IOT
Chapter 7
PROTOTYPING ONLINE COMPONENTS

Introduction:

- Each component has a critical part to play.
- The physical, designed object ties into the design principles of context.
- The controller and associated electronics allow it to sense and act on the real world.
- The Internet adds a dimension of communication.
- The network allows the device to inform you or others about events or to gather data and let you act on it in real time.
- It lets you aggregate information from disparate locations and types of sensors.
- Similarly, it extends your reach, so you can control or activate things from afar

Q1)What is an API? Explain the concept of Mashing and scrapping APIs.

GETTING STARTED WITH AN API

- The most important part of a web service, with regards to an Internet of Things device, is the Application Programming Interface, or API.
- An API is a way of accessing a service that is targeted at machines rather than people.
- If you think about your experience of accessing an Internet service, you might follow a number of steps.
- **For example, to look at a friend's photo on Flickr, you might do the following:**
- **1.** Launch Chrome, Safari, or Internet Explorer.
- **2.** Search for the Flickr website in Google and click on the link.
- **3.** Type in your username and password and click "Login".
- **4.** Look at the page and click on the "Contacts" link.
- **5.** Click on a few more links to page through the list of contacts till you see the one you want.
- **6.** Scroll down the page, looking for the photo you want, and then click on it.
- Although these actions are simple for a human, they involve a lot of looking, thinking, typing, and clicking.
- A computer can't look and think in the same way.
- The tricky and lengthy process of following a sequence of actions and responding to each page is likely to fail the moment that Flickr slightly changes its user interface.
- For example, if Flickr rewords "Login" to "Sign in", or "Contacts" to "Friends", a human being would very likely not even notice, but a typical computer program would completely fail.
- Instead, a computer can very happily call defined commands such as login or get picture #142857.

- **MASHING UP APIS**
- (Meaning: something created by combining elements from two or more sources.)
- Perhaps the data you want is already available on the Internet but in a form that doesn't work for you?
- The idea of "mashing up" multiple APIs to get a result has taken off and can be used to powerful effect.
- For example:
- ▪ Using a mapping API to plot properties to rent or buy.
- ▪ Showing Twitter trends on a global map or in a timeline or a charting API.

- ▪ Fetching Flickr images that are related to the top headlines retrieved from newspaper's API.

SCRAPING

- (meaning: the action of using a computer program to copy data from a website.)
- **Screen scraping** is the process of collecting **screen** display data from one application and translating it so that another application can display it.
- In many cases, companies or institutions have access to fantastic data but don't want to or don't have the resources or knowledge to make them available as an API.
- This is normally done to capture data from a legacy application in order to display it using a more modern user interface.
-
- **Here are a few examples:**
- Adrian has scraped the Ship AIS system to get data about ships on the river Mersey, and this information is then tweeted by the @merseyshipping account.
- The Public Whip website (www.publicwhip.org.uk/) is made possible by using a scraper to read the Hansard transcripts of UK government sessions (released as Word documents).
- With the resultant data, it can produce both human- and machine-readable feeds of how our elected representatives vote.
- The ScraperWiki site (https://scraperwiki.com) has an excellent platform for writing scrapers, in a number of dynamic programming languages, which collate data into database tables.
- It provides infrastructure for scripts that you could run on your own computer or server but allows you to outsource the boring, repetitive parts to ScraperWiki.
- **LEGALITIES**
- Screen-scraping may break the terms and conditions of a website.
- For example, Google doesn't allow you to screen-scrape.
- Alternative sources of information often are available.
- For example, you could use OpenStreetMap instead of Google Maps.

Q2)How to write new API? Explain with example of Timer.

WRITING A NEW API

- You plan to assemble the data from free or licensed material you have and process it.
- Or perhaps your Internet-connected device can populate this data.
- The process of building your own API is explained with an example project, Clockodillo.
- This is an Internet of Things device that Hakim built to help him use the Pomodoro time management technique
- The technique uses a timer to break down work into intervals, traditionally 25 minutes in length, separated by short breaks.
- These intervals are named ***pomodoros***, the plural in English of the Italian word *pomodoro* (tomato)
- (the tomato-shaped kitchen timer).
- 

- Clockodillo explores how the Internet of Things might help with that:
- connecting the kitchen-timer to the Internet to make the tracking easier while keeping the simplicity of the physical twist-to-set timer for starting the clock and showing progress as it ticks down.
- Clockodillo is an Internet-connected task timer
- The user can set a dial to a number of minutes, and the timer ticks down until completed.
- It also sends messages to an API server to let it know that a task has been started, completed, or cancelled.
- **A number of API interactions deal precisely with those features of the physical device:**
- ▪ Start a new timer
- ▪ Change the duration of an existing timer
- ▪ Mark a timer completed
- ▪ Cancel a timer
- Some interactions with a timer data structure are too complicated to be displayed on a device consisting mostly of a dial—for example, anything that might require a display or a keyboard!
- Those could be done through an app on your computer or phone instead.
- ▪ View and edit the timer's name/description And, naturally, the user may want to be able to see historical data:
- ▪ Previous timers, in a list
- • Their name/description
- • Their total time and whether they were cancelled
- Assume that each device will send some identifying token, such as a MAC address.
- The user will somehow identify himself with the server, after which all the preceding actions will relate just to a given user ID.

Q3) Explain the concept of security with example of Timer.

SECURITY

- How important security is depends a lot on how sensitive the information being passed is and whether it's in anyone's interest to compromise it.
- For Clockodillo, perhaps a boss might want to double-check that employees are using the timer.
- Or a competitor might want to check the descriptions of tasks to spy what your company is working on.
- Or a competitor might want to disrupt and discredit the service by entering fake data.
- If the service deals with health or financial information, it may be an even more attractive target.
- Location information is also sensitive; burglars might find it convenient to know when you are out of the house.

Task	Inputs	Outputs
1. Create a new timed task	User, Timer duration	Timer ID
2. Change duration of timed task	User, Timer ID, New duration	OK
3. Mark timer complete	User, Timer ID	OK
4. Cancel timer	User, Timer ID	OK
5. Describe the timed task	User, Timer ID, Description	OK
6. Get list of timers	User	List of Timer IDs
7. Get information about a timer	User, Timer ID	Description, Create time, Status

-
- The request has to pass details to identify the user, which is the problem of identity; that is, the application needs to know for which user to create the timer so that the user can retrieve information about it later.
- But the application should also authenticate that request.
- A password is "good enough" authentication for something that isn't hypersensitive.
- You have to consider the risks in sending the identification or authentication data over the Internet
- If the username and password are in "clear text", they can be read by anyone who is packet sniffing.

- **The two main cases here are as follows:**
- ▪ **Someone who is targeting a specific user and has access to that person's wired or (unencrypted) wireless network.**
- This attacker could read the details and use them (to create fake timers or get information about the user).
- ▪ **Someone who has access to one of the intermediate nodes.**
- This person won't be targeting a specific device but may be looking to see what unencrypted data passes by, to see what will be a tempting target.
- if a software password is compromised, a website can easily provide a way of changing that password.
- But while a computer has a monitor and keyboard to make that task easy, an Internet-connected device may not.
- So you would need a way to configure the device to change its password—for example, a web control panel hosted on the server or on the device itself.
- This solution is trickier (and does require the machine to have local storage to write the new password to).
- One obvious solution to the problem of sending cleartext passwords would be to encrypt the whole request, including the authentication details.
- For a web API, you can simply do this by targeting https:// instead of http://.
- Instead of username/password on HTTPs allowing a MAC address over HTTP for requests 1-4 that will be sent by the timer.
- Here's a revised table;

Task	Auth	Inputs	Outputs
1. Create a new timed task	MAC or User/Pass	Timer duration	Timer ID
2. Change duration of timed task	MAC or User/Pass	Timer ID, New timer duration	OK
3. Mark timer complete	MAC or User/Pass	Timer ID	OK
4. Cancel timer	MAC or User/Pass	Timer ID	OK

-

Q4)List and explain the standards to consider for implementing the API.
IMPLEMENTING THE API

- An API defines the messages that are sent from client to server and from server to client.
- You can send data in whatever format you want, but it is almost always better to use an existing standard because convenient libraries will exist for both client and server to produce and understand the required messages.
- Here are a few of the most common standards that you should consider:
- ▪ **Representational State Transfer (REST):**
- Access a set of web URLs using HTTP methods such as GET and POST.
- The result is often XML or JSON but can often depend on the HTTP content-type negotiation mechanisms.
- ▪ **JSON-RPC:**
- JavaScript Object Notation (JSON), is a way of formatting data so that it can be easily exchanged between different systems.
- Remote Procedure Call (RPC) is a term to describe ways of calling programming code which isn't on the same computer as the code you are writing.
- Access a single web URL like http://timer. roomofthings.com/api/, passing a JSON string such as
- {'method':'update', 'params':
- [{'timer-id':1234,'description':'Writing API chapter for book'}],
- 'id':12
- }.
- The return value would also be in JSON like {'result':'OK', 'error':null, 'id':12}.
- ▪ **XML-RPC:** This standard is just like JSON-RPC but uses XML instead of JSON.
- **Simple Object Access Protocol (SOAP):** This standard uses XML for transport like XML-RPC but provides additional layers of functionality, which may be useful for very complicated systems.

- In REST, you attach each resource to a URL and act on that.
- You can use different "methods" depending on whether you want to GET a resource, POST it onto the server in the first place, PUT an update to it, or DELETE it from the server.
- So the REST API will finally look like this:

Resource URL	Method	Auth	Parameters	Outputs
1. /timers	POST	MAC or Cookie	Timer duration	Timer ID
2. /timers/:id/duration	PUT	MAC or Cookie	Timer duration	OK
3. /timers/:id/complete	PUT	MAC or Cookie		OK
4. /timers/:id	DELETE	MAC or Cookie		OK
5. /timers/:id/description	PUT	Cookie	Description	OK
6. /timers	GET	Cookie		List of Timer IDs
7. /timers/:id	GET	Cookie		Info about Timer
8. /user/device	PUT	Cookie	MAC address	OK
9. /user/device	GET	Cookie		MAC address
10. /login	POST	User/Pass	User/Pass	Cookie + OK
11. /user	POST		User/Pass	Cookie + OK

Q6) Explain parameters you should consider when deciding on a platform for your web back end with the help of example (PERL).

Following are some of the parameters you should consider when deciding on a platform for your web back end:

- ▪ What do you already know (if you are planning to develop the code yourself)?
- ▪ What is the local/Internet recruiting market like (if you are planning to outsource)?
- ▪ Is the language thriving? Is it actively developed? Does it have a healthy community (or commercial support)? Is there a rich ecosystem of libraries available?
- Ex: Back-end Code in Perl
- Each handler declares the HTTP verb (GET, POST, PUT, DELETE) and the route that it handles.
- Parameters can be passed within the route, marked by a colon (:id, for example), or as part of the HTTP request.
- #1 Create a new timed task
- post "/timers.:format" => sub {
- my $user = require_user;
- # 'require_user' wants a session cookie # OR a valid MAC address.
- my $duration = param 'duration'
- or return status_bad_request('No duration passed');
- my $timer = schema->resultset('Timer')->create({
- user_id => $user->id,
- duration => $duration,
- status => 'O', # open
- });

- return status_created({
- status=>'ok',
- id => $timer->id,
- });
- };
-
- **USING CURL TO TEST**
- While you're developing the API, and afterwards, to test it and show it off, you need to have a way to interact with it.
- You could create the client to interface with it at the same time
- # the -F flag makes curl POST the request
- $ curl http://localhost:3000/user.json \
- -F user=hakim -F pass=secret
- {
- "status" : "ok",
- "id" : 2
- }
- curl simply makes an HTTP request and prints out the result to a terminal.

Q7) Explain the following factors with respect to API.

- **API Rate Limiting.**
- **Interaction via HTML**
- **Designing a Web Application for Humans**

GOING FURTHER

- **API Rate Limiting**
- If the service becomes popular, managing the number of connections to the site becomes critical.
- Setting a maximum number of calls per day or per hour or per minute might be useful.
- You could do this by setting a counter for each period that you want to limit.
- Then the authentication process could simply increment these counters and fail if the count is above a defined threshold.
- The counters could be reset to 0 in a scheduled cron job.
- software application can easily warn users that their usage limit has been exceeded and they should try later.

- **Interaction via HTML**
- The API currently serialises the output only in JSON, XML, and Text formats.
- You might also want to connect from a web browser.
- Note that not every Internet of Things product needs a browser application.
- Perhaps the API is all you need, with maybe a static home page containing some documentation about how to call the API.
- In the case of Clockodillo, however, we do want to have a set of web pages to interact with: Users should be able to look at their timers, assign descriptions, and so on.
- Drawbacks:
- Although web browsers do speak HTTP, they don't commonly communicate in all the methods that we've discussed.
- Web browsers don't commonly support PUT and DELETE.
- They support only GET and POST.
- As a solution we can "tunnel" the requests through a POST.

- There is a convention in Perl to use a field called x-tunneled-method, which you could implement like this:
- <form method="POST" action="/timer.html?x-tunneled-method=DELETE">
- <input type="hidden" name="id" value="1234">
- <input type="submit" value="Cancel this timer!">
- </form>

- **Designing a Web Application for Humans**

- For example, the above figure shows a static login page.
- This page is entirely for the convenience of a human.
- All the labels like "Your email address" and the help text like "Remember your password is case sensitive" are purely there to guide the user.
- The logo is there as a branding and visual look and feel for the site.

Q8)What are REAL-TIME REACTIONS? Explain two options such as "polling" and "Comet"

- To establish an HTTP request requires several round-trips to the server.
- There is the TCP "three-step handshake" consisting of a SYN (synchronise) request from the client, a SYN-ACK from the server to "acknowledge" the request, and finally an ACK from the client.
- Although this process can be near instantaneous, it could also take a noticeable amount of time.
- If you want to perform an action the instant that something happens on your board, you may have to factor in the connection time.
- If the server has to perform an action immediately, that "immediately" could be nearly a minute later, depending on the connection time.
- For example, with the task timer example, you might want to register the exact start time from when the user released the dial, but you would actually register that time plus the time of connection.
- We look at two options here: polling and the so-called "Comet" technologies.

POLLING

- If you want the device or another client to respond immediately, how do you do that?
- You don't know when the event you want to respond to will happen, so you can't make the request to coincide with the data becoming available.
- Consider these two cases:

- ▪ The WhereDial should start to turn to "Work" the moment that the user has checked into his office.
- ▪ The moment that the task timer starts, the client on the user's computer should respond, offering the opportunity to type a description of the task.
- The traditional way of handling this situation using HTTP API requests was to make requests at regular intervals. This is called *polling*.
- You might make a call every minute to check whether new data is available for you.
- However, this means that you can't start to respond until the poll returns.
- So this might mean a delay of (in this example) one minute plus the time to establish the HTTP connection.
- You could make this quicker, polling every 10 seconds, for example.
- But this would put load on the following:
- ▪ **The server:** If the device takes off, and there are thousands of devices, each of them polling regularly, you will have to scale up to that load.
- ▪ **The client:** This is especially important if, as per the earlier Arduino example, the microcontroller blocks during each connect!

- **COMET**
- *Comet* is an umbrella name for a set of technologies developed to get around the inefficiencies of polling.
- **Long Polling (Unidirectional)**
- starts off with the client making a polling request as usual.
- However, unlike a normal poll request, in which the server immediately responds with an answer, even if that answer is "nothing to report", the long poll waits until there is something to say.
- This means that the server must regularly send a keep-alive to the client to prevent the Internet of Things device or web page from concluding that the server has simply timed out.
- Long polling would be ideal for the case of WhereDial: the dial requests to know when the next change of a user's location will be.
- As soon as WhereDial receives the request, it moves the dial and issues a new long poll request.
- However, it isn't ideal for the task timer, with which you may want to send messages from the timer quickly, as well as receive them from the server.

- **Multipart XMLHttpRequest (MXHR) (Unidirectional)**
- Many browsers support a multipart/x-mixed-replace content type, which allows the server to send subsequent versions of a document via XHR.
- It is perfectly possible to simply long poll and create a new request on breaking the old one, but this means that you might miss a message while you're establishing the connection.
- In the example of WhereDial, this is unlikely; you're unlikely to change location first to Home and then to Work in quick succession.

- **HTML5 WebSockets (Bidirectional)**
- Traditionally, the API used to talk directly to the TCP layer is known as the *sockets API*.
- When the web community was looking to provide similar capabilities at the HTTP layer, they called the solution *WebSockets*.
- WebSockets have the benefit of being bidirectional.

- You can consider them like a full Unix socket handle that the client can write requests to and read responses from.
- This might well be the ideal technology for the task timer.
- After a socket is established, the timer can simply send information down it about tasks being started, modified, or cancelled, and can read information about changes made in software, too.

Q9) Explain how to implement Comet? And how scaling works with Comet.

- **Implementations**
- Let's look at support for these techniques on the three main platforms that you may need to consider for an Internet of Things application: the browser web app (if applicable), the microcontroller itself, and the server application.
- On the browser side, it is often possible to abstract the actual transport using a library which chooses which method to connect to the server.
- For example, it might use WebSockets if available; otherwise, it will fall back to MXHR or long polling.
- Many web servers have abstractions to support Comet techniques.
- Web::Hippie::Pipe provides a unified bidirectional abstraction for Perl web servers.
- There are also libraries for the microcontroller; however, they tend to support only one scheme.
- For example, several dedicated WebSockets libraries are available for Arduino.

- **Scaling**
- An important consideration is that all these Comet techniques require the client to have a long-term connection with the server.
- For a single client, this is trivial.
- But if there are many clients, the server has to maintain a connection with each of them.
- If you run a server with multiple threads or processes, you effectively have an instance of the server for each client.
- As each thread or process will consume system resources, such as memory, this doesn't scale to many clients.
- Instead, you might want to use an asynchronous web server, which looks at each client connection in turn and services it when there is new input or output.
- If the server can service each client quickly, this approach can scale up to tens of thousands of clients easily.

Q10)What OTHER PROTOCOLS are available to replace HTTP.

- **MQ TELEMETRY TRANSPORT**
- MQTT (http://mqtt.org) is a lightweight messaging protocol, designed specifically for scenarios where network bandwidth is limited.
- It was developed initially by IBM but has since been published as an open standard, and a number of implementations, both open and closed source, are available, together with libraries for many different languages.
- Rather than the client/server model of HTTP, MQTT uses a publish/ subscribe mechanism for exchanging messages via a *message broker*.
- Rather than send messages to a pre-defined set of recipients, senders publish messages to a specific *topic* on the message broker.
- Recipients subscribe to whichever topics interest them, and whenever a new message is published on that topic, the message broker delivers it to all interested recipients.

- This makes it much easier to do one-to-many messaging, and also breaks the tight coupling between the client and server that exists in HTTP.

- **EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL**
- Another messaging solution is the Extensible Messaging and Presence Protocol, or XMPP (http://xmpp.org).
- This is both a blessing and a curse: it is well understood and widely deployed, but because it wasn't designed explicitly for use in embedded applications, it uses XML to format the messages.
- This choice of XML makes the messaging relatively verbose ("using more words than necessary".) which could preclude (prevent from happening; make impossible) it as an option for RAM-constrained microcontrollers.

- **CONSTRAINED APPLICATION PROTOCOL**
- The Constrained Application Protocol (CoAP) is designed to solve the same classes of problems as HTTP but, for networks without TCP.
- There are proposals for running CoAP over UDP, SMS mobile phone messaging.
- CoAP draws many of its design features from HTTP and has a defined mechanism to proxies to allow mapping from one protocol to the other.