Day 4

Review async, AngularJS filters, Firebase intro

Table of Contents

Review - Async JS

What are the three parts of the JavaScript run-time and what are their functions?

Solution

- The heap, the stack and the queue.
- **Heap:** Dynamically allocates memory
- **Stack:**
  - Main execution engine
  - Functions are pushed to the stack, the stack executes them and then pops them
- **Queue:**
  - Takes care of asynchronous requests, how?
  1. Recognizes calls to the *async* API
  2. Takes **callback** from call, through the **event loop** and waits for response to arrive in a background thread
  3. When **callback** arrives, asks the stack if it can pass the callback to execute it with the response.

How come JavaScript seems like it executes sequentially? What happens to a function that “hugs” execution?

Solution

- When there are no *async* functions, JavaScript behaves like a normal programming language, the stack pushes and pops functions as you would expect.
- They block the main thread, rendering the whole application unresponsive while it executes.

- What are promises?
- What does *promisifying* mean?
- What are the three states of promises?

Solution

- A *promise* represents an eventual result from an asynchronous call

- *Promisifying* is the procedure used to convert asynchronous calls into promise semantics.

- *Pending, Fulfilled, Failed*

Opal promise creation example

```
function requestToServer(request, params)
{
    var deferred = $q.defer();
    var db = firebase.database();
    var key = db.set("request",{"name":request, parameters:params});
    db.ref("response"+"/"+key).once("value", function(snapshot){
        deferred.resolve(snapshot.value());
    }).catch(function(err){
        deferred.reject(err);
    });
    return deferred.promise;
}
```

How do we call a promise?

- Once we have *'promisified'* a function, how do we call it?
  - Use the **then/catch** promise semantics

```
// Suppose function okToGreet exists
function asyncGreet(name) {
    var deferred = $q.defer();

    setTimeout(function() {
        if (okToGreet(name)) {
            deferred.resolve('Hello, ' + name + '!');
        } else {
            deferred.reject('Greeting ' + name +
                ' is not allowed.');
        }
    }, 1000);
    return deferred.promise;
}
```

```
asyncGreet('Robin Hood')
    .then(function(greeting){
    alert('Success: ' + greeting);
    }).catch(function(error){
    alert('Failed: ' + reason);
});
```

Cases

- **Scenario 1:** One simple async request (Done)
- **Scenario 2:** Two or more simple requests that **depend** on one another
- **Scenario 3:** Two or more simple requests that **do not depend** on one another.
- Every other scenario is a combination of this three.

Scenario 1 - Example

```
fetchUrlContent(imageUrl)
    .then(function(content){

    }).catch(function(error){

    });
```

Scenario 2 - Example

```
// Assume getImages function exists, which fetches
// the images from conversations
requestToServer("GetConversations",{userId:1})
    .then(function(response){

        return getImages(response.data.conversations);
    }).then(function(conversationsWithImages){
        // Handle conversations
    })
    .catch(function(error){ alert(error); });
```

Scenario 3 - Example

```
function getImages(conversations){
    var promiseArray = [];
    for(var i = 0; i< conversations.length; i++)
    {
        promiseArray.push(fetchUrlContent(conversations[i].imageUrl));
    }
    return $q.all(promiseArray).then(function(images){
        images.forEach(function(image,index){
            conversations[index].image = image;
        });
        return images;
    });
}
```

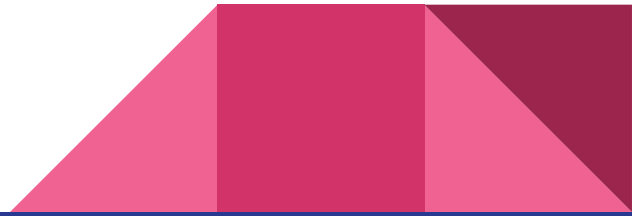
AngularJS Filters

What is a filter?

- **Filters**: Their role is to format the value of an expression for display to the user or manipulate
- They allow you to play around with your data and apply different **transformations**
- **Examples:**
  - Filtering a list, see filter list AngularJS
  - Sorting a list, see orderBy AngularJS
  - Applying a function map to an array, for instance, transform all the dates from text format to JS dates
    - See JS Dates

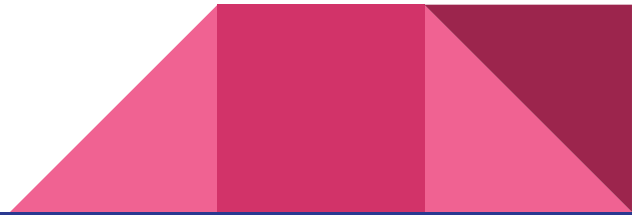
Types of filters

- Custom-made. (see custom filters)
- AngularJS Built-in filters (see built-in filters)
  - orderBy
  - Filter
  - Date
  - Lowercase
  - Uppercase
  - limitTo
  - json



Filter calls

- Filters can be called in either a controller, a template or a service.
  - In a **template** filters are used to filter, order lists with the ng-repeat attributes, or to change display formats
  - In a **controller** filters are used to apply custom sorting based on a specialized function
  - In a **service** filters are used to organize the model data normally while the app is in loading.



Filters in templates

- **Filter array of conversations by substring:**

```
<ons-list-item class="timeline-li" modifier="tappable"
               ng-repeat="conversation in vm.conversations | filter: vm.searchConversationString"
               ng-click="vm.goToConversation(conversation)">
```

- **Text formatting, reverse (applies to lists too)**

```
<div >
    <input ng-model="greeting" type="text"><br>
    No filter: {{greeting}}<br>
    Reverse: {{greeting|reverse}}<br>
    Reverse + uppercase: {{greeting|uppercase}}<br>
    Date: {{someDate | date: 'MMM d, y h:mm:ss a' }}<br>
</div>
```

Filters in controllers and services

- **Format:**

```
$filter("<name-filter>")(argument1, argument2);
```

- **Example:**

```
var conversations  = $filter("orderBy")(conversations, 'lastMessage',sortFunction);
function sortFunction(lastMessage1, lastMessage2)
{
    //Comparator function, returns boolean
}
```

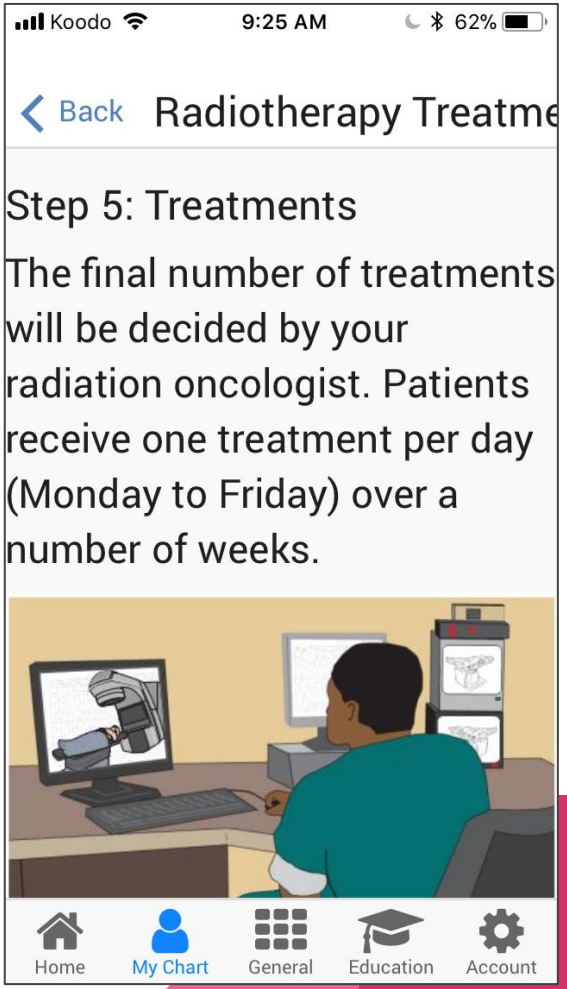
- **Note:** Must inject $filter dependency into controller/service

Custom filters

- Radiotherapy Treatment overfills the text, what can we do?

Solutions

- Change location of text from toolbar
- Recognize this and set a different font-size
- Use a filter that based on length returns a shortened version with … at the end.



Custom filters syntax

```
(function(){
    var module = angular.module('messaging-app');
    /**
     * @ngdoc filter
     * @name messaging-app.filter:ellipsis
     * @param {string} text Text to be processed
     * @param {string|number} maxLength Maximum length of text
     * @returns {Function}
     * @description If the text is larger than maxLength,
     *              shortened to maxLength and apply ellipsis
     */
    module.filter("ellipsis", EllipsisFilter);
    EllipsisFilter.$inject = [];
    function EllipsisFilter()
    {
        return function (text, maxLength)
        {
            maxLength = Number(maxLength);
            if(typeof text !== 'string' || isNaN(maxLength)) return text;
            if(text.length < maxLength)
            {
                var tempText = text.substring(0, maxLength);
                return tempText+"...";
            }
        }
    }
})();
```

Firebase

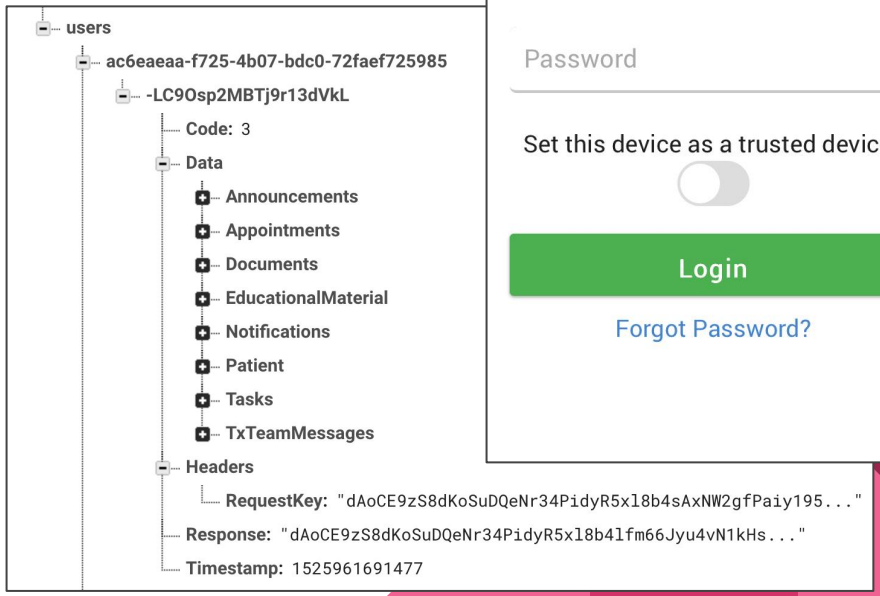
Firebase - Table of Contents

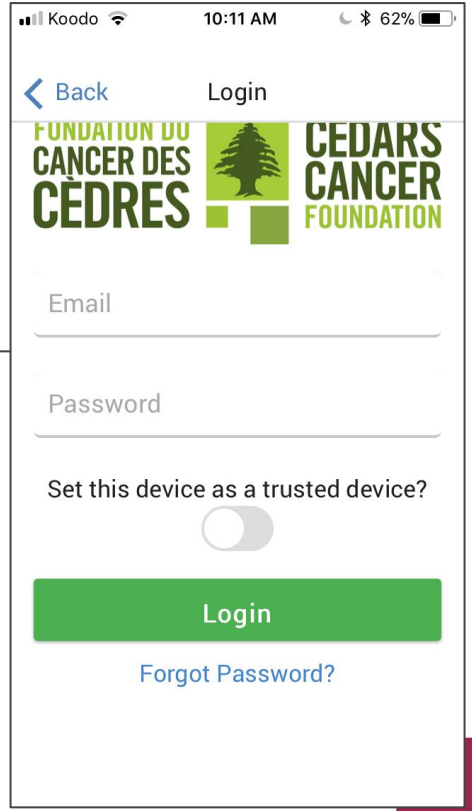
What is Firebase

- Firebase is **real-time cloud database.**
- Bought by Google in 2015
- Currently well integrated into Google developer services.
- Firebase is in charge of cloud-messaging now

Firebase in Opal

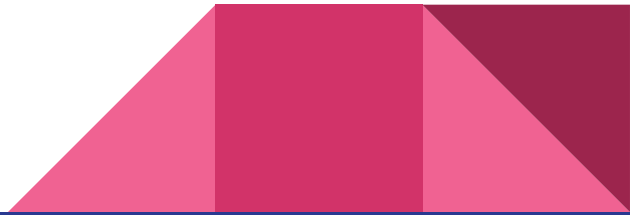
- Authentication
- Real-time database
- Cloud-messaging





Getting started with Firebase

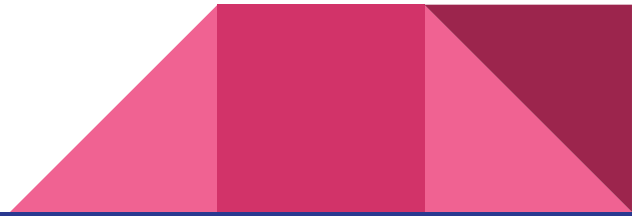
- Get an account at: https://firebase.google.com/docs/web/setup?authuser=0
- Add a new database
- Add Firebase to your web project
  - Head to Project Settings
  - Click on “Add App”
  - Copy-and-paste into your app



Adding Firebase

Resource: https://firebase.google.com/docs/web/setup?authuser=0

1. Head to Project Settings
2. Click on “Add App”
3. Copy-and-paste into your app



Adding Firebase

- Copy your Firebase credentials into your project

```
<script src="https://www.gstatic.com/firebasejs/5.0.1/firebase.js"></script>
<script>
  // Initialize Firebase
  // TODO: Replace with your project's customized code snippet
  var config = {
    apiKey: "<API_KEY>",
    authDomain: "<PROJECT_ID>.firebaseapp.com",
    databaseURL: "https://<DATABASE_NAME>.firebaseio.com",
    projectId: "<PROJECT_ID>",
    storageBucket: "<BUCKET>.appspot.com",
    messagingSenderId: "<SENDER_ID>",
  };
  firebase.initializeApp(config);
</script>
```

The AngularJS way

1. **bower install firebase --save**
2. Add files to index.html

```
<!-- Firebase App is always required and must be first -->
<script src="/__/firebase/5.0.1/firebase-app.js"></script>

<!-- Add additional services you want to use -->
<script src="/__/firebase/5.0.1/firebase-auth.js"></script>
<script src="/__/firebase/5.0.1/firebase-database.js"></script>
```

3. Add to ***./app/js/app.js*** the Firebase credentials right under the module definition.
4. Refresh the page and check for errors in console
5. If no errors, Firebase should be available globally in your workspace.
    - Type firebase in console to check this.

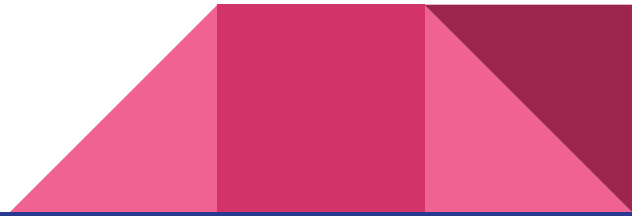
Firebase data structure

- Firebase is a **no-sql** database based on a JSON format.
- We have **key/value** pairs at each level
- We can add data to Firebase given a value based on a **key**
- We can listen to changes in an specific **key sub-tree.**
- Structure of Firebase DB is not **normalized.**

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

Firebase References

- In Firebase we use references which are paths in the database relative to the **root**.
- We can provide **filters**, and **orderings** to those references.
- We attach **listeners** to references
- We use the references to **set/update** the path.
- We can specify **query criteria** on this references.



Firebase References

```
var ref = firebase.database().ref();
var refMessages = ref.ref("messages");
var ref2Messages = firebase.database().ref("messages");
var refUsers = firebase.database().ref("users").orderByKey();
var refMessages = firebase.database().ref("messages")
            .orderByChild("lastMessageDate");
// Querying type example
ref.child("users").orderByChild("userId")
            .equalTo("54ca2c11d1afc1612871624a").limitToFirst(1);
```

Firebase reading

- Instantiates a **listener** at the given path
- There are different **types of listener**.
  - on
  - once
- There are different **events:**
  - child_added
  - value
  - child_moved
  - child_changed

```
var ref = firebase.database().ref("messages");
ref.<type-of-listener>(<type-of-event>), function(snapshot){
        if(snapshot.exists())
        {
                var val = snapshot.value();
                var key = snapshot.key();
                console.log(val, key);
        }
});
```

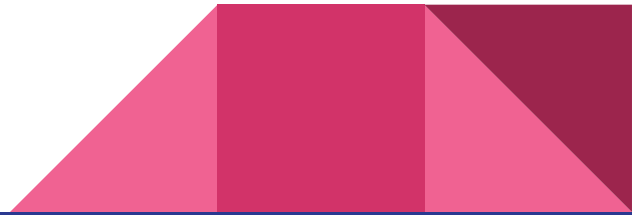
```
ref.on("child_changed",function(snapshot){
        if(snapshot.exists())
        {
                var val = snapshot.value();
                var key = snapshot.key();
                console.log(val, key);
        }
}
```

Firebase reading - event types

Event	Typical usage
child_added	Retrieve lists of items or listen for additions to a list of items. This event is triggered once for each existing child and then again every time a new child is added to the specified path. The listener is passed a snapshot containing the new child's data.
child_changed	Listen for changes to the items in a list. This event is triggered any time a child node is modified. This includes any modifications to descendants of the child node. The snapshot passed to the event listener contains the updated data for the child.
child_removed	Listen for items being removed from a list. This event is triggered when an immediate child is removed.The snapshot passed to the callback block contains the data for the removed child.
child_moved	Listen for changes to the order of items in an ordered list. **child_moved** events always follow the **child_changed** event that caused the item's order to change (based on your current order-by method).

Firebase reading - Types of Listeners

- **ref.on(“<event-type>”,function(){})**
  - Gives an update any time there is a change to path of the specified event type.
  - To disconnect the listener we use ref.off(“”)
- **ref.once(“<event-type>”,function(){})**
  - Only listens to the reference until there is a change.
  - Once it fetches the first time, it disconnects the reference.



Firebase writing

- Instantiates a **listener** at the given path
- Three types
  - Update
  - Push
  - Set

```
// Creates a random string as key and pushes request, then listens to that
// key for updates
var newPostRef = ref.child("request").push({"requesType":"GetConversatios"});
var key = newPostRef().key;
newPostRef.child("users"+'/'+key).once("value",function(){});
// Updates the sub-tree
refConversation.child(idConversation).update({"lastMessage":...});
// Overwrites the sub-tree
refConversation.child(idConversation).set({"lastMessage":...});
// Deletes the sub-tree
refConversation.child(idConversation).set(null);
```

Firebase writing

```
// Creates a random string as key and pushes request, then listens to that
// key for updates
var newPostRef = ref.child("request").push({"requesType":"GetConversatios"});
var key = newPostRef().key;
newPostRef.child("users"+'/'+key).once("value",function(){});
// Updates the sub-tree
refConversation.child(idConversation).update({"lastMessage":...});
// Overwrites the sub-tree
refConversation.child(idConversation).set({"lastMessage":...});
// Deletes the sub-tree
refConversation.child(idConversation).set(null);
```

- **Push:** Uses a randomly generated string as key, and writes to it the value provided.
- **Update:** Does not overwrite the entire sub-tree, simple updates the child that matches the provided key.
- **Set:** Overwrites sub-tree pointed to by reference completely.

Demo