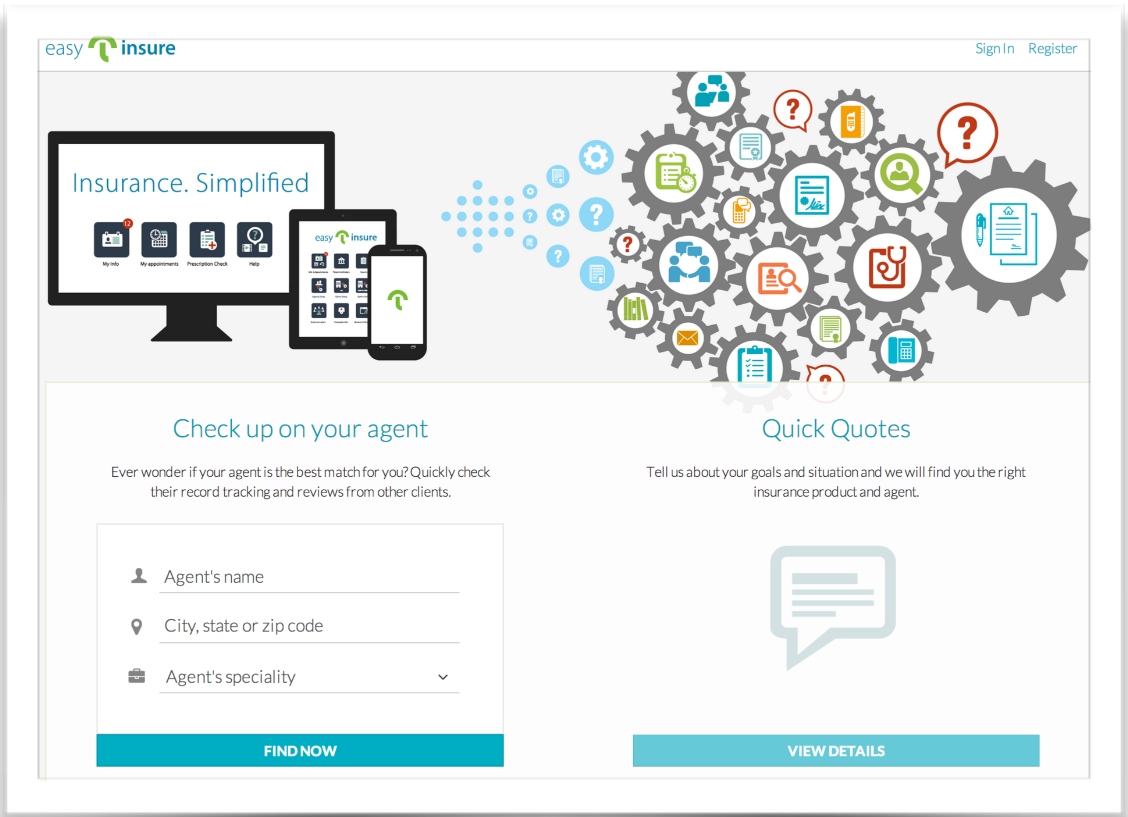
RESTful services and OAUTH protocol in IoT

by Yakov Fain, Farata Systems

Farata Systems and SuranceBay

We build applications. Every app is unique.
We create it. You own it.



Check up on your agent

Ever wonder if your agent is the best match for you? Quickly check their record tracking and reviews from other clients.

Quick Quotes

Tell us about your goals and situation and we will find you the right insurance product and agent.

http://easy.insure

The three parts of this presentation

- One approach to integrating consumer devices in the business workflow

- Live demo: integrating a blood pressure monitor into a business workflow

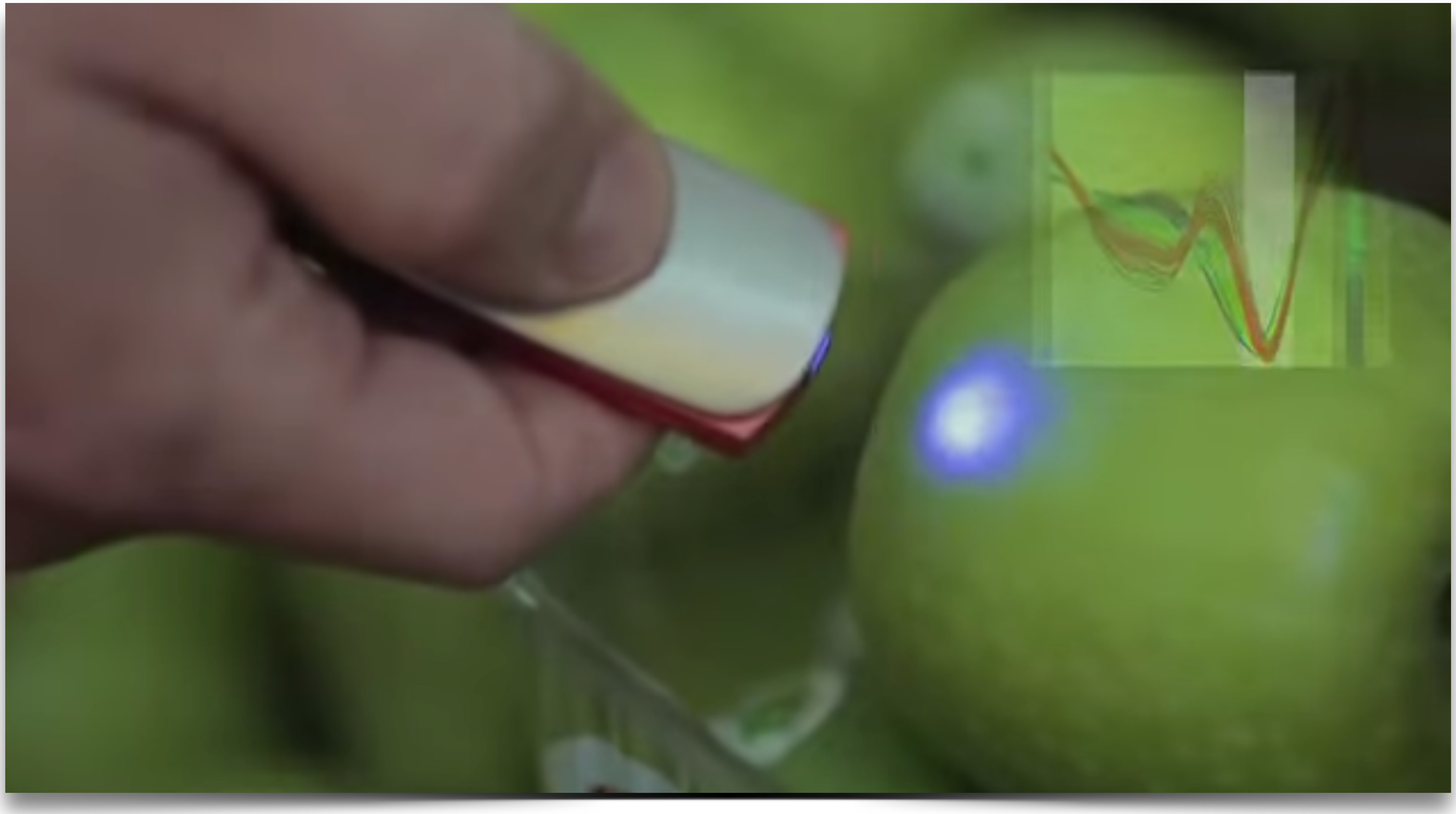
- A brief review of REST, OAUTH, Websockets and their roles tin our application.

Yesterday’s Sensors (Things)

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

Today’s Sensors

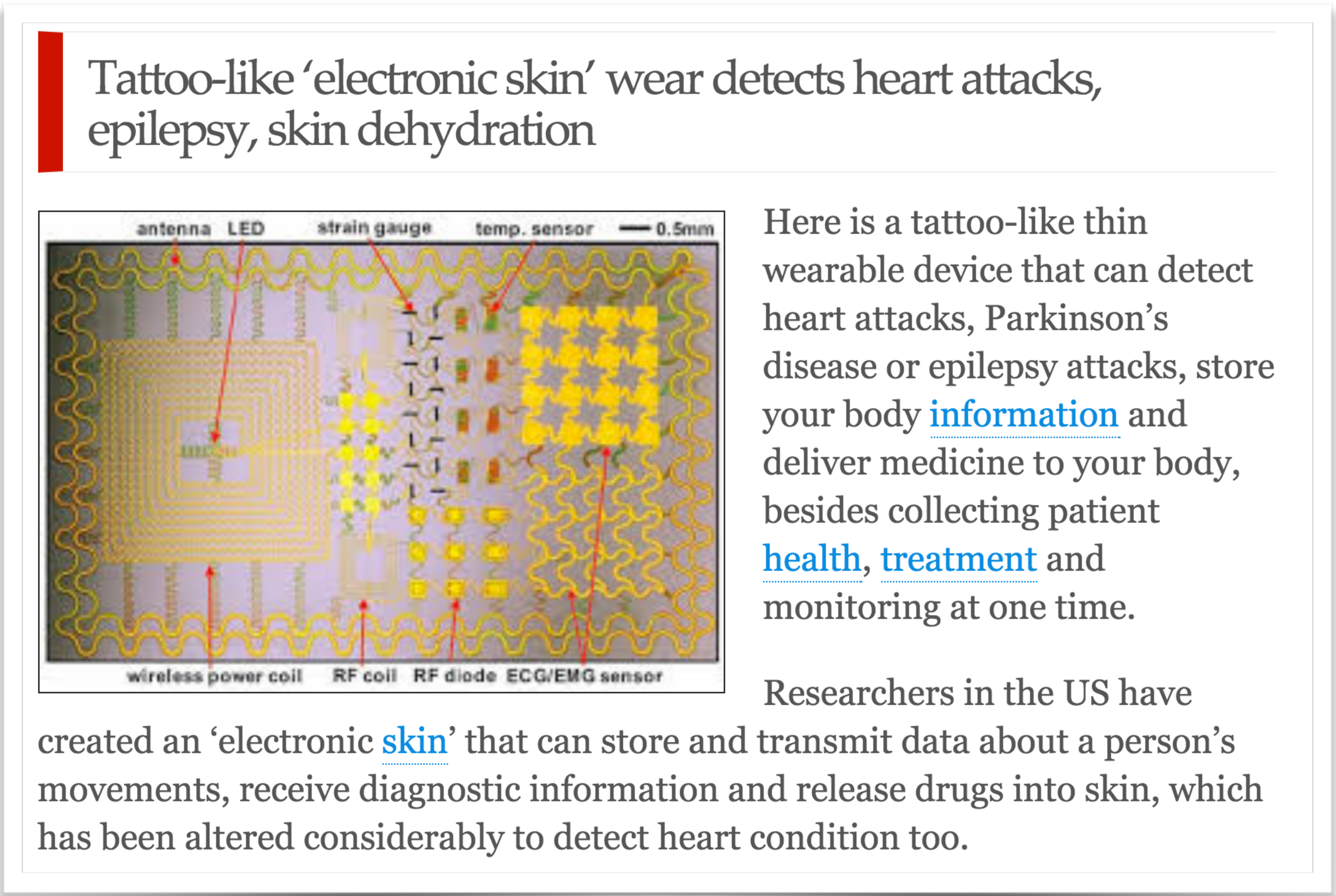
SCIO: a molecular sensor that scans physical objects and receives instant information to your smartphone.



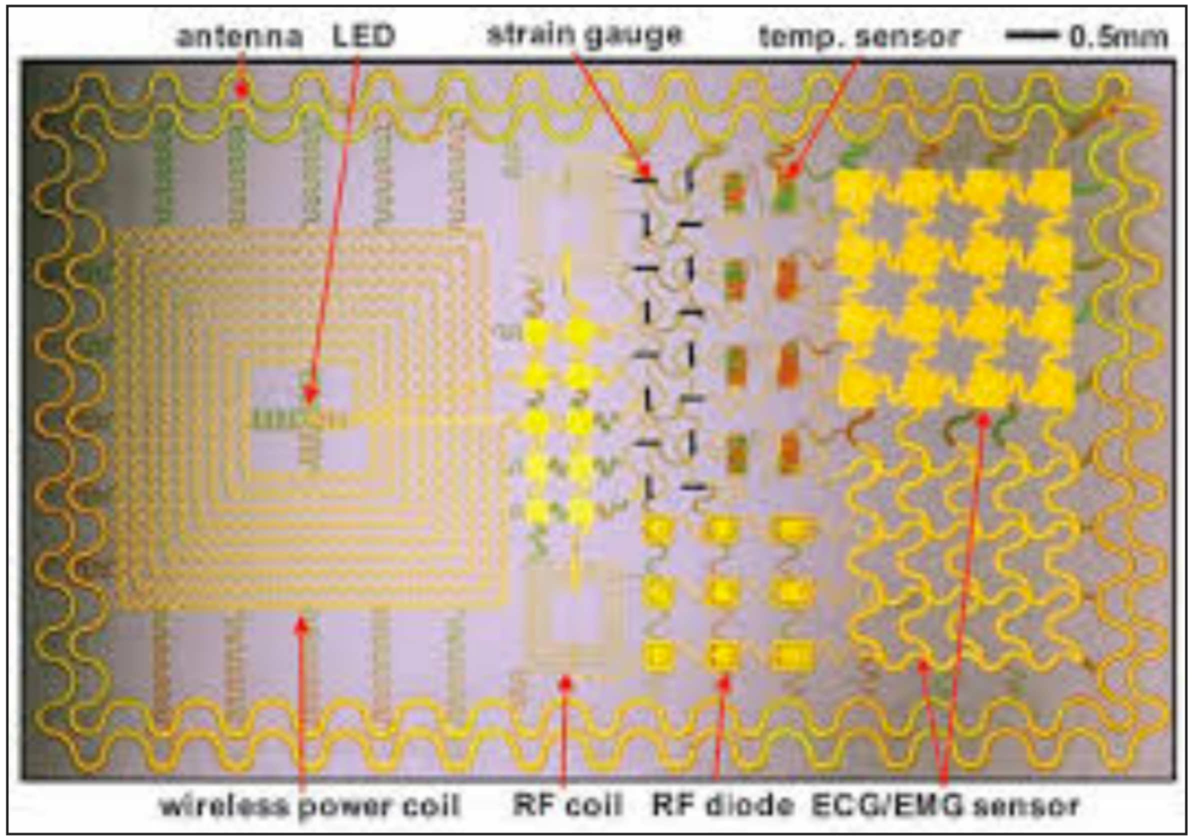
http://www.consumerphysics.com/

Tomorrow: Streachable Wearables

epidermal electronics



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



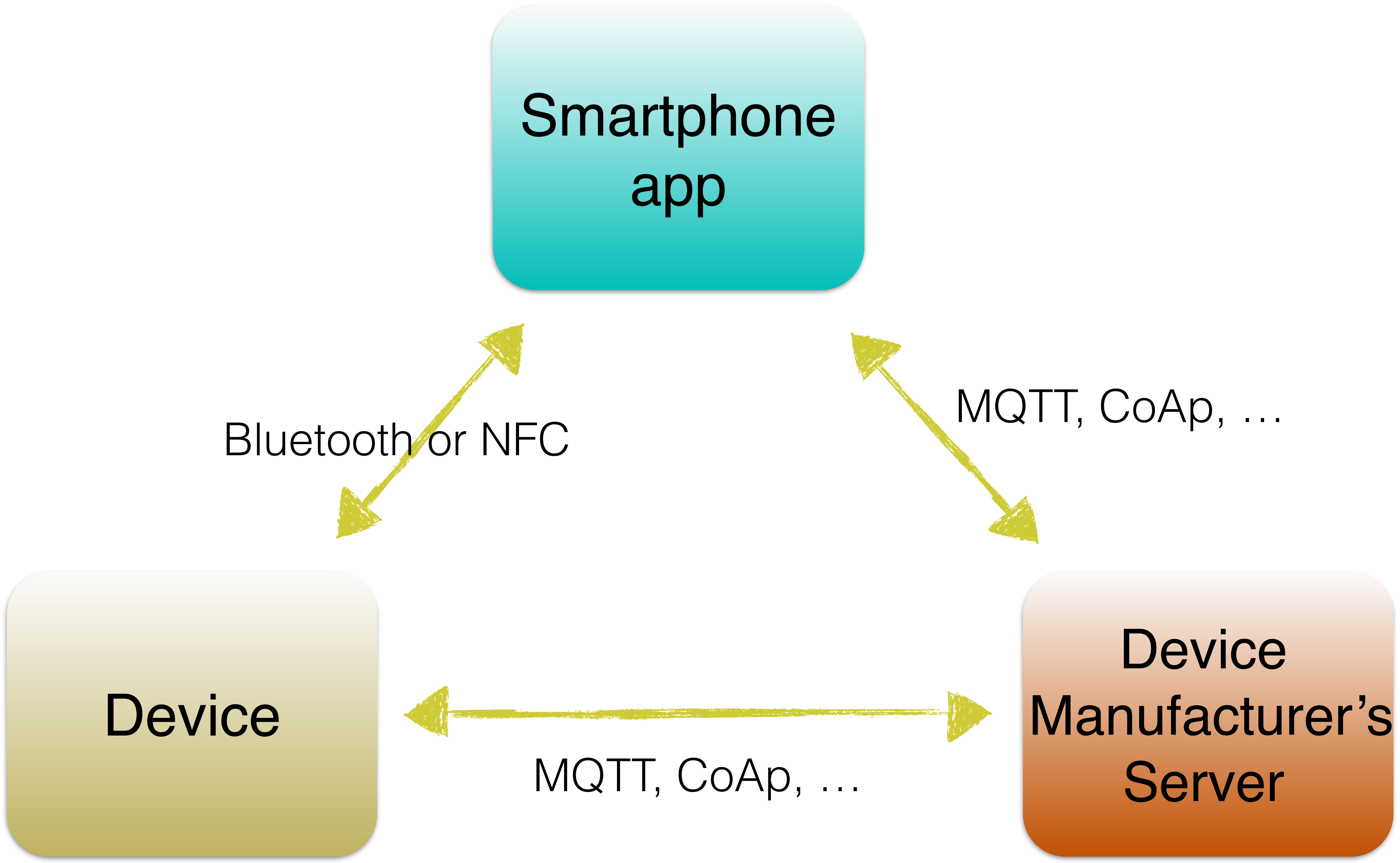
Here is a tattoo-like thin wearable device that can detect heart attacks, Parkinson’s disease or epilepsy attacks, store your body information and deliver medicine to your body, besides collecting patient health, treatment and monitoring at one time.

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 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 the valuable data from devices for analisys.

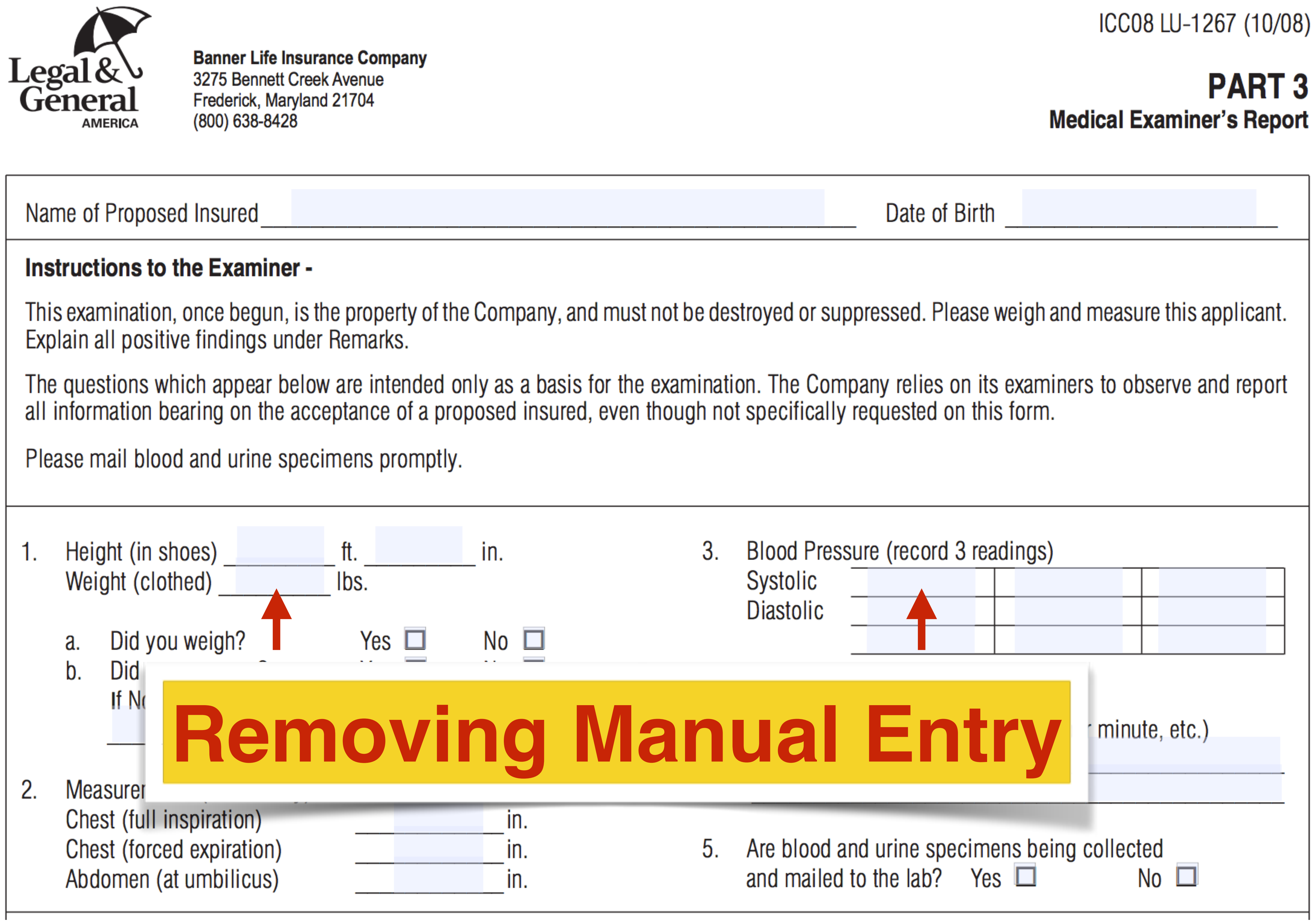
Java dominates on the middleware market.

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. Measure
   Chest (full inspiration) ________ in.
   Chest (forced expiration) ________ in.
   Abdomen (at umbilicus) ________ in.

3. Blood Pressure (record 3 readings)

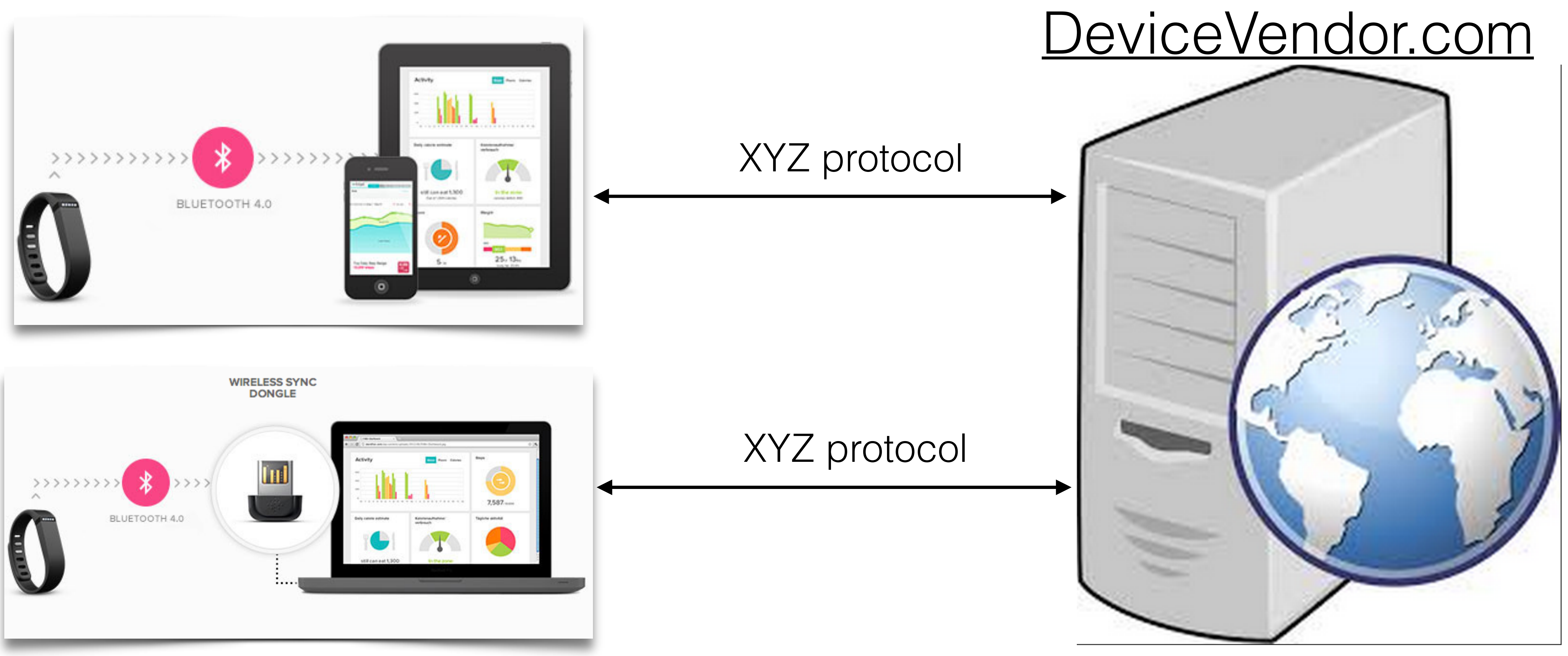
Systolic			
Diastolic			

minute, etc.)

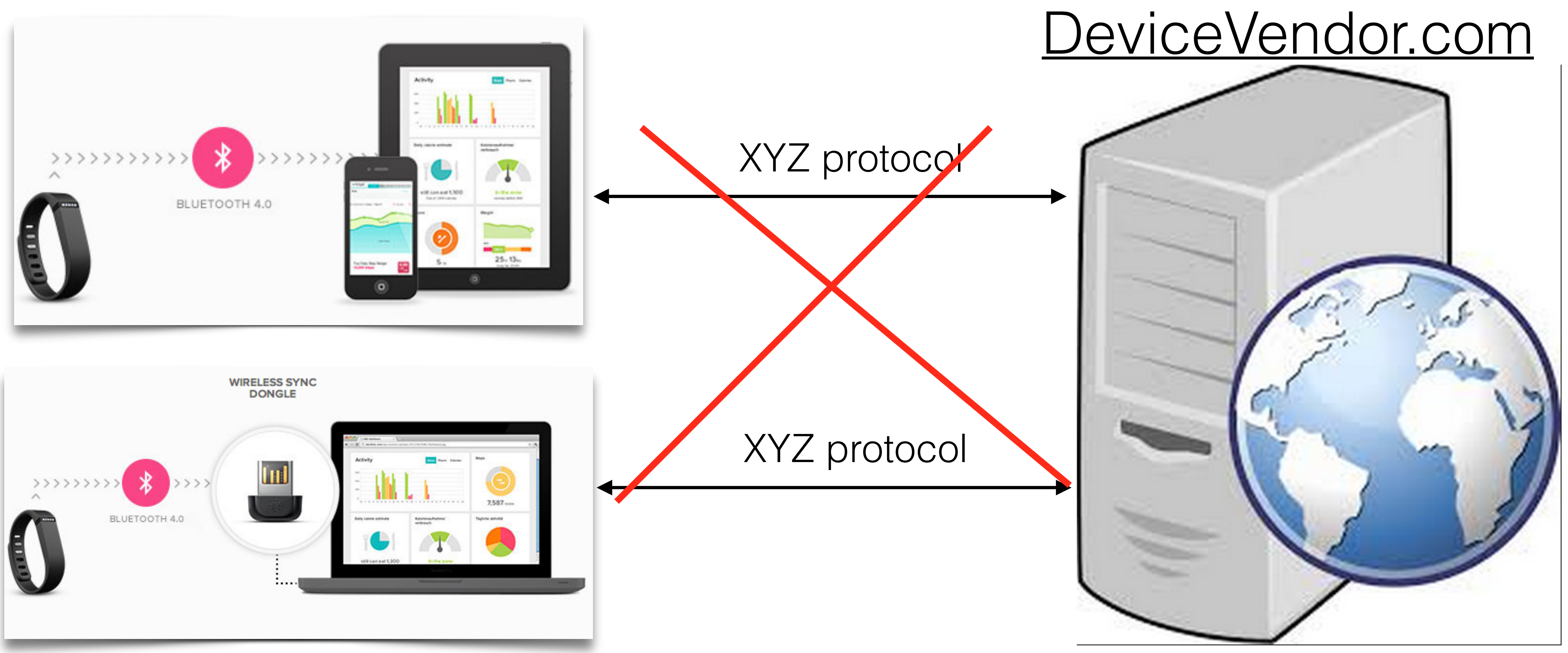
5. Are blood and urine specimens being collected and mailed to the lab? Yes ☐ No ☐

Removing Manual Entry

A Typical IoT Workflow



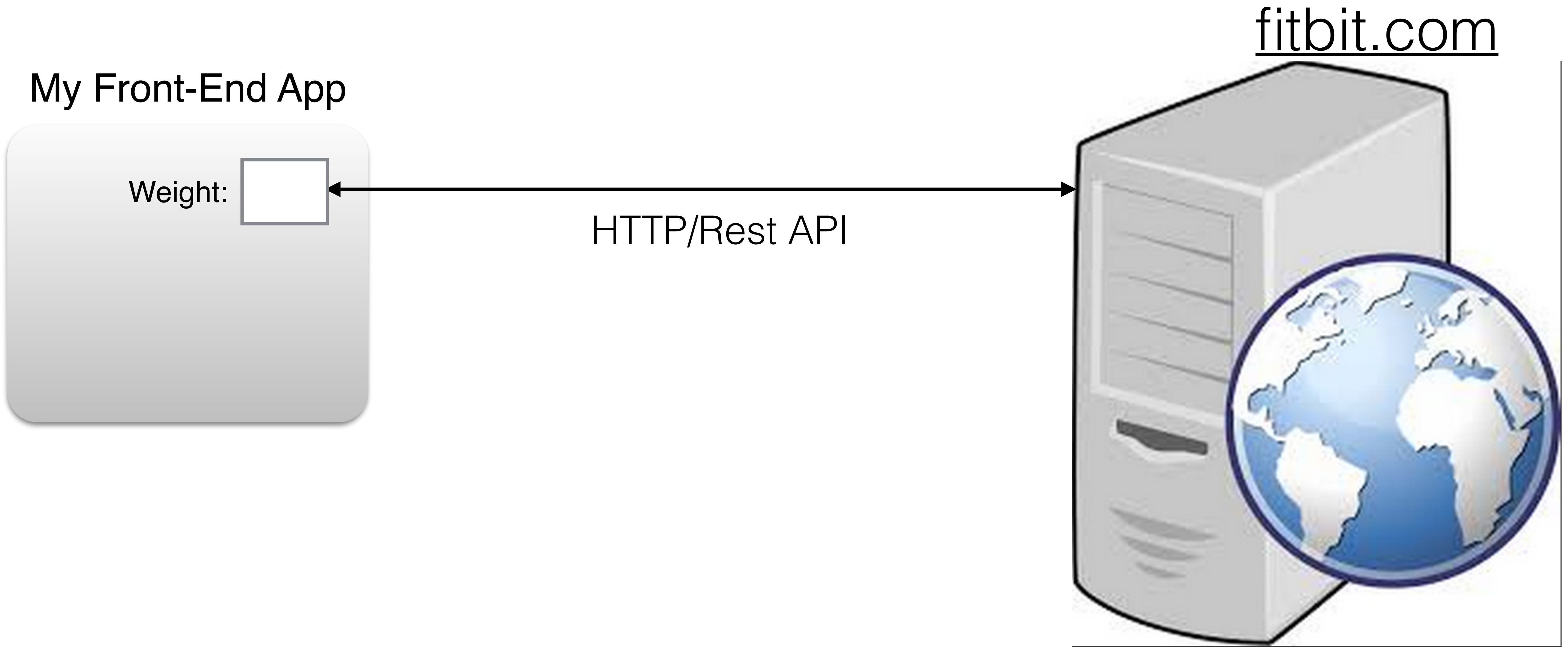
A Typical IoT Workflow



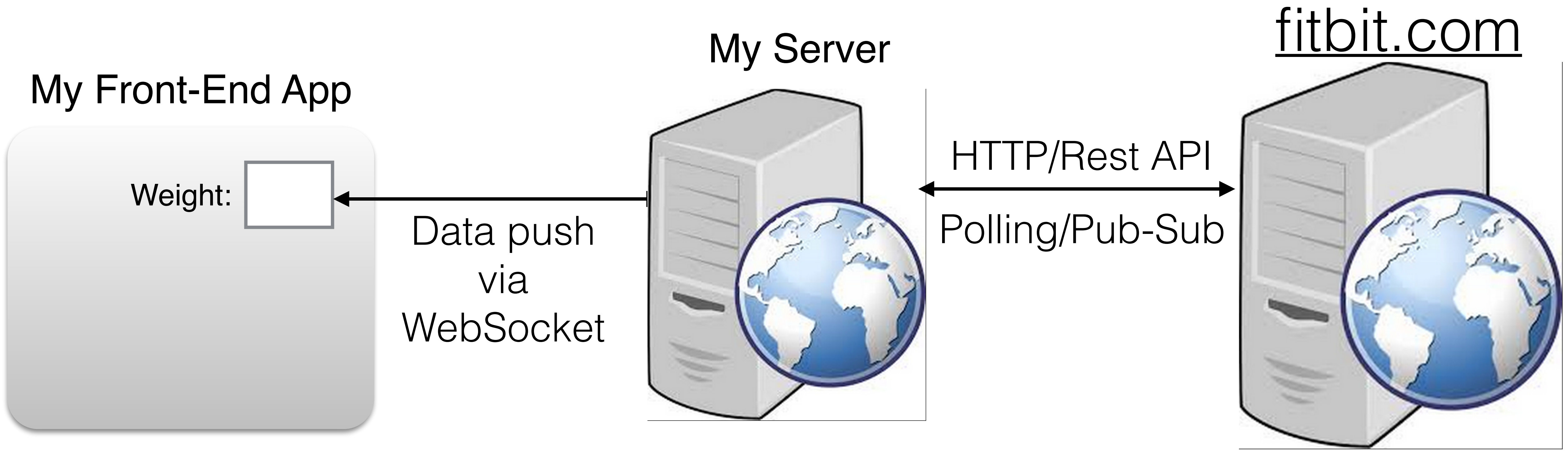
We're not dealing with XYZ

Our server communicates with the vendor's server using HTTPS

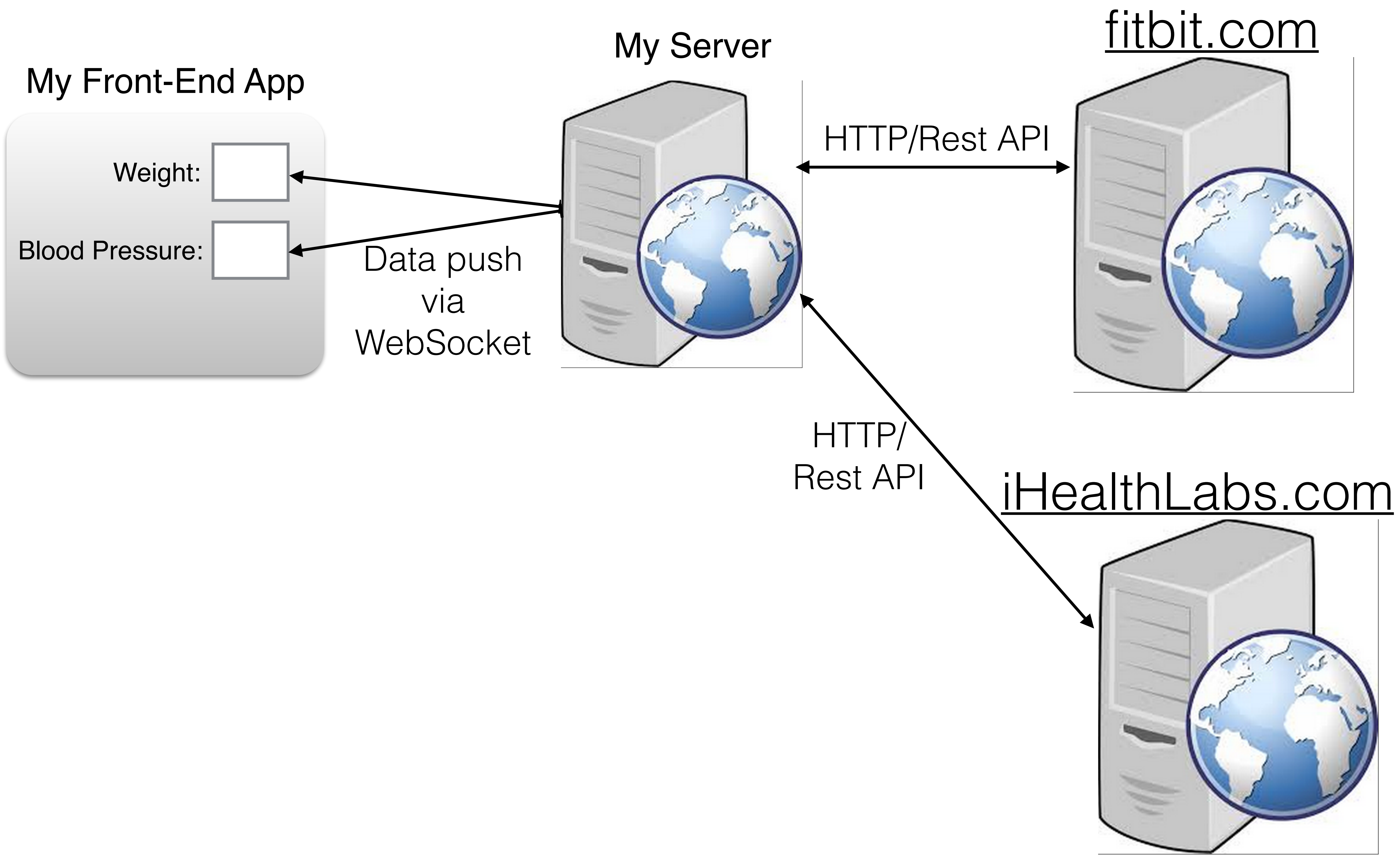
Integrating With Fitbit Scale: Take 1.



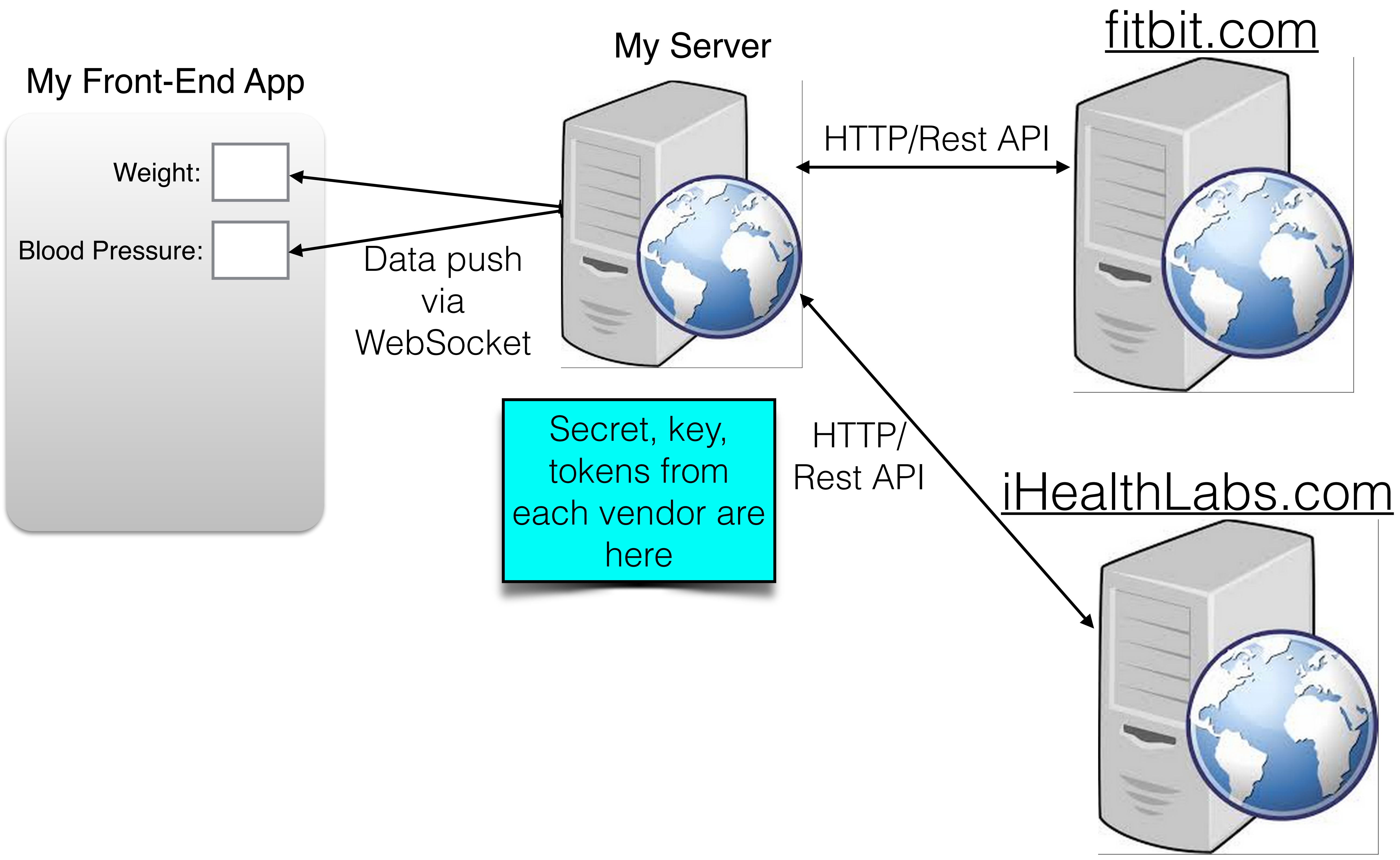
Integrating With Fitbit Scale: Take 2.



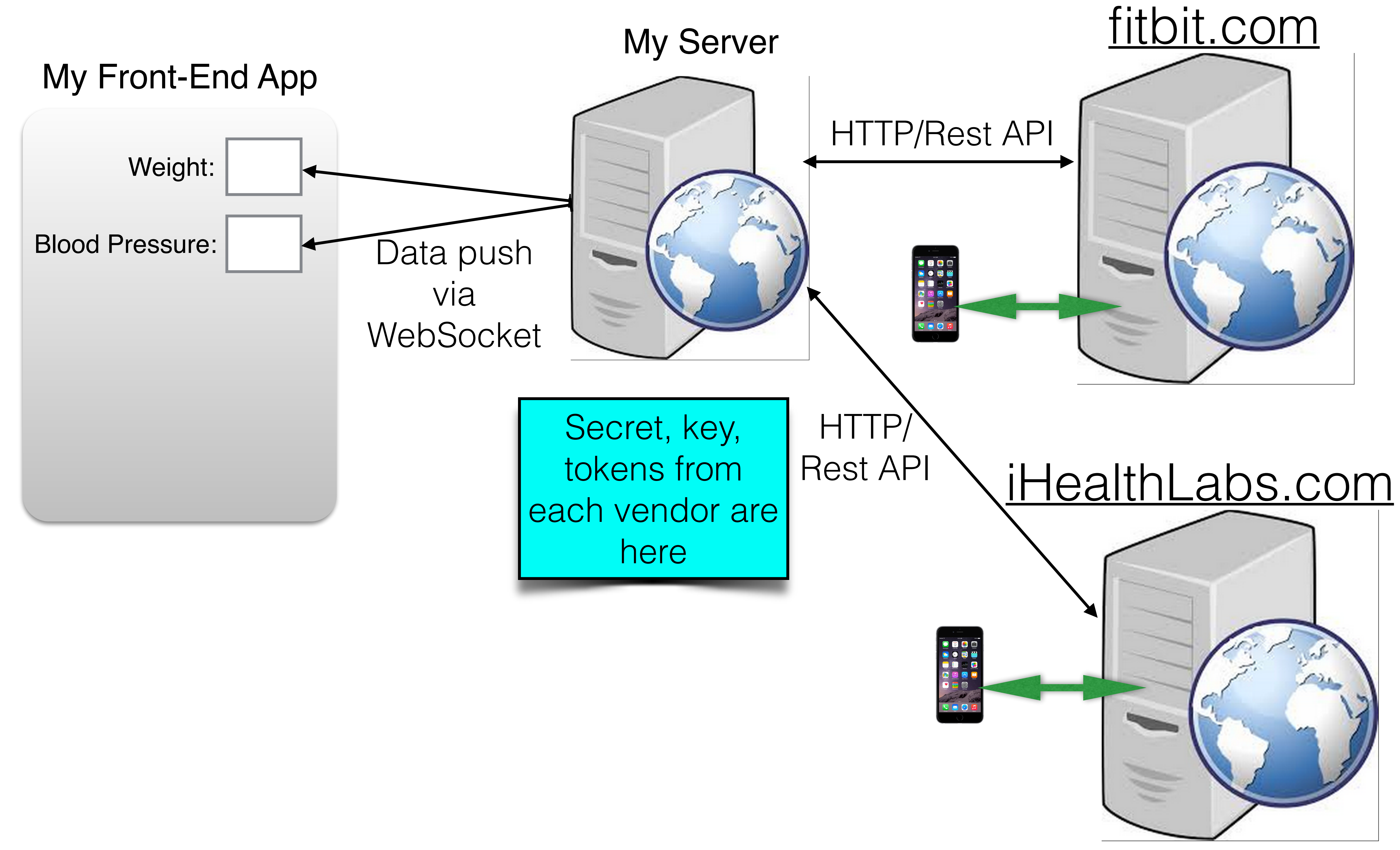
Integrating With Fitbit and iHealthLabs.



Adding OAuth Authentication



The Final Architecture



⟷ - Vendor’s consumer app

Demo

Measuring Blood Pressure

What's used in our app

- RESTful Web services
- OAuth authentication and authorization
- WebSocket protocol
- Front end: written in Dart, deployed as JavaScript
- Data exchange format: JSON
- Back-end: Java with Spring Boot and embedded Tomcat
- Build automation: Gradle

REST API

REpresentational State of Transfer

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;
  }
}
```

A Rest Endpoint in Spring Framework

```java
// 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;
  }
}
```

OAuth 2

Authorizing an app to act on behalf of the user

Authorization and Authentication

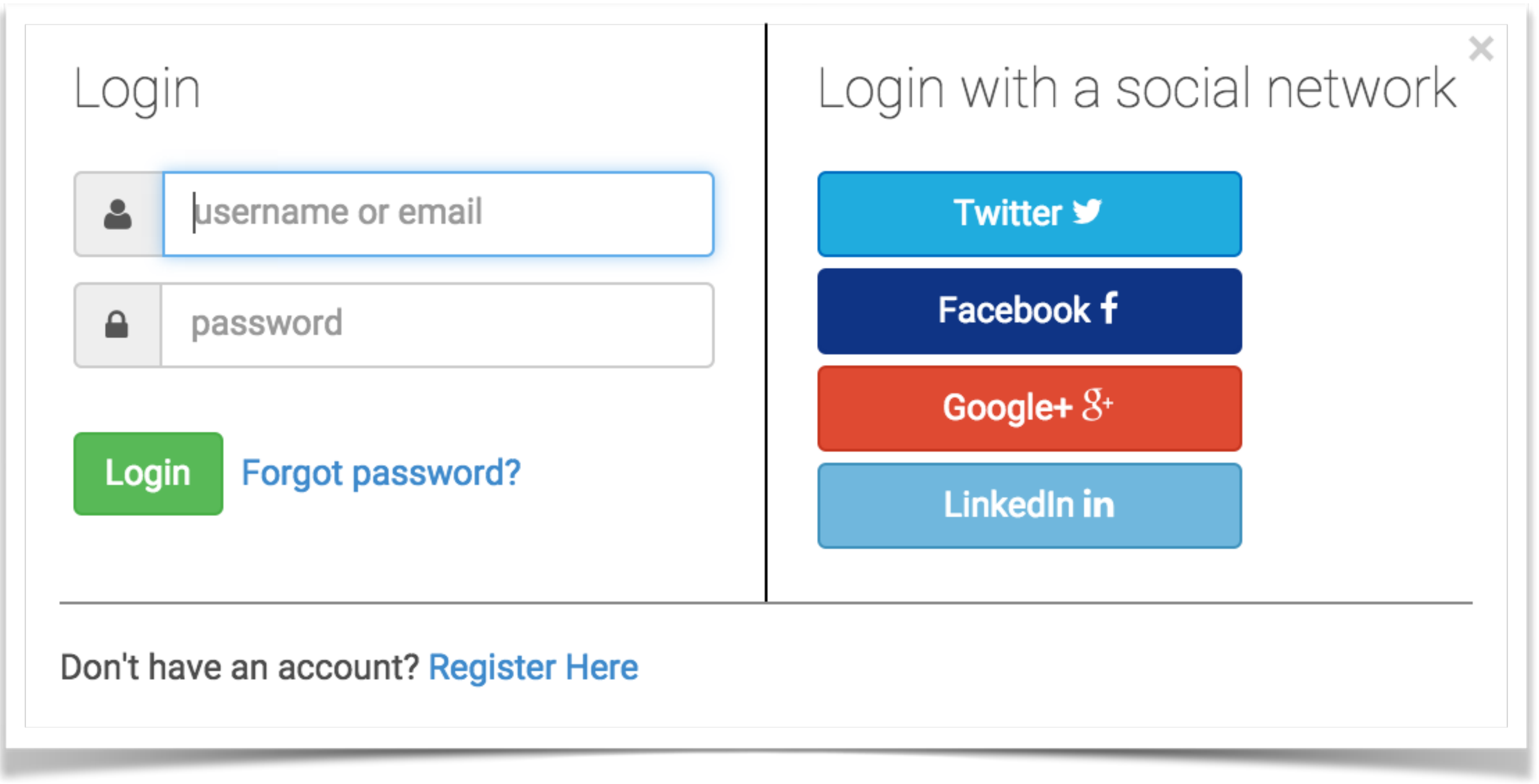
- **Authentication:** Is the user who he says he is?
- **Authorization:** Which resources the user can access?

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

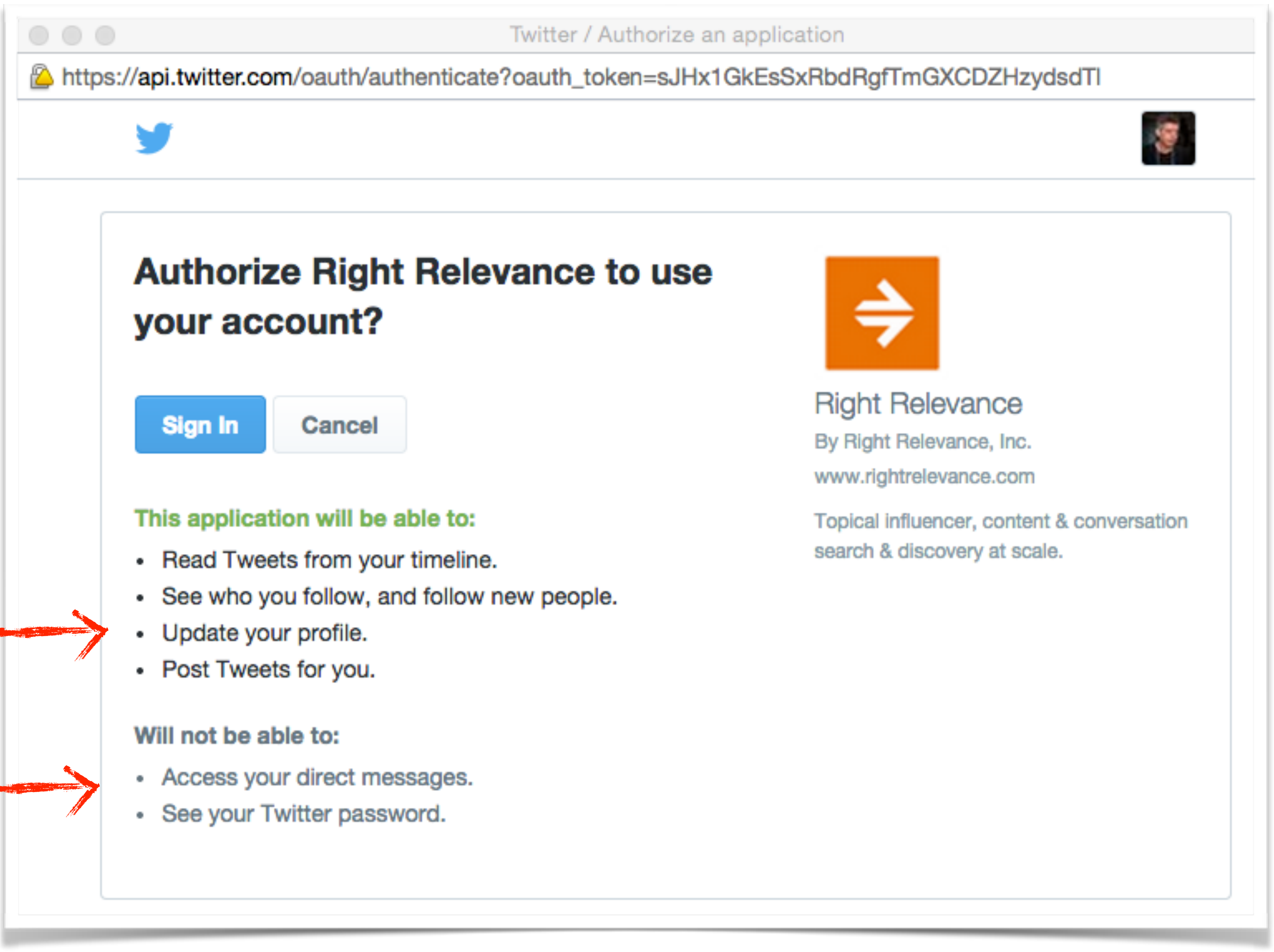
The OAuth Players

- The User
- The client app that accesses the user’s resources
- The server with the user’s resources (data)
- The authorization server

Delegating Authorization to 3rd Party Servers



Authorization with Hot Wi-Fi in Moscow


Twitter / Authorize an application

https://api.twitter.com/oauth/authenticate?oauth_token=sJHx1GkEsSxRbdRgfTmGXCDZHzydsdTl

Authorize Right Relevance to use your account?

This application will be able to:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

Will not be able to:

- Access your direct messages.
- See your Twitter password.

Right Relevance

By Right Relevance, Inc.

www.rightrelevance.com

Topical influencer, content & conversation search & discovery at scale.

Bad $\longmapsto$

Good $\longmapsto$

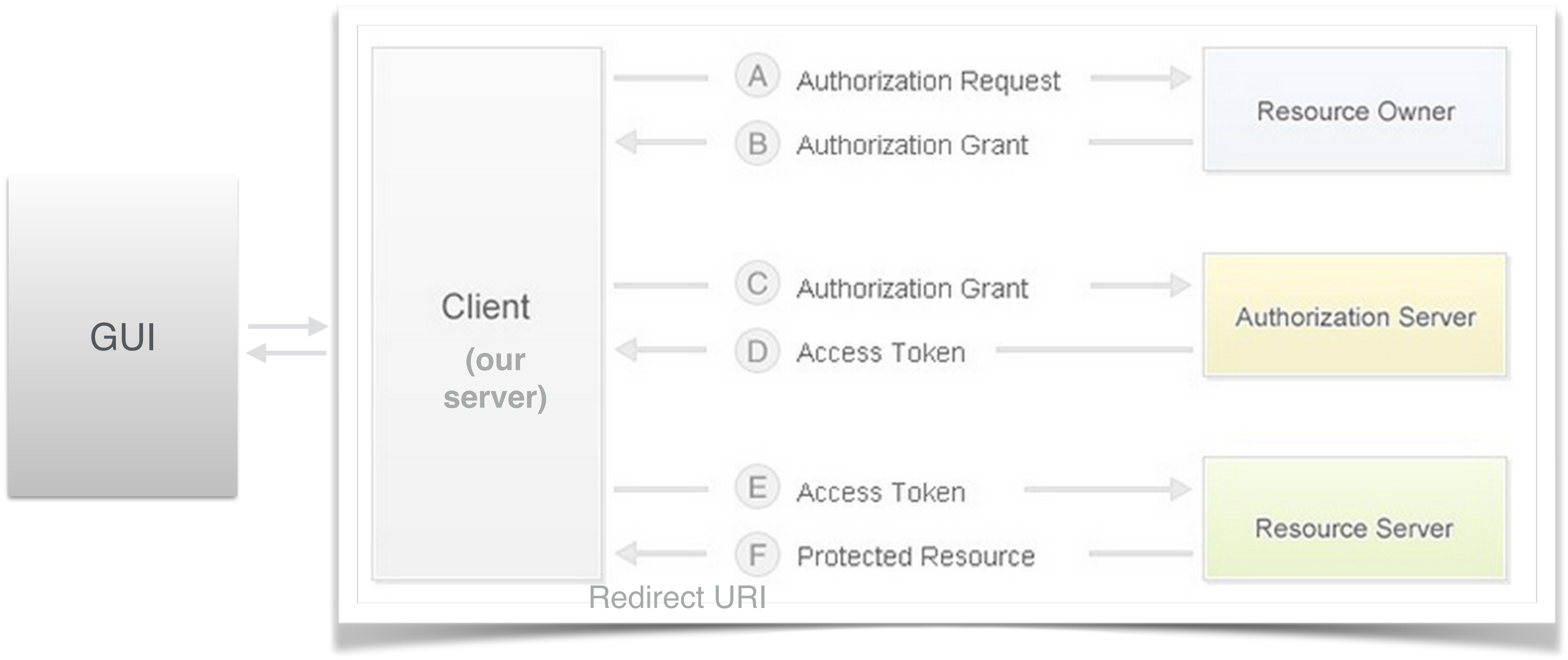
OAuth 2 Access Token

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

Typical OAuth 2 Workflows

- A client app is located on the user’s device
- A client app is located on the server (our use case)

iHealthLabs Authorization



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 a temporary auth code from the thing's OAuth server, regenerates the state and compares it with the received one from the server:

  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 OAuth server, regenerates the state and compares with the received one from the server:

  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 OAuth server, regenerates the state and compares with the received one from the server:

  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 OAuth server, regenerates the state and compares with the received one from the server:
  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.

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 can provide a refresh token that the client app uses to request a new token instead of the expired one.

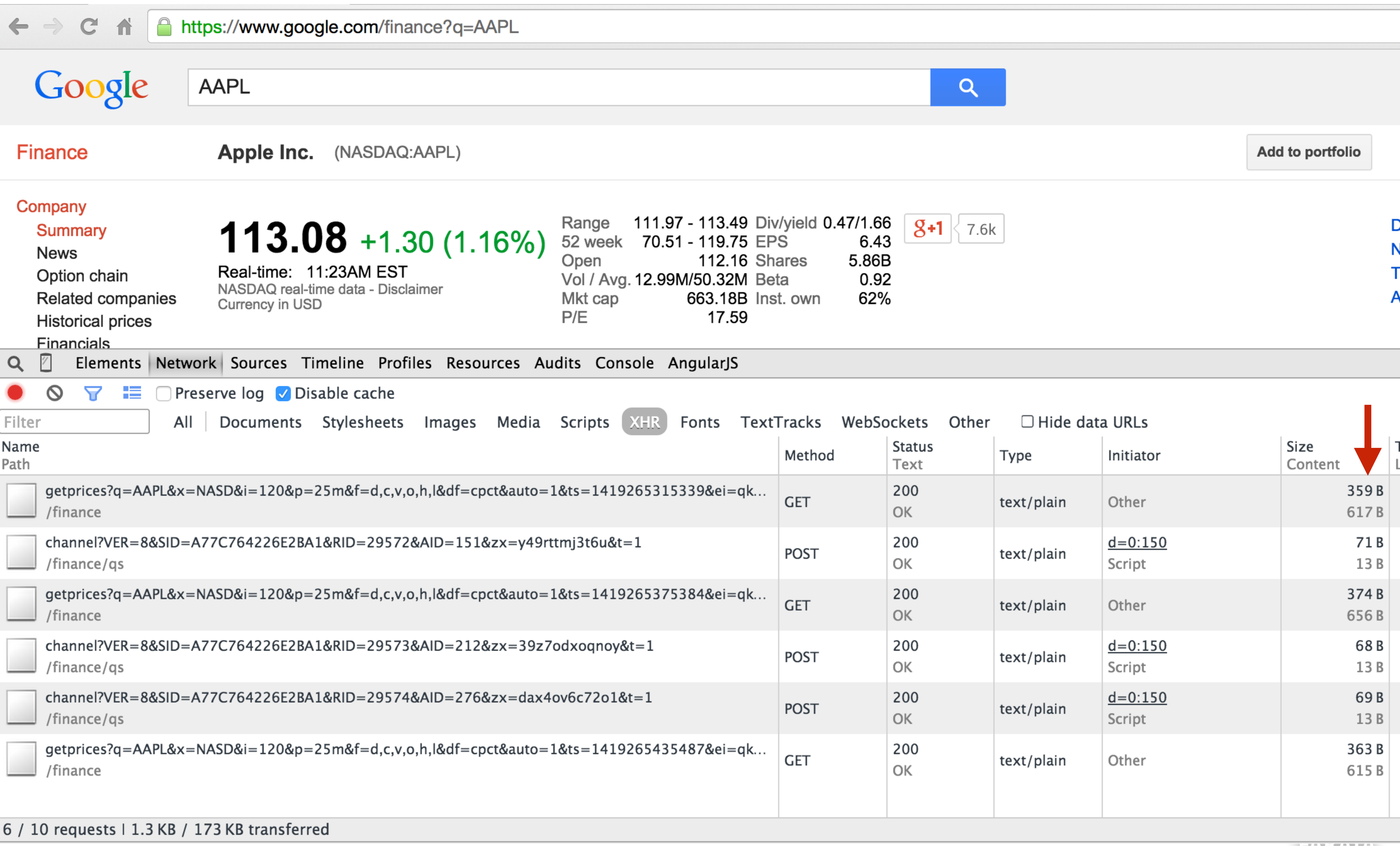
WebSocket Protocol

Bi-directional communication for the Web

HTTP - Request/Response, Half Duplex

WebSocket - Full Duplex

Monitoring AJAX requests



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

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		

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

WebSocket Workflow

- Establish connection with the service endpoint upgrading the protocol from HTTP to WebSocket

- Send messages in both directions at the same time (Full Duplex)

- Close the connection

Apps for Websockets

- Live trading/auctions/sports notifications
- Controlling medical equipment over the web
- Chat applications
- Multiplayer online games
- Any app that requires a data push from a server

WebSocket Client/Server handshake

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

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 a request: `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());
     }
  }
}
```

A detailed description at http://bit.ly/1DHuKwg

Websockets with Spring Framework

```
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.
- A sample section from Gradle build replacing Tomcat with Jetty:

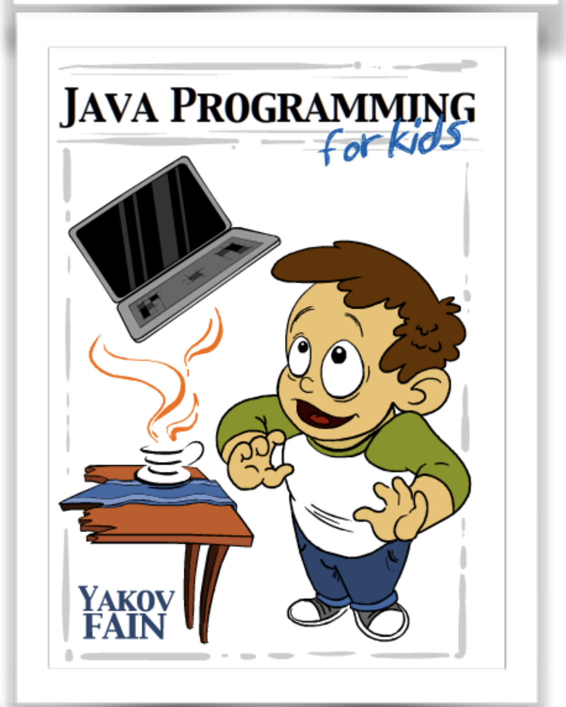
```
dependencies {
   compile("org.springframework.boot:spring-boot-starter-web") {
        exclude module: "spring-boot-starter-tomcat"
   }
   compile("org.springframework.boot:spring-boot-starter-jetty")
}
```

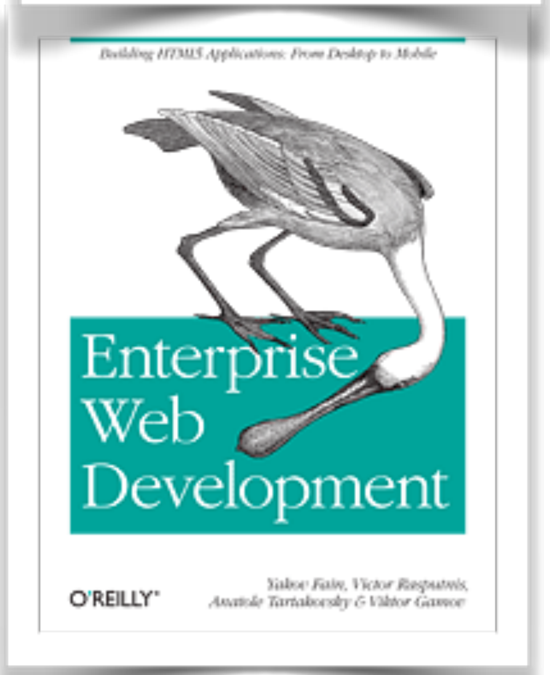
Security

- Device vendors should take security very seriously.
- We don't deal with security between the thing and its vendor.
- The OAuth **state** attribute helps ensuring 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).

Thank you!

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




Building HTML5 Applications: From Desktop to Mobile
Enterprise Web Development
O'REILLY
Yakov Fain, Victor Rasputnis, Anatole Tartakovsky & Viktor Gamov