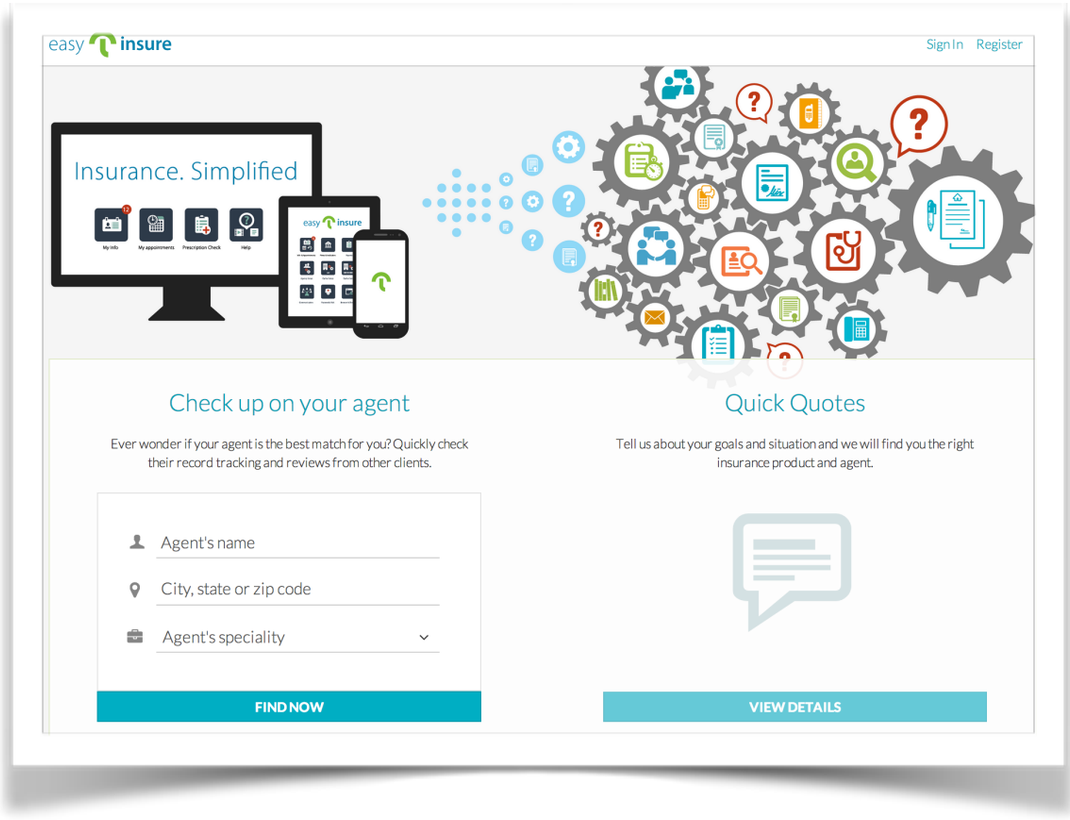
RESTful services and OAUTH protocol in IoT

by Yakov Fain, Farata Systems

Farata Systems and SuranceBay



http://easy.insure

The three parts of this presentation

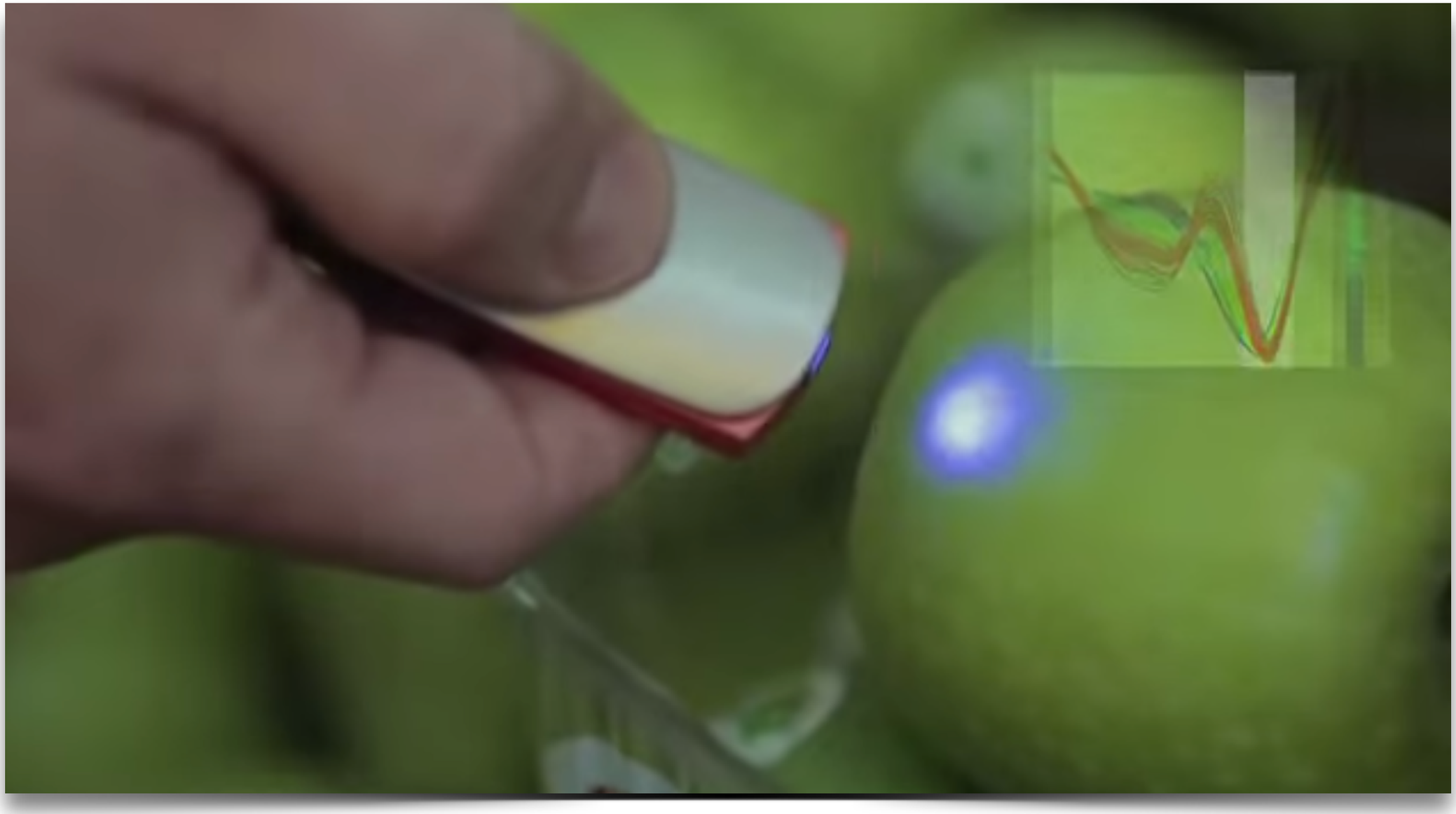
- One approach to integrating consumer devices in the business workflow
- Live demo: integration of a blood pressure monitor
- A brief review of REST, OAUTH, Websockets and their roles tin our application.

Yesterday's Sensors (Things)

- 18 years ago. Telephony.
- I've been programming IoT!

Today's Sensors

SCIO: a molecular sensor that scans physical objects and receives instant information to your smartphone.

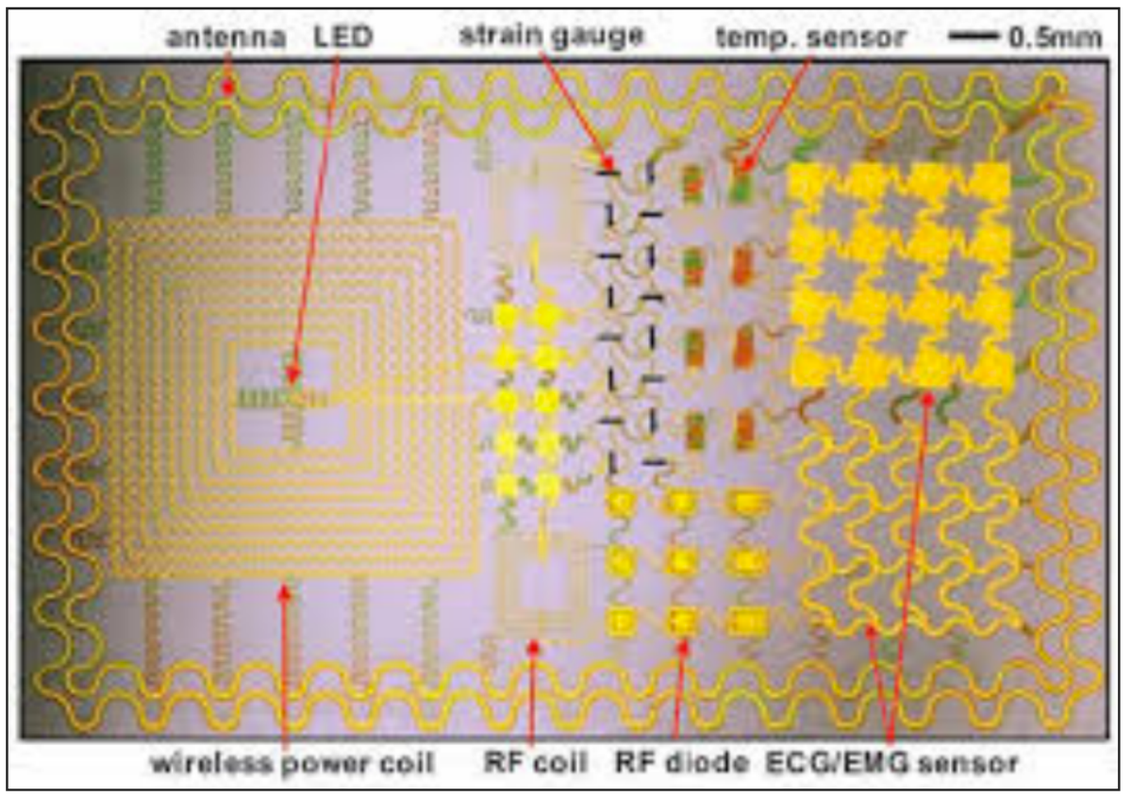


http://www.consumerphysics.com/

Tomorrow: Streachable Wearables

epidermal electronics

Tattoo-like 'electronic skin' wear detects heart attacks, epilepsy, skin dehydration



Here is a tattoo-like thin wearable device that can detect heart attacks, Parkinson's disease or epilepsy attacks, store your body information and deliver medicine to your body, besides collecting patient health, treatment and monitoring at one time.

Researchers in the US have created an 'electronic skin' that can store and transmit data about a person's movements, receive diagnostic information and release drugs into skin, which has been altered considerably to detect heart condition too.

Source: http://bit.ly/1uu0srr

A thing is a thing,

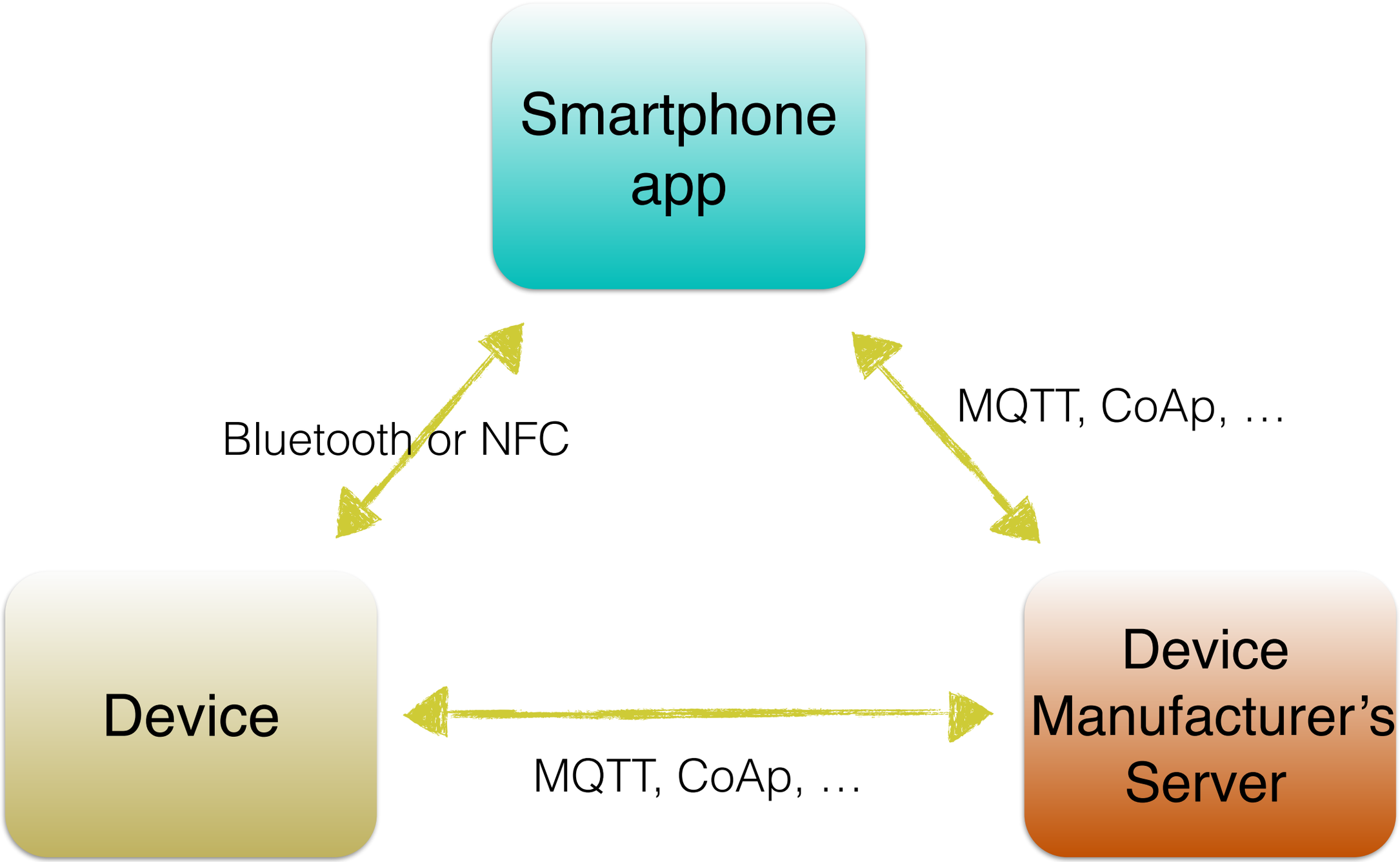
not what is said of that thing.

The Birdman movie

A thing is a thing +

an app + an API + a Web site.

A Typical Consumer Device Setup



Low-Level IoT Approach

Learn and implement IoT protocols: MQTT, XMPP, AMQP, CoAp,…

Write Java programs for Raspberry Pi or Arduino

Learn HomeKit and HealthKit from Apple

High-Level IoT Approach

Create applications using standard technologies to integrate things into an existing business workflow.

A Proof of Concept App

- Integrate consumer devices into one of the **insurance business workflows**
- Leverage existing software technologies
- Create a standard-based application layer that connects things

Your Server in the Middle

- Create a software layer as a proxy for all communications with IoT devices.
- Find the use-cases for data-gathering devices in your business applications.
- Collect valuable data from devices for analisys.

Java dominates on the middleware market.

The Use Case: Integrating Scale and Blood Pressure Monitor into insurance workflow

IHealthLabs Blood Pressure Monitor

Fitbit Scale Aria

Medical Examiner's Report

Legal & General AMERICA

Banner Life Insurance Company
3275 Bennett Creek Avenue
Frederick, Maryland 21704
(800) 638-8428

ICC08 LU-1267 (10/08)

PART 3
Medical Examiner's Report

Name of Proposed Insured _____ Date of Birth _____

Instructions to the Examiner -

This examination, once begun, is the property of the Company, and must not be destroyed or suppressed. Please weigh and measure this applicant. Explain all positive findings under Remarks.

The questions which appear below are intended only as a basis for the examination. The Company relies on its examiners to observe and report all information bearing on the acceptance of a proposed insured, even though not specifically requested on this form.

Please mail blood and urine specimens promptly.

1. Height (in shoes) _____ ft. _____ in.
   Weight (clothed) _____ lbs.

   a. Did you weigh? Yes ☐ No ☐
   b. Did
   If N

   _____

2. Measurer
   Chest (full inspiration) _____ in.
   Chest (forced expiration) _____ in.
   Abdomen (at umbilicus) _____ in.

3. Blood Pressure (record 3 readings)

Systolic			
Diastolic			

minute, etc.)

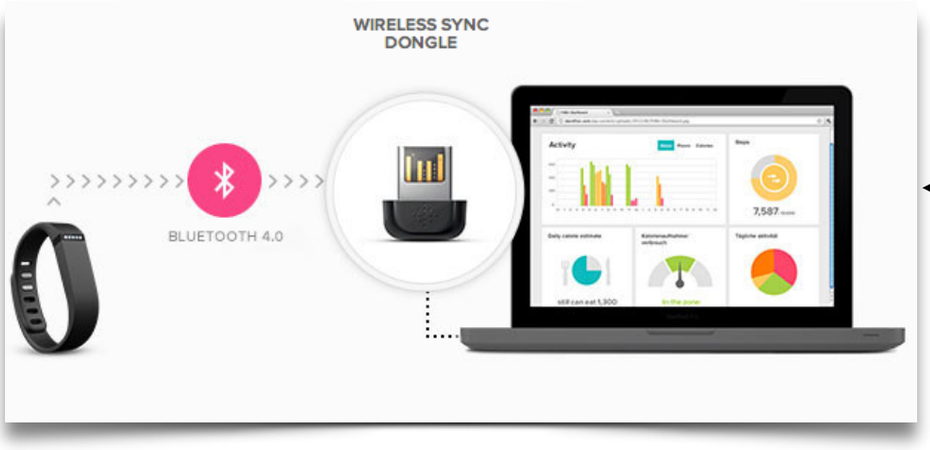
5. Are blood and urine specimens being collected and mailed to the lab? Yes ☐ No ☐

Removing Manual Entry

A Typical IoT Workflow

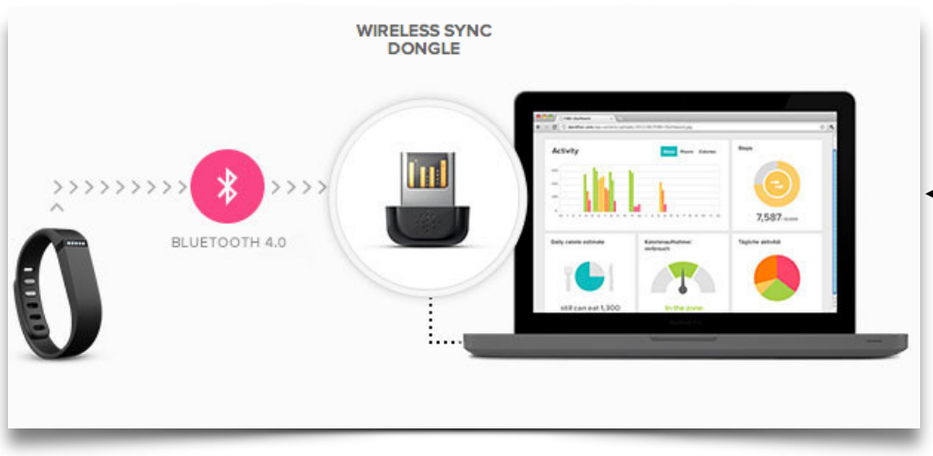
XYZ protocol

DeviceVendor.com



XYZ protocol

A Typical IoT Workflow



XYZ protocol

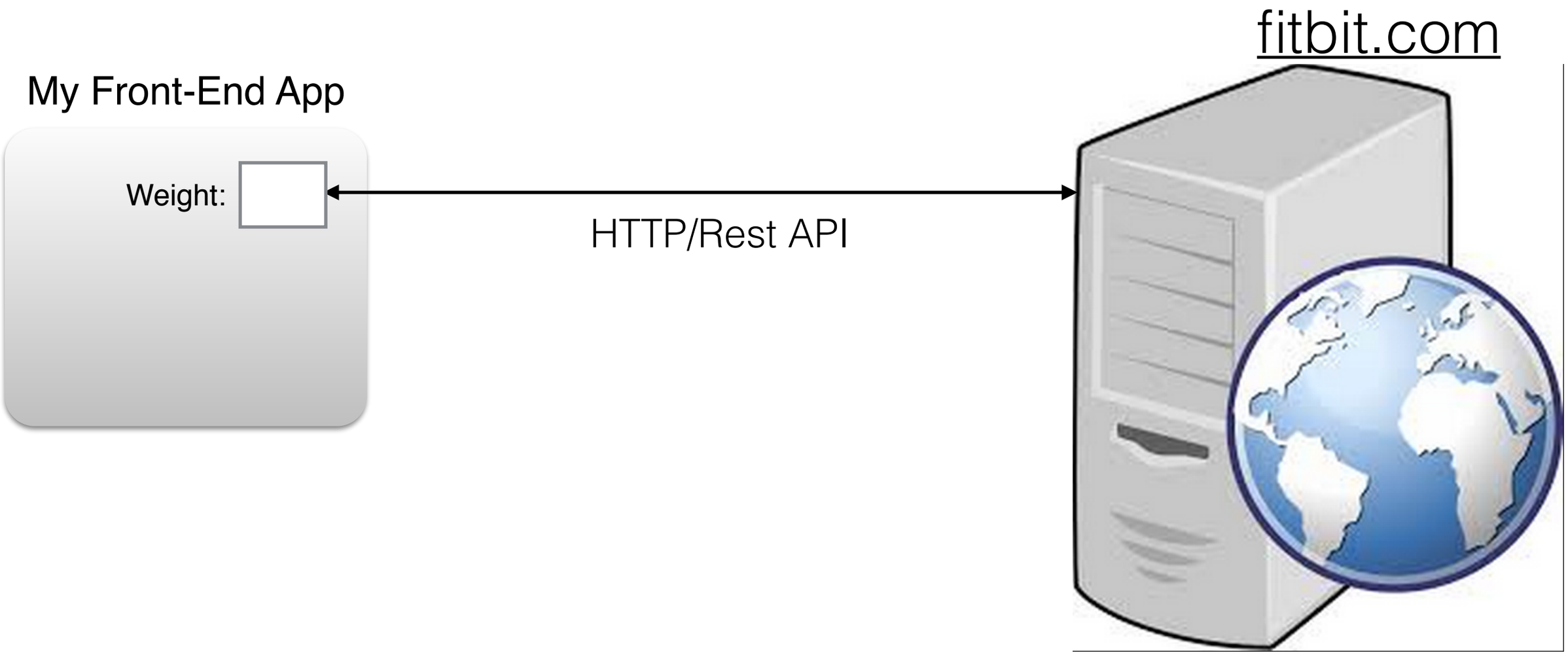
XYZ protocol

DeviceVendor.com

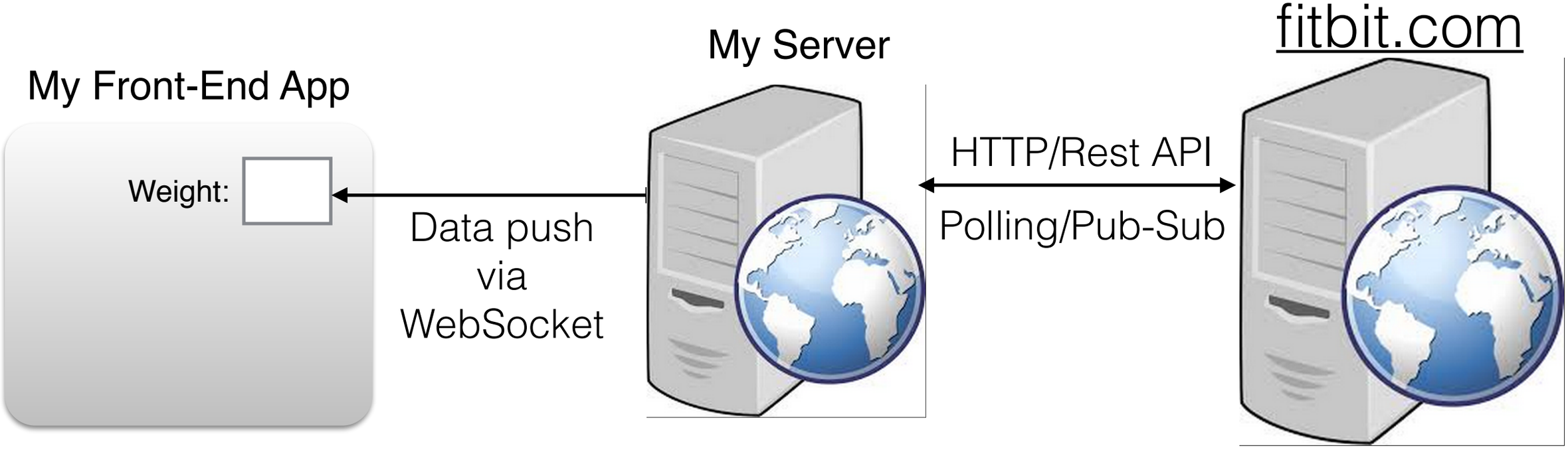
We're not interested in XYZ

Our server communicates with the vendor's server using HTTP

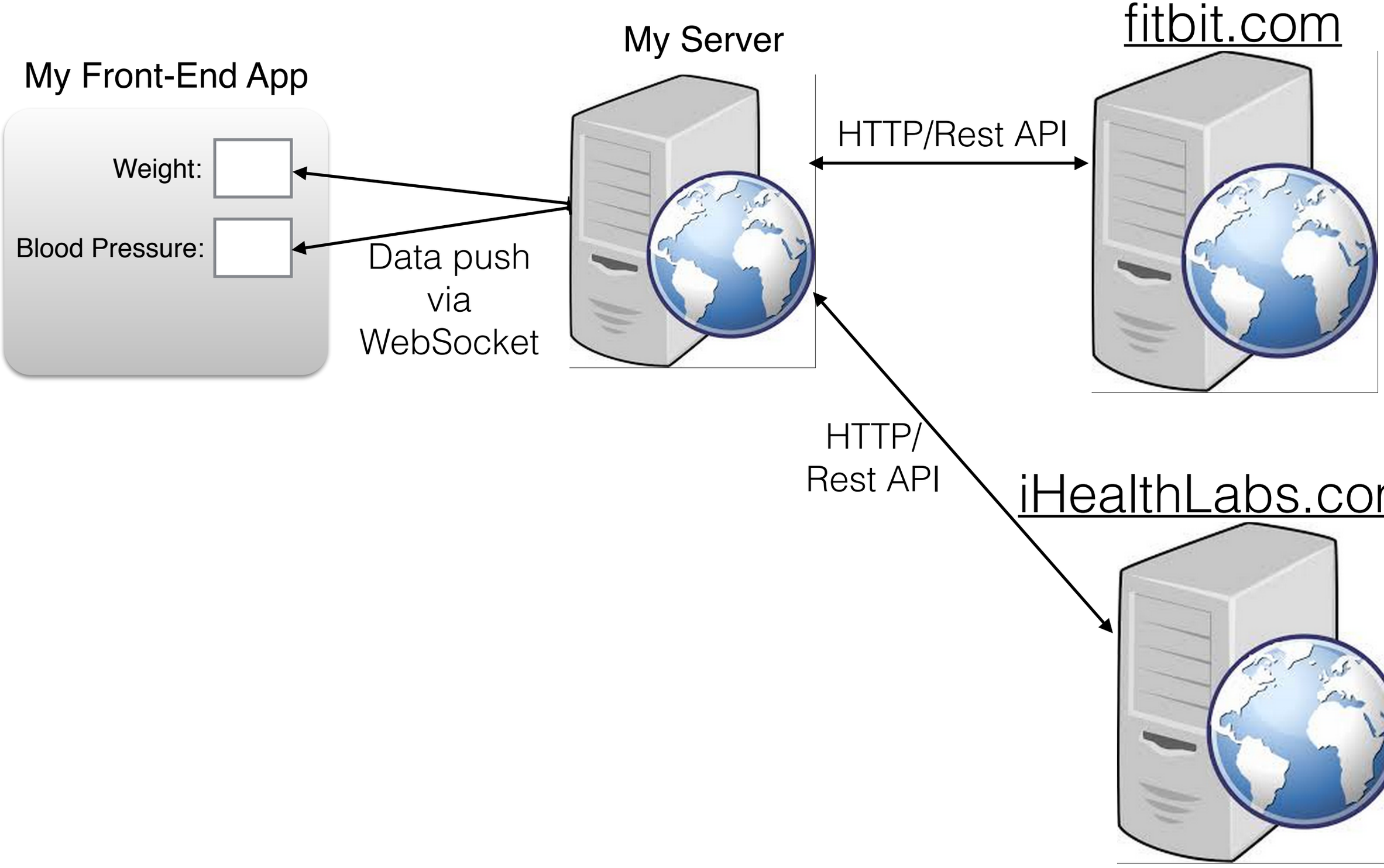
Integrating With Fitbit Scale: Take 1.



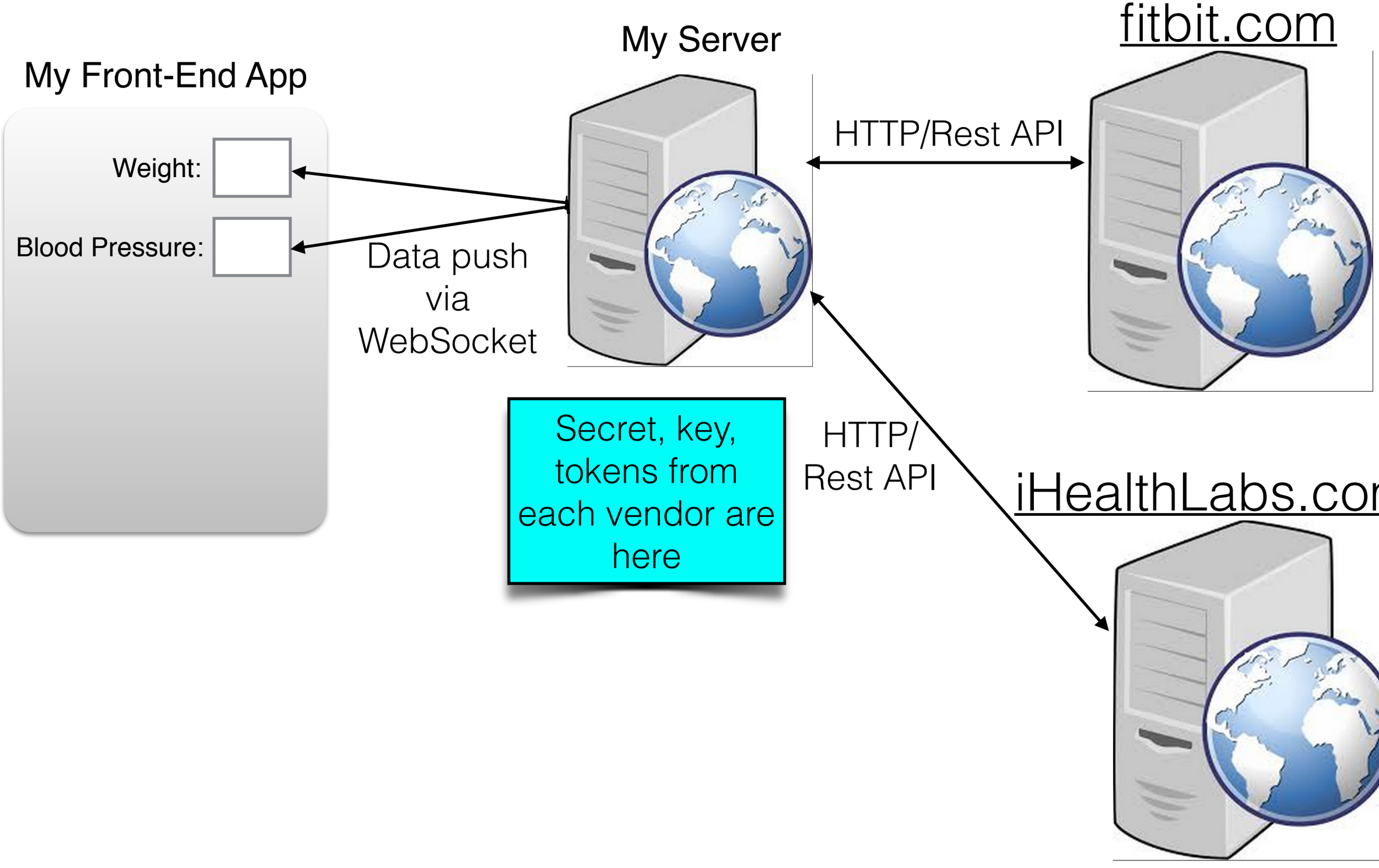
Integrating With Fitbit Scale: Take 2.



Integrating With Fitbit and iHealthLabs.



Adding OAuth Authentication



The Final Architecture

My Front-End App

Weight:

Blood Pressure:

Data push via WebSocket

My Server

HTTP/Rest API

fitbit.com

Secret, key, tokens from each vendor are here

HTTP/ Rest API

iHealthLabs.com

- Vendor's consumer app

Demo

Measuring Blood Pressure

What we've used in our app

- RESTful Web services
- OAuth authentication and authorization
- WebSocket protocol
- Front end is written in Dart, deployed as JavaScript
- JSON data format
- Back-end in written in Java with Spring Boot and embedded Tomcat
- Gradle for build automation

REST API

REpresentational State of Transfer

REST Principles (by Roy Fielding)

- Every resource on the Web has a unique ID (a unique URI)
- Use uniform interface: HTTP **Get, Post, Put, Delete.** Separation of concerns.
- A resource can have multiple representations (text, JSON, XML, PDF, etc.)
- Requests are stateless – no client-specific info is stored between requests
- You can link one resource to another
- Resources should be cacheable
- A REST app can be layered

Selected HTTP Request Methods

- GET Safe, nullipotent, cacheable
- PUT Idempotent
- DELETE Idempotent
- POST None of the above

<- Retrieve

<- Update

<- Delete

<- Create

Nullipotent: a method has no side effect; it doesn't change the data.

Idempotent: regardless of how many times the method is invoked, the end result is the same.

Java EE 7: JAX RS 2.0

- Rest Endpoint - a POJO, typically deployed inside WAR
- Has Client API
- Message Filters and Entity Interceptors (e.g. Login Filter, encryptions et al.)
- Async processing on both client and server
- Validation

Selected JAX-RS Annotations

- @ApplicationPath - defines the URL mapping for the application packaged in a war. It's the base URI for all @Path annotations.
- @Path - a root resource class (POJO),that has at least one method annotated with @Path.
- @PathParam - injects values from request into a method parameter (e.g. Product ID)
- @GET - the class method that handles HTTP Get. You can have multiple methods annotated with @GET, and each produces different MIME type.
- @POST - the class method that handles HTTP Post
- @PUT- the class method that handles HTTP Put
- @DELETE - the class method that handles HTTP Delete
- @Produces - specifies the MIME type for response (e.g. "application/json"). The client's Accept header of the HTTP request declares what's acceptable. The client gets 406 if no methods that produce required is found.
- @Consumes - specifies the MIME types that a resource can consume when sent by the client. If a resource is unable to consume the requested MIME type, the clients get HTTP error 415.
- @QueryParam - if a request URL has parameters, each param will be placed in the provided Java variable.

HTTP Request and Java EE Rest Endpoint

A sample client's HTTP request:
"https://iHealthLabs.com:8443/iotdemo/ihealth/bp"

HTTP Request and Java EE Rest Endpoint

A sample client's HTTP request:
"https://iHealthLabs.com:8443/iotdemo/ihealth/bp"

```
// Configuring The App
@ApplicationPath("iotdemo")
public class MyIoTApplication extends Application {
}
```


HTTP Request and Java EE Rest Endpoint

A sample client's HTTP request:
"https://iHealthLabs.com:8443/iotdemo/ihealth/bp"

```
// Configuring The App
@ApplicationPath("iotdemo")
public class MyIoTApplication extends Application {
}
```

```
// Receiving and handling blood pressure on our server
@Path("/ihealth")
public class BloodPressureService {

  // …
  // The method to handle HTTP Get requests
  @GET
  @Path("/bp")
  @Produces("application/json")
  public String getBloodPressureData() {
    // The code to get bp and prepare JSON goes here
    return bloodPressure;
  }
}
```


FARATA

A Sample Spring's Rest Endpoint

```
// The endpoint handling blood pressure
@RestController
@RequestMapping("/ihealth")
public class HealthLabsController {


// …
// The method to handle HTTP Get requests
@RequestMapping(value="/bp", method = RequestMethod.GET,
                                produces = "application/json")
public Measurement getBloodPressureData() {
    // The code to get blood pressure goes here
    return bloodPressure;
  }
}
```


FARATA

OAuth

Authorizing an app to act on behalf of the user

Authorization and Authentication

- **Authentication** is verifying the identity of the user.
  Is he who he says he is?

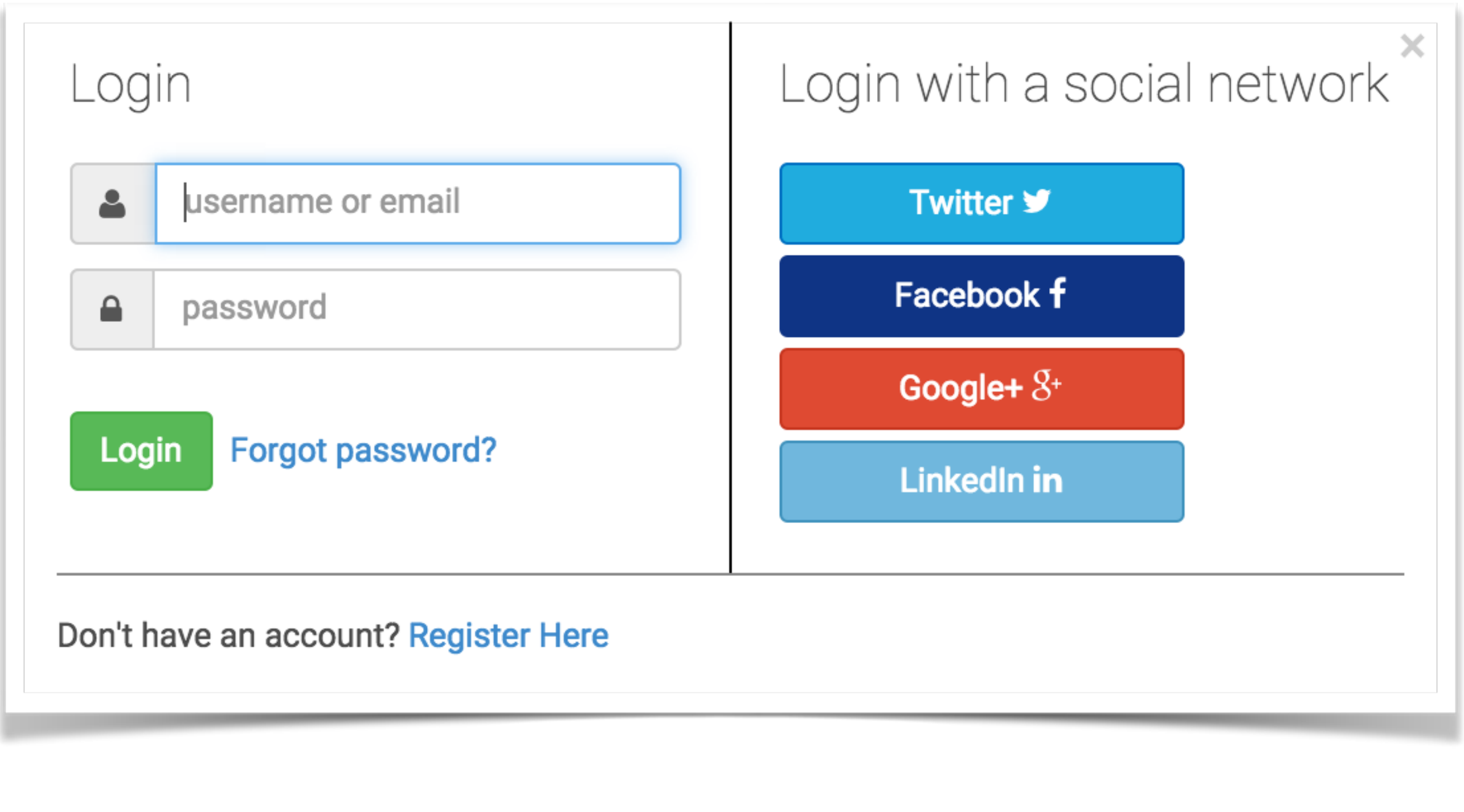
- **Authorization** is figuring out what resources the user can access.

> The owner of the Blood Pressure Monitor can see only the measurments taken from his device.

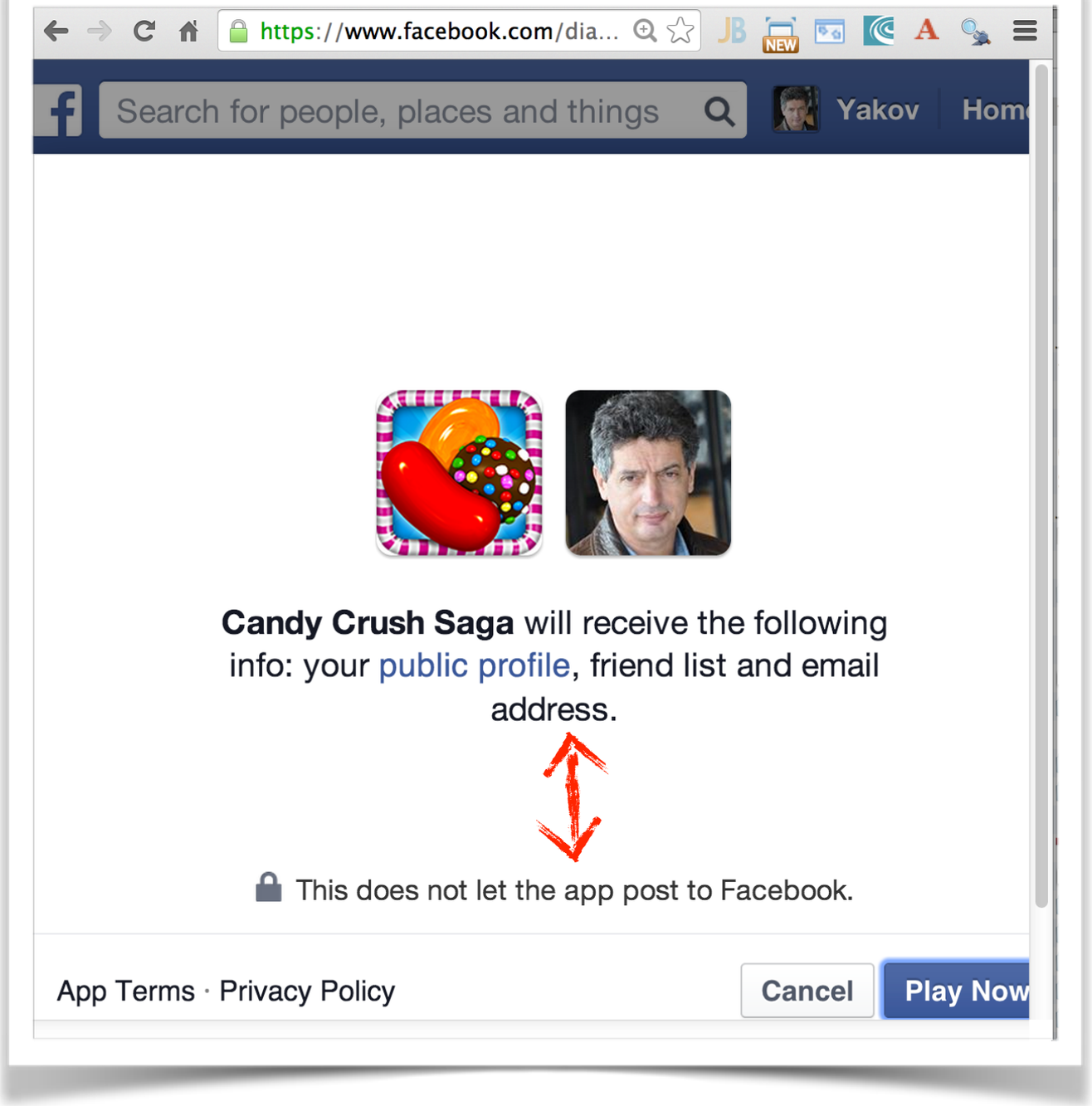
The OAuth Players

- The server with user's resources (data)
- The authorization server
- The client app that wants to acccess user's resources

Delegating to 3rd Party Authorization Servers



Candy Crush Authorization with Facebook



You don’t give your Facebook password to Candy Crush

OAuth 2 Access Token

A client app needs to aquire an access token that can be used on behalf of the user.

A Sample OAuth 2 Workflow

- **My company** registers the app with the thing's vendor providing a **redirect URI** for successful and failed logins and gets a **client id** and a **secret**.

A Sample OAuth 2 Workflow

- My company registers the app with the thing's vendor providing a **redirect URI** for successful and failed logins and gets a **client id** and a **secret**.

- My company builds an app that uses the thing's API (e.g. with REST).

A Sample OAuth 2 Workflow

- My company registers the app with the thing's vendor: providing a **redirect URI** for successful and failed logins and gets a **client id** and a **secret**.

- My company builds an app that uses the thing's API (e.g. with REST).

- The user opens my app and logs into thing's vendor site via its authentication server (not the OAuth provider).

A Sample OAuth 2 Workflow

- **My company** registers the app with the thing's vendor providing a **redirect URI** for successful and failed logins and gets a **client id** and a **secret**.

- **My company** builds an app that uses the thing's API (e.g. with REST)

- **The user** opens **my app** and logs into thing's vendor site via its authentication server (not the OAuth provider).

- My app (not the browser) generates a session-based random state and sends the request to the thing vendor's OAuth provider:

  https://<auth_server>/path?clientid=123&redirect_uri=https//myCallbackURL&response_type=code&scope="email user_likes"&state=7F32G5

A Sample OAuth 2 Workflow

- **My company** registers the app with the thing's vendor providing a **redirect URI** for successful and failed logins and gets a **client id** and a **secret**.

- **My company** builds an app that uses the thing's API (e.g. with REST)

- **The user** opens **my app** and logs into thing's vendor site via its authentication server (not the OAuth provider).

- My app (not the browser) generates a session-based random state and sends the request to the thing vendor's OAuth provider:

  https://<auth_server>/path?clientid=123&redirect_uri=https//myCallbackURL&response_type=code&scope="email user_likes"&state=7F32G5

- My app receives temporary auth code from the thing's vendor, regenerates state and compares with the received one:

  https://myCallbackURL?code=54321&state=7F32G5

A Sample OAuth 2 Workflow

- My company registers the app with the thing's vendor providing a **redirect URI** for successful and failed logins and gets a **client id** and a **secret**.

- My company builds an app that uses the thing's API (e.g. with REST)

- The user opens my app and logs into thing's vendor site via its authentication server (not the OAuth provider).

- My app (not the browser) generates a session-based random state and sends the request to the thing vendor's OAuth provider:

  https://<auth_server>/path?clientid=123&redirect_uri=https//myCallbackURL&response_type=code&scope="email user_likes"&state=7F32G5

- My app receives temporary auth code from the thing's vendor, regenerates state and compares with the received one:

  https://myCallbackURL?code=54321&state=7F32G5

- ,My app makes another request adding the secret and exchanging the code for the authorization token:

  https://<auth_server>/path?clientid=123&client_secret=…&code=54321&redirect_uri=https//myCallbackURL&grant_type=authorization_code

A Sample OAuth 2 Workflow

- My company registers the app with the thing's vendor: providing a **redirect URI** for successful and failed logins and gets a **client id** and a **secret**.

- My company builds an app that uses the thing's API (e.g. with REST)

- The user opens my app and logs into thing's vendor site via its authentication server (not the OAuth provider).

- My app (not the browser) generates a session-based random state and sends the request to the thing vendor's OAuth provider:

  https://<auth_server>/path?clientid=123&redirect_uri=https//myCallbackURL&response_type=code&scope="email user_likes"&state=7F32G5

- My app receives temporary auth code from the thing's vendor, regenerates state and compares with the received one:

  https://myCallbackURL?code=54321&state=7F32G5

- ,My app makes another request adding the secret and exchanging the code for the authorization token:

  https://<auth_server>/path?clientid=123&client_secret=…&code=54321&redirect_uri=https//myCallbackURL&grant_type=authorization_code

- The thing's vendor redirects the user to my app and returns the authorization token.

A Sample OAuth 2 Workflow

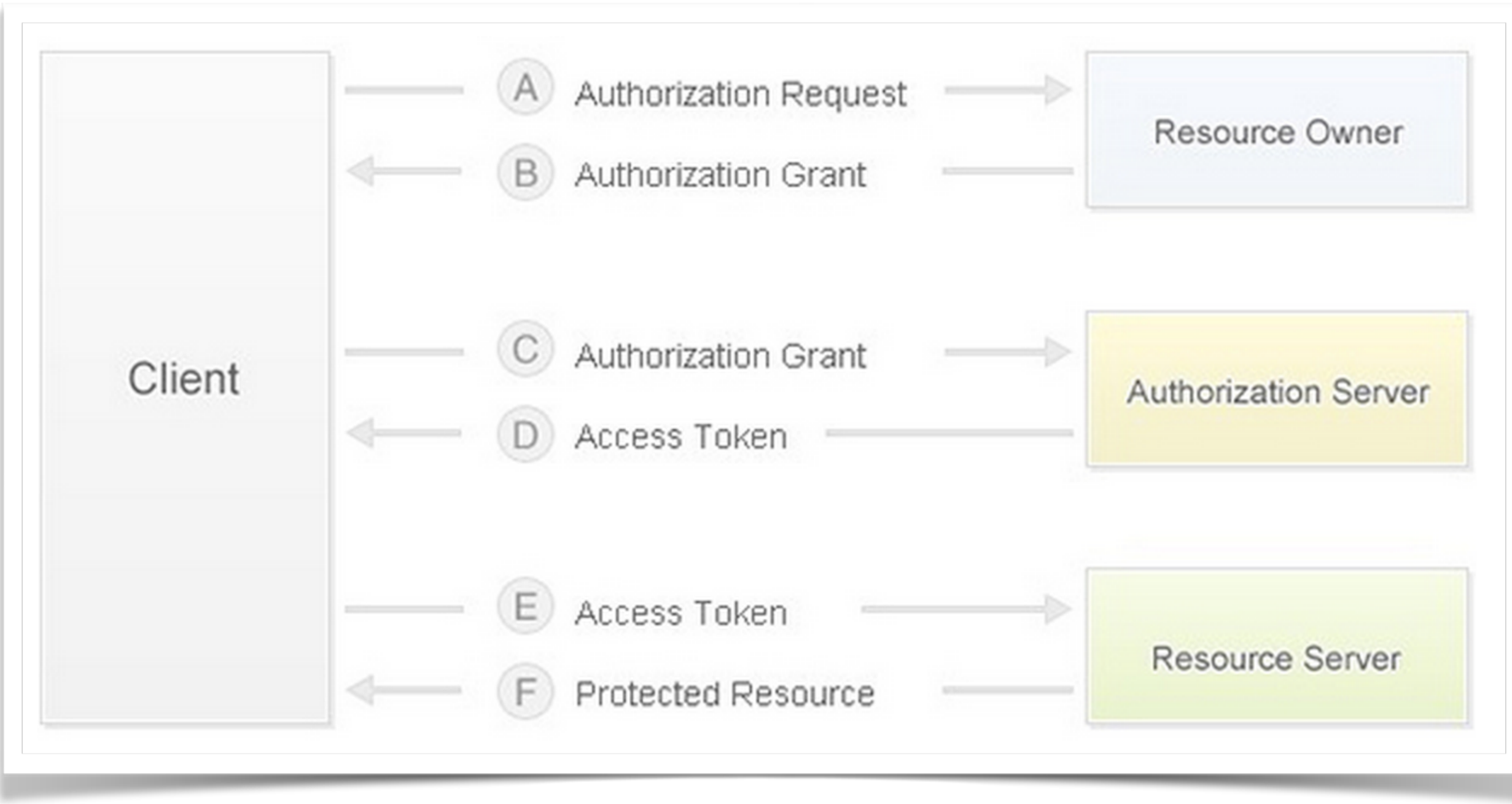
- **My company** registers the app with the thing's vendor providing a **redirect URI** for successful and failed logins and gets a **client id** and a **secret**.
- **My company** builds an app that uses the thing's API (e.g. with REST)
- **The user** opens **my app** and logs into thing's vendor site via its authentication server (not the OAuth provider).
- My app (not the browser) generates a session-based random state and sends the request to the thing vendor's OAuth provider:

  https://<auth_server>/path?clientid=123&redirect_uri=https//myCallbackURL&response_type=code&scope="email user_likes"&state=7F32G5
- **My app** receives temporary auth code from the thing's vendor, regenerates state and compares with the received one:

  https://myCallbackURL?code=54321&state=7F32G5
- ,My app makes another request adding the secret and exchanging the code for the authorization token:

  https://<auth_server>/path?clientid=123&client_secret=…&code=54321&redirect_uri=https//myCallbackURL&grant_type=authorization_code
- **The thing's vendor** redirects the user to my app and provides the **authorization token**.
- **My app starts invoking the vendor's API using the token.**

iHealthLabs Authorization



Access and Refresh Tokens

- The OAuth 2 server returns the **authorization token**. It expires after certain time interval. iHealtLabs sends the token in JSON format that expires in 10 min.

- The OAuth 2 server also provides a **refresh token** that the application uses to request a new token instead of the expired one.

WebSocket Protocol

Bi-directional communication for the Web

HTTP Hacks for Server's push

- HTTP is request-based and high-overhead protocol
- Hacks for achieving the server-side "push":
  - Polling
  - Long Polling
  - HTTP Streaming
  - Server-Side Events (SSE)

Monitoring AJAX requests

https://www.google.com/finance?q=AAPL

Google AAPL

Finance **Apple Inc.** (NASDAQ:AAPL) Add to portfolio

Company
Summary
News
Option chain
Related companies
Historical prices
Financials

113.08 +1.30 (1.16%)
Real-time: 11:23AM EST
NASDAQ real-time data - Disclaimer
Currency in USD

Range	111.97 - 113.49	Div/yield	0.47/1.66
52 week	70.51 - 119.75	EPS	6.43
Open	112.16	Shares	5.86B
Vol / Avg.	12.99M/50.32M	Beta	0.92
Mkt cap	663.18B	Inst. own	62%
P/E	17.59		

g+1 7.6k

Elements Network Sources Timeline Profiles Resources Audits Console AngularJS

Preserve log Disable cache

Filter All Documents Stylesheets Images Media Scripts XHR Fonts TextTracks WebSockets Other Hide data URLs

Name / Path	Method	Status / Text	Type	Initiator	Size / Content
getprices?q=AAPL&x=NASD&i=120&p=25m&f=d,c,v,o,h,l&df=cpct&auto=1&ts=1419265315339&ei=qk... /finance	GET	200 OK	text/plain	Other	359 B 617 B
channel?VER=8&SID=A77C764226E2BA1&RID=29572&AID=151&zx=y49rttmj3t6u&t=1 /finance/qs	POST	200 OK	text/plain	d=0:150 Script	71 B 13 B
getprices?q=AAPL&x=NASD&i=120&p=25m&f=d,c,v,o,h,l&df=cpct&auto=1&ts=1419265375384&ei=qk... /finance	GET	200 OK	text/plain	Other	374 B 656 B
channel?VER=8&SID=A77C764226E2BA1&RID=29573&AID=212&zx=39z7odxoqnoy&t=1 /finance/qs	POST	200 OK	text/plain	d=0:150 Script	68 B 13 B
channel?VER=8&SID=A77C764226E2BA1&RID=29574&AID=276&zx=dax4ov6c72o1&t=1 /finance/qs	POST	200 OK	text/plain	d=0:150 Script	69 B 13 B
getprices?q=AAPL&x=NASD&i=120&p=25m&f=d,c,v,o,h,l&df=cpct&auto=1&ts=1419265435487&ei=qk... /finance	GET	200 OK	text/plain	Other	363 B 615 B

6 / 10 requests | 1.3 KB / 173 KB transferred

Introducing WebSocket

- Standardized full-duplex low overhead protocol.
- Client-side API: Web browser's `window.WebSocket` object or your Java app.
- Server-side API: Java EE 7, Spring Framework, etc.
- All modern browsers support WebSocket protocol

http://caniuse.com/websockets

Apps for Websockets

- Live trading/auctions/sports notifications
- Controlling medical equipment over the web
- Chat applications
- Multiplayer online games

The WebSocket Workflow

- Establish a connection with the server's endpoing upgrading the protocol form HTTP to WebSocket
- Send messages in both directions at the same time (Full Duplex)
- Close the connection

WebSocket Client/Server handshake

- Client sends UPGRADE HTTP-request
- Server confirms UPGRADE
- Client receives UPGRADE response
- Client sets `readyState=1` on the WebSocket object

Web Browser WebSocketClient

- Initiate the connection to the server's endpoint by creating an instance of WebSocket object providing the URL of the server
- Write an `onOpen()` callback function
- Write an `onMessage()` callback
- Write an `onClose()` callback
- Write an `onError()` callback

The JavaScript Client

```
if (window.WebSocket) {
    ws = new WebSocket("ws://www.websocket.org/echo");

    ws.onopen = function() {
        console.log("onopen");
    };

    ws.onmessage = function(e) {
       console.log("echo from server : " + e.data);
    };

    ws.onclose = function() {
       console.log("onclose");
    };
    ws.onerror = function() {
       console.log("onerror");
    };

} else {
   console.log("WebSocket object is not supported");
}
```

Sending request to server: `ws.send(“Hello Server”);`

Java EE WebSocket Server's APIs

1. Annotated WebSocket endpoint

Annotate a POJO with `@ServerEndpoint`, and its methods with `@OnOpen`, `@OnMessage`, `@OnError`, and `@OnClose`

2. Programmatic endpoint

Extend your class from `javax.websocket.Endpoint` and override `onOpen()`, `onMessage()`, `onError()`, and `onClose()`.

HelloWebSocket Server

The server-side push without client's requests

```
@ServerEndpoint("/hello")
public class HelloWebSocket {

  @OnOpen
  public void greetTheClient(Session session){
     try {
        session.getBasicRemote().sendText("Hello stranger");

     } catch (IOException ioe) {
        System.out.println(ioe.getMessage());
     }
  }
}
```

Code Fragment with Websockets in Spring

```
public class WebSocketEndPoint extends TextWebSocketHandler {
    private final static Logger LOG = LoggerFactory.getLogger(WebSocketEndPoint.class);

    private Gson gson;
    private WebSocketSession currentSession;

    @Override
    public void afterConnectionEstablished(WebSocketSession session) throws Exception {
        super.afterConnectionEstablished(session);

        setCurrentSession(session);
    }

    public boolean sendMeasurement(Measurement m) {
        if (getCurrentSession() != null) {
            TextMessage message = new TextMessage(getGson().toJson(m));

            try {
                getCurrentSession().sendMessage(message);
            } catch (IOException e) {
                e.printStackTrace();
                return false;
            }

            return true;
        } else {
            LOG.info("Can not send message, session is not established.");
            return false;
        }
    }
```


Deploying with Spring Boot

- Java EE REST services are deployed in a WAR under the external Java Server.
- Spring Boot allows creating a standalone app (a JAR) with an embedded servlet container.
- Starting our RESTful server: **java -jar MyJar**.
- We used Tomcat. To use another server, exclude Tomcat in build configuration and specify another dependency. Here's a sample section from Gradle build:

```
dependencies {
   compile("org.springframework.boot:spring-boot-starter-web") {
        exclude module: "spring-boot-starter-tomcat"
   }
   compile("org.springframework.boot:spring-boot-starter-jetty")
}
```

What about security?

- Device vendors should take security very seriously.
- We don’t deal with security between the thing and its vendor.
- We just use OAauth **state** attribute, and the OAuth provider must check that the received redirect_uri is the same as provided during the app registration.
- IoT integration apps are as as secure as any other Web app (see owasp.org).

Contact Info

- Farata Systems: faratasystems.com
- email: yfain@faratasystems.com
- Twitter: @yfain
- My blog: yakovfain.com
- My podcast: americhka.us