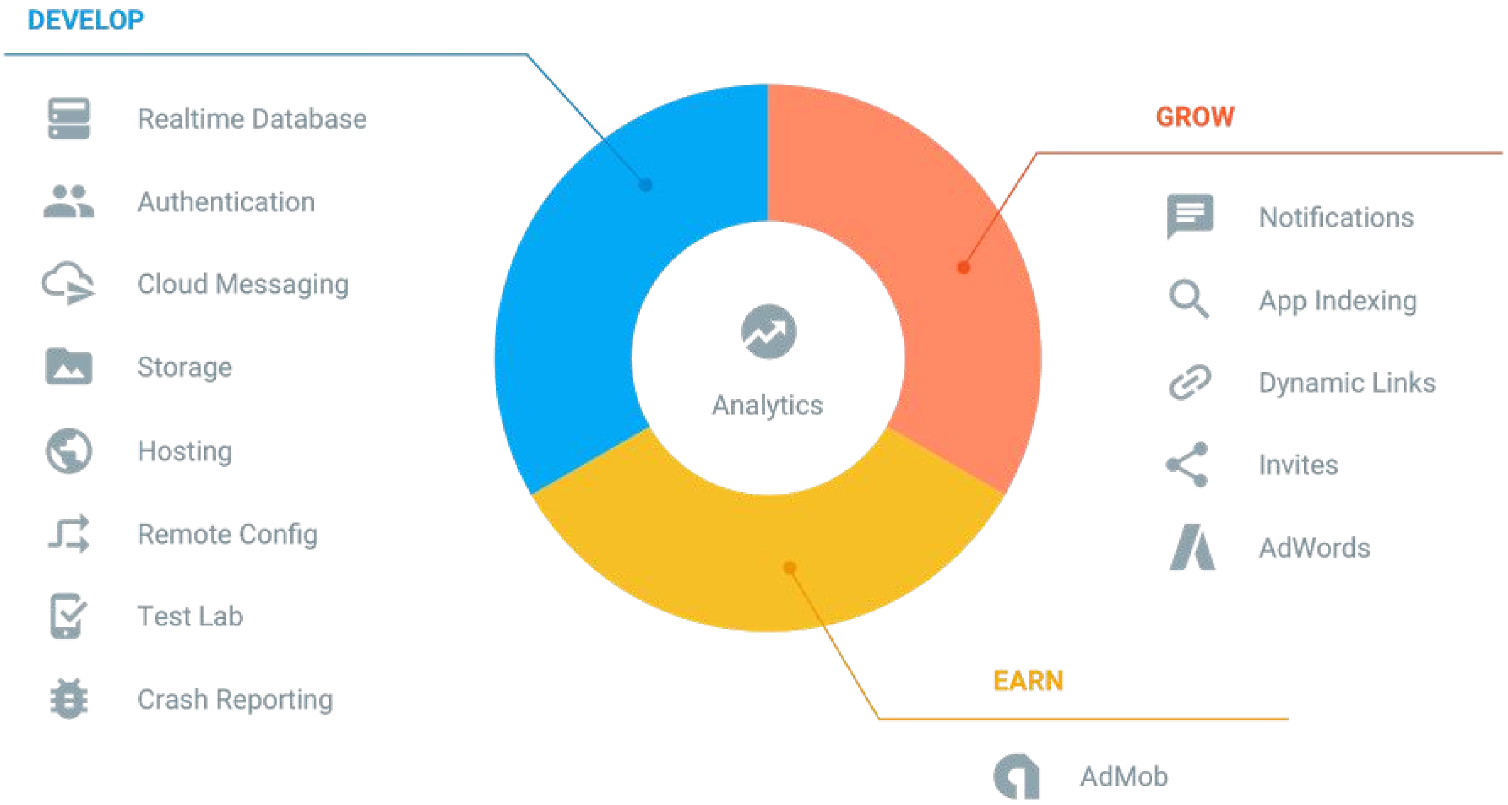
API require POST body to be a JSON object that has properties:
emails - array of to emails [strings]
message - message [string]
subject - subject [string]
from - email of sender [string]
fromName - name of sender [string]

3.
What is Firebase? Serverless architecture.

What is Firebase? Serverless architecture.
Basic info

- Firebase is a platform for developing apps
- Firebase should be able to replace backend servers
- Their have libs to handle Firebase from web, Android and iOS apps.
- Firebase, Inc. was founded in 2011
- In 2014 was acquired by Google
- In 2015 Google calls Firebase “United app platform” on Google I/O conference

What is Firebase? Serverless architecture.
Unified app platform



4.
Getting started

“ *Task 7*

Make a personal and team project in Google Firebase

Getting started
Console and features

EXPLORING FIREBASE CONSOLE & FEATURES

Getting started
Firebase docs

EXPLORING FIREBASE DOCS - REFERENCE AND GUIDES

Getting started
Firebase CLI

The Firebase CLI (GitHub) provides a variety of tools for managing, viewing, and deploying to Firebase projects.

```
npm install -g firebase-tools
firebase login
```

Getting started
Firebase Init

```
firebase init
```

The firebase init command does not create a new directory. If you're starting a new project from scratch, you should first make a directory and change directories into it before running the init command.

Getting started
Firebase Hosting

- Free hosting with limited transfer
- Free to conect own domain
- Free and auto SSL certificate
- Deploy from CLI
- Roll-back option from console

Getting started
Firebase Hosting

Testing project locally

```
firebase serve
```

Deploying project

```
firebase deploy
```

“ *Task 8*

Deploy your team landing page on your personal Firebase project.

5.
Realtime database

Realtime database
noSQL databases

A **NoSQL** (originally referring to "non SQL" or "non relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. - Wikipedia

Realtime database
noSQL databases

The **Firebase Realtime Database** is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.
We can access that database by REST endpoints also.

Realtime database
Database Rules

Firebase Realtime Database Rules determine who has read and write access to your database, how your data is structured, and what indexes exist. These rules live on the Firebase servers and are enforced automatically at all times. **Every read and write request will only be completed if your rules allow it.** By default, your rules are set to allow only authenticated users full read and write access to your database.

Realtime database

Database Rules - default

```
// These rules require authentication
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

Realtime database

Database Rules - public

```
// These rules give ANYONE,
// read and write access to your
database
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

Realtime database

Database Rules - users access their data

```
// These rules grant access to a node matching the
authenticated
// user's ID from the Firebase auth token
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

“ *Task 9*

Add public rules to your personal project.

Realtime database
Connecting firebase app to our JS project

We need Firebase library to get all Firebase possibilities like real time database.

In simple website we can use CDN:

```
<script
src="https://www.gstatic.com/firebasejs/4.5.0/firebase.js "></script>
```

In React app we can install it from npm:

```
npm install --save firebase
```

Realtime database
Connecting firebase app to our JS project

```
// Set the configuration for your app
var config = {
    apiKey: "apiKey",
    authDomain: "projectId.firebaseapp.com",
    databaseURL: "https://databaseName.firebaseio.com",
    storageBucket: "bucket.appspot.com"
};
firebase.initializeApp(config);
```

Realtime database
Connecting firebase app to our JS project

We can add and initialise Firebase wherever we want, but the best place for it is separate file firebase.js in ours /src folder. In that file we can import firebase library initialize the app and export the initialized app.

Realtime database
Getting into Firebase Database

As Firebase offers us many different services we must firstly choose what service we want to use.

If we want to use realtime database we can call:

```
var db = firebase.database();
```

Task 10

Connect firebase to your react app in separate .js file. Export form in firebase.database()

Realtime database

How we store data - some rules

1. Data in Realitme Database is a JSON tree
2. It allows nesting data up to 32 levels deep but nesting isn't recommended. When you fetch data at a location in your database, you also retrieve ALL of its child nodes.
3. Flat data structures (denormalization). If the data is split into parts it can be efficiently downloaded in separate calls, as it is needed.

Realtime database
Getting the path to write

We can reference any path in our database (even that doesn't exists) by calling ref on database reference.

```
var dbRef = firebase.database().ref();

var dbRef =
firebase.database().ref(‘my/firts/path’);
```

Realtime database

Simply writing the data

For basic write operations, you can use **set()** to save data to a specified reference, **replacing any existing data at that path**. For example:

```
firebase.database().ref(cats/).set(“Fluffy”);

firebase.database().ref(cats/).set({name: “Fluffy”});
```

Task 11

Write “it works” to “my/first/path” path in your personal Firebase project.

“ *Task 12*

Write an object with your personal data (name, surname, age) to “myData” path in your personal Firebase project.

“ *Task 13*

Write an array with integers (for example: [1,2,3]) to ”array” path in your personal Firebase project.

Realtime database
Deleting the data

The simplest way to delete data is to call **remove()** on a reference to the location of that data.

You can also delete by specifying **null** as the value for another write operation such as **set()**.

“ *Task 14*

Write the code that deletes all data that was be written previously to your personal project.

Realtime database
Simply reading the data

In Firebase we can fetch data **once or every time** subscribing to database changes and get notifications on one of these events:

- `child added`
- `child changed`
- `child_removed`
- `value`

Realtime database
Read data once

In some cases you may want a snapshot of your data without listening for changes, such as when initializing a UI element that you don't expect to change. You can use the **once()** method to simplify this scenario: it triggers once and then does not trigger again.

```
var dbRef = firebase.database().ref();
dbRef.once('value', function(snapshot) {
  let value = snapshot.val();
});
```

Realtime database

Read data on each event

In other cases we want to be informed when something happens in database. We can call **on()** instead of **once()**

```
var dbRef = firebase.database().ref();
dbRef.on('value', function(snapshot) {
  let value = snapshot.val();
});
```

Task 15

Read the data once from your database, and explore what snapshot object is.
Pay attention to forEach method.

Realtime database
What is the realtime thing?

LIVE CODING EXAMPLE

Task 16

Subscribe to value event on some path in your personal project. Display it contents. Update it every time it changes.

Realtime database
Reading and writing lists

When we want to add another child to a location in our database we can call **push()** method on that reference.

```
var dbRef = firebase.database().ref();

// Create a new post reference with an auto-generated id
var newPostRef = dbRef.push();
newPostRef.set({...});

// this line do the same as line above
dbRef.push({...});
```

Task 17

Push some random user values fetched from randomuser.me/api to “userList” path in your personal project.

Realtime database
Sorting and filtering data

We can build simple queries to obtain data sorted by key, by value, or by value of a child.

You can also filter the sorted result to a specific number of results or a range of keys or values.

Realtime database
Sort data

- **orderByChild()** - Order results by the value of a specified child key
- **orderByKey()** - Order results by child keys
- **orderByValue()** - Order results by child values

Most useful is **orderByChild()** method. We can sort list of user objects by age, for example.

How the data is ordered - https://firebase.google.com/docs/database/web/lists-of-data#data-order

Realtime database
Sort data

LIVE CODING EXAMPLE

“ **Task 18**

Order data previously fetched from randomuser.me by firstname.

Realtime database
Filtering data

- **limitToFirst()** - Sets the maximum number of items to return from the beginning of the ordered list of results.
- **limitToLast()**- Sets the maximum number of items to return from the end of the ordered list of results.
- **startAt()** - Return items greater than or equal to the specified key or value, depending on the order-by method chosen.
- **endAt()** - Return items less than or equal to the specified key or value, depending on the order-by method chosen.
- **equalTo()** - Return items equal to the specified key or value, depending on the order-by method chosen.

Task 19

Get the last and first 2 children of previously fetched from randomuser.me user list.

“ ***Task 20***

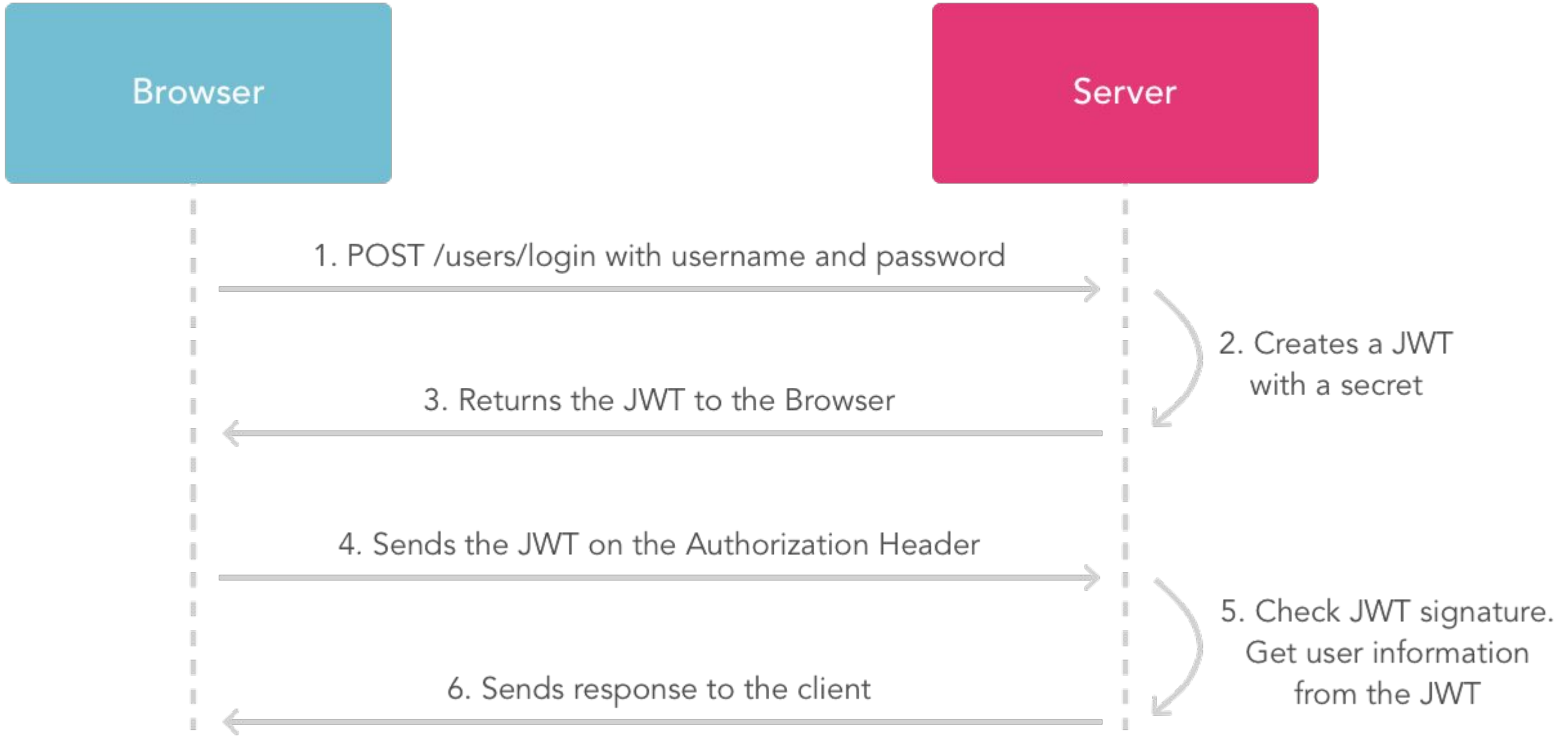
Get the user that have certain email (check in Firebase console to choose one) using the equalTo()

6.
Authentication

Authentication
Possibilities

- Email and password based authentication
- 3rd party - Google, Facebook, Twitter, GitHub
- Phone number (SMS)
- Custom
- Anonymus

Authentication
JWT conception


Browser
Server
1. POST /users/login with username and password
2. Creates a JWT with a secret
3. Returns the JWT to the Browser
4. Sends the JWT on the Authorization Header
5. Check JWT signature. Get user information from the JWT
6. Sends response to the client

Grafika pochodzi z auth0.com

Authentication
Step by step

1. Set up sign-in methods
2. Implement UI flows for your sign-in methods
3. Pass the user's credentials to the Firebase Authentication SDK

Authentication
Firebase auth service

```
var auth = firebase.auth();
```

It’s very similar to database service.

Authentication
User creation

```
firebase.auth()
    .createUserWithEmailAndPassword(email, password)
        .catch(function(error) {
          // Handle Errors here.
          var errorCode = error.code;
          var errorMessage = error.message;
        });
```

“ *Task 21*

Create first e-mail and password user in Firebase console or by code.

Authentication
User sign in

```
firebase.auth()
    .signInWithEmailAndPassword(email, password)
        .catch(function(error) {
          // Handle Errors here.
          var errorCode = error.code;
          var errorMessage = error.message;
        });
```

Authentication

User sign in

```
firebase.auth()
    .signOut()
        .then(function() {
          // Sign-out successful.
        }, function(error) {
          // An error happened.
        });
```

Task 22

Create simple form to log in and out users.

Authentication
Check login/logout state

```
firebase.auth().onAuthStateChanged(function(user) {
  if (user) {
    // User is signed in.
  } else {
   // User is NOT signed in.
  }
});
```

Task 23

Log user in and checkout information provided in user object from onAuthStateChanged() method.

Task 24

Display information if the user is logged in or not.

Make login panel from that.

Authentication
3-rd party login - Google

Before begin!

Enable Google Sign-In in the Firebase console:

1. In the Firebase console, open the Auth section.
2. On the Sign in method tab, enable the Google sign-in method and click Save.

Authentication
3-rd party login - Google

First we must choose provider:

```
var provider = new firebase.auth.GoogleAuthProvider();
```

Code below is identical to all providers! Configuration differs!

```
firebase.auth().signInWithPopup(provider)
    .then(function(result) {
      // This gives you a result
      console.log(result);
    }).catch(function(error) {
      // Handle Errors here.
    });
```

“ Task 25

Log yourself in by Google, check the result object and find and decode auth token (jwt.io).

“ Task 26

Create login form that logs an user and push current date (timestamp) to “log/userLogins/$userId”.

Authentication
Current user

We can access current user from every place we can access firebase object by calling:

```
var user = firebase.auth().currentUser;
```

Authentication
Easy updating profile

You can update a user's basic profile information —the user's display name and profile photo URL—with the **updateProfile()** method. For example:

```
var user = firebase.auth().currentUser;
user.updateProfile({
  displayName: "Jane Q. User",
  photoURL: "https://example.com/jane-q-user/profile.jpg"
}).then(function() {
  // Update successful.
}).catch(function(error) {
  // An error happened.
});
```

Task 27 - put it all together

“

Create an app that:
- displays login box if no user logged
- logs user by Google or email/pass
- displays a txt input that user can pushed under “tasks/$userId/” path
- displays a list of items saved in “tasks/$userId/”

7.
Storage

Storage
Getting started

Cloud Storage for Firebase lets you upload and share user generated content, such as images and video, which allows you to build rich media content into your apps. Your data is stored in a Google Cloud Storage bucket.

Cloud Storage lets you securely upload these files directly from mobile devices and web browsers, handling spotty networks with ease.

Storage

Storage rules

```
// Anyone can read or write to the bucket,
   even non-users of your app.

service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write;
    }
  }
}
```

Storage

Gettnig into Firebase Storage

If we want to use storage service we can call:

```
var storage = firebase.storage();
```

We can create references to certain paths in storage the same way as in database:

```
var storageRef = storage.ref();
```

Storage
Downloading data

```
storageRef.child('images/stars.jpg')
    .getDownloadURL()
    .then(function(url) {
      // `url` is the download URL for
      // 'images/stars.jpg'

      // here we can set this URL
      // as img src attribute (if it is an image)

    });
```

Storage

Simple upload

```
var file = ... // use the file from eg. file input

ref.put(file).then(function(snapshot) {
  console.log('Uploaded a blob or file!');
});
```

Storage
Simple upload

LIVE CODING EXAMPLE

Storage
Simple upload

The put() method return an uploadTask.

```
var uploadTask =
    storageRef.child('images/rivers.jpg').put(file);
```

We can monitor state of upload by callon on() method of uploadTask.

Storage
Upload with progress monitoring

```
uploadTask.on('state_changed', function(snapshot){
    var progress =
        (snapshot.bytesTransferred /
        snapshot.totalBytes) * 100;

    console.log('Upload is ' + progress + '% done');
}, function(error) {
    console.log(‘error’);
}, function() {
        var downloadURL =
            uploadTask.snapshot.downloadURL;
});
```

Storage
Upload with progress monitoring

LIVE CODING EXAMPLE

THANKS!