

# Firestore: REST and Web API

DSCI 551

Wensheng Wu

# CRUD in REST


- CRUD
- C – create PUT/POST
- R – retrieve/read GET
- U – update PATCH
- D – delete DELETE
  
- startAt, endAt, equalTo
- limitToFirst=1, limitToLast=2
  
- Anomaly
  
- John, CA, west
- Mary, CA, west
  
- CA, west
  
- Denormalization
- BCNF

# Firebase

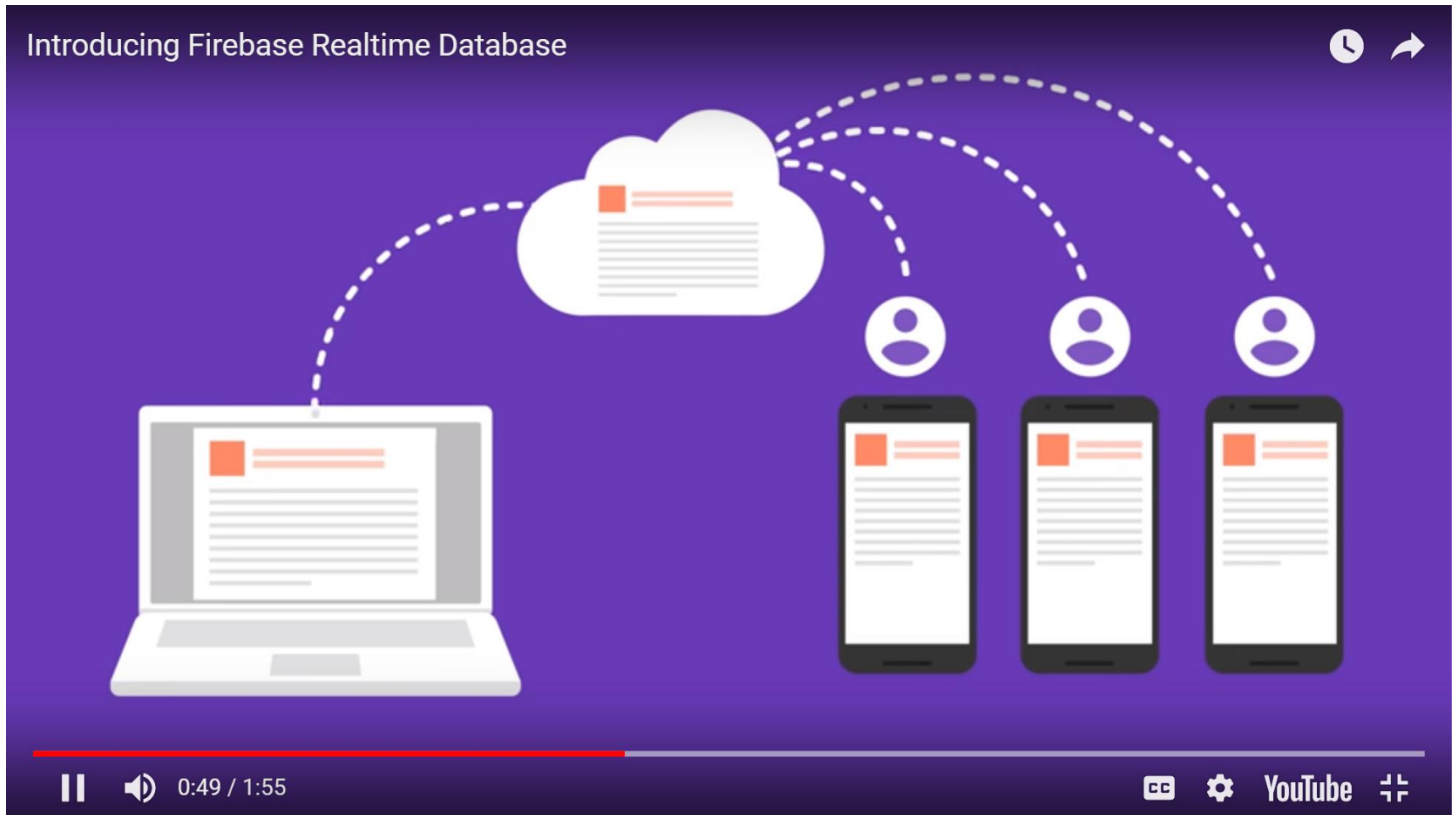
- A cloud-based platform to support web and mobile app development
- Used to be Envolv, a startup founded in 2011
  - For adding online chat functions into websites
- Later expanded into Firebase which was then acquired by Google in 2015

# Products

Also has Cloud Firestore in Beta

- Firebase (realtime) database
  - Manage JSON documents
  - Real-time syncing data between users and devices
- Firebase (cloud) storage
  - Store images, photos, videos
- Firebase (user) authentication
  - Support signin using Google, Facebook


# Firebase realtime database



**Develop**

**Authentication**

**Database**

**Storage**

**Hosting**

**Functions**

**ML Kit**

INF55x ▼

# Storage


**Files**


Rules

Usage

 gs://inf55x.appspot.com

☐ Name

☐  tropical-coast-10132,

 tropical-coast-1013... ✕

Name  
[tropical-coast-10132.jpg](#) 

Size  
554,404 bytes

Type  
image/jpeg

# Create a Firebase account

- You may use your Google account
- Go to Firebase console:
  - <https://console.firebase.google.com/>


# Click on "Add project"





Add app

App nickname

Demo 

App ID 


Choose this for web app

## Add Firebase to your app

Select a platform to get started



db5261

nfig 

tom of your <body> tag, but before you use any

you need from the SDKs you need  
from "<https://www.gstatic.com/firebasejs/>

## 2 Add Firebase SDK

☐ Use npm  ☒ Use a <script> tag 

Copy and paste these scripts into the bottom of your <body> tag, but before you use any Firebase services:

```
<script type="module">
  // Import the functions you need from the SDKs you need
  import { initializeApp } from "https://www.gstatic.com/firebasejs/9.0.0/firebase-app.js";
  // TODO: Add SDKs for Firebase products that you want to use
  // https://firebase.google.com/docs/web/setup#available-libraries

  // Your web app's Firebase configuration
  const firebaseConfig = {
    apiKey: "AIzaSyDSyDdbCB6m9JXqgCDvMMHbNqY0L8WixiI",
    authDomain: "inf55x.firebaseio.com",
    databaseURL: "https://inf55x.firebaseio.com",
    projectId: "inf55x",
    storageBucket: "inf55x.appspot.com",
    messagingSenderId: "163182188596",
    appId: "1:163182188596:web:21c837832289f1ccdb5261"
  };

  // Initialize Firebase
  const app = initializeApp(firebaseConfig);
</script>
```

Copy this &  
replace the  
firebaseConfig in  
the sample html  
file

Are you using npm and a bundler like webpack or Rollup? Check out the [modular SDK](#).

Learn more about Firebase for web: [Get Started](#), [Web SDK API Reference](#), [Samples](#)

Continue to console

# Create a firebase realtime db

**Firebase**

test ▾

Project Overview ⚙️

**Build**

- Authentication
- Firestore Database
- Realtime Database**
- Storage
- Hosting
- Functions
- Machine Learning

Extensions

### Set up database

1 Database options — 2 Security rules

Your location setting is where your Realtime Database data will be stored.

Realtime Database location


United States (us-central1) ▾

Cancel **Next**

# Realtime database

INF55x ▾

## Database

 Realtime Database ▾

[Data](#) [Rules](#) [Backups](#) [Usage](#)

<https://inf55x.firebaseio.com/>

inf55x

- users
  - 100
    - name: "john"
    - weather: "sunny"

# Choose test mode

## Set up database



1 Database options — 2 Security rules

Once you have defined your data structure you will have to write rules to secure your data.

[Learn more](#)

☐ Start in **locked mode**

Your data is private by default. Client read/write access will only be granted as specified by your security rules.

☒ Start in **test mode**

Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.

```
{
  "rules": {
    ".read": "now < 1632639600000", // 2021-9-26
    ".write": "now < 1632639600000", // 2021-9-26
  }
}
```



The default security rules for test mode allow anyone with your database reference to view, edit and delete all data in your database for the next 30 days

Cancel


Enable


## Firestore pricing plans



### Spark

Free \$0/month

 Usage quotas for Database, Firestore, Storage, Functions, Phone Auth, Hosting, and Test Lab

 Ability to extend your project with Google Cloud Platform


✓ **Included in all plans**  
Analytics, Notifications, Crash Reporting, support, and more


[See full plan details](#) 

Current Plan

### Blaze

Pay as you go

 Includes free usage, calculated daily. After, pay only for what your project uses.

 Ability to extend your project with Google Cloud Platform

✓ **Included in all plans**  
Analytics, Notifications, Crash Reporting, support, and more

[See full plan details](#) 

Select plan

# May use this (but we expect you to know the curl way)

The screenshot shows the REST client interface with the following details:

- Browser Address Bar:** `restninja.io`
- Page Header:** REST ninja logo, a share icon, a "Share Request" button, and tabs for "settings", "server", and "ajax".
- Request Bar:** Method: GET, Retries: 1, URL: `https://test-f3333-default-rtdb.firebaseio.com/students.json`, and a "Send" button.
- Response Panel:** Tabs for "headers", "auth", "preview", "raw", and "headers" (13). The "preview" tab is active, showing a JSON array of three student objects. A download icon and status bar (200, 785 ms, 108 bytes) are also present.
- JSON Response (Preview):**


```
1 {  
2   "100": {  
3     "age": 20,  
4     "name": "john"  
5   },  
6   "200": {  
7     "age": 23,  
8     "gender": "M",  
9     "name": "bill"  
10  },  
11  "300": {  
12    "age": 21,  
13    "name": "david"  
14  }  
15 }
```
- Left Sidebar:** Includes a "classic" dropdown, an eye icon, a trash icon, and a "Value" input field.

# JSON (Javascript Object Notation)

- Light-weight data exchange format
  - Much simpler than XML
  - Language-independent
  - Inspired by syntax of JavaScript object literals
- Some differences from JavaScript objects, e.g.,
  - String in JSON must be double-quoted
  - Ok to single-quote in JavaScript (& Python)



# Syntax of JSON

- value =  
string | number | object | array | **true** | **false** | **null**  

- object = { } | { members }
  - members = pair | pair, members
  - pair = string : value
- array = [ ] | [ elements ]
  - elements = value | value, elements

# Valid JSON or not?

- []
- {}
- {[]}
- [{}]
- {"name": john}
- {name: "john"}
- {"name": 25}
- "name"
- 25
- {25}
- [25]


# JSON is case-sensitive

- Valid or not?
  - True
  - true
  - TRUE
  - Null
  - false


# Example JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Value is an object

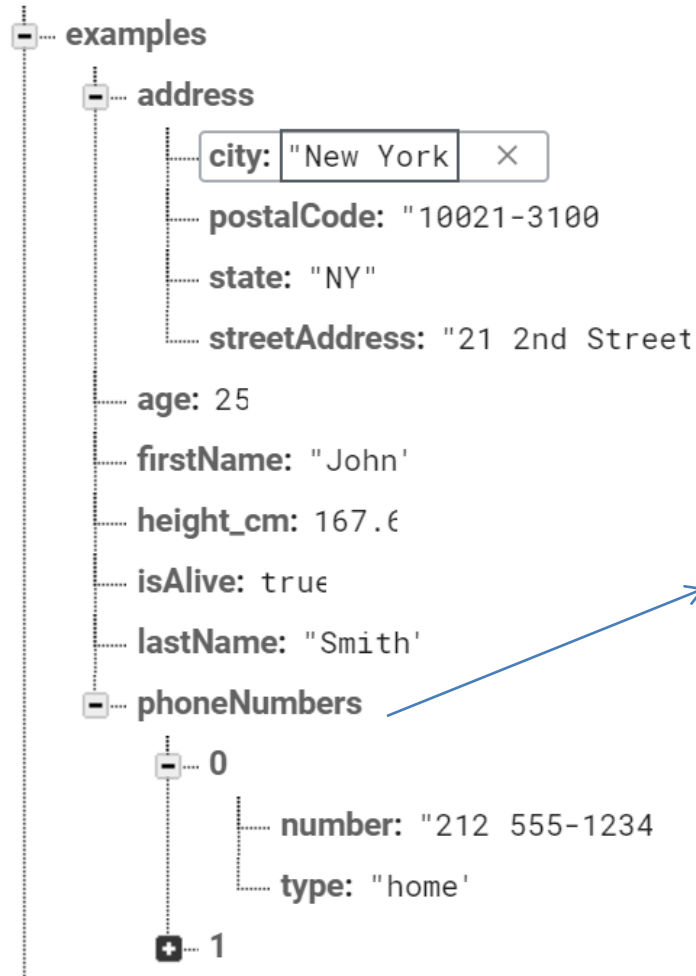


Value is an array



# Stored in Firebase

inf551-1b578



Note: array stored as an object  
Key = index of element in array

# Check syntax of JSON

- JSON validator
  - <http://jsonlint.com/>

# Roadmap

- **Firestore REST API**
- **Firestore Javascript API**
  - Useful for your project



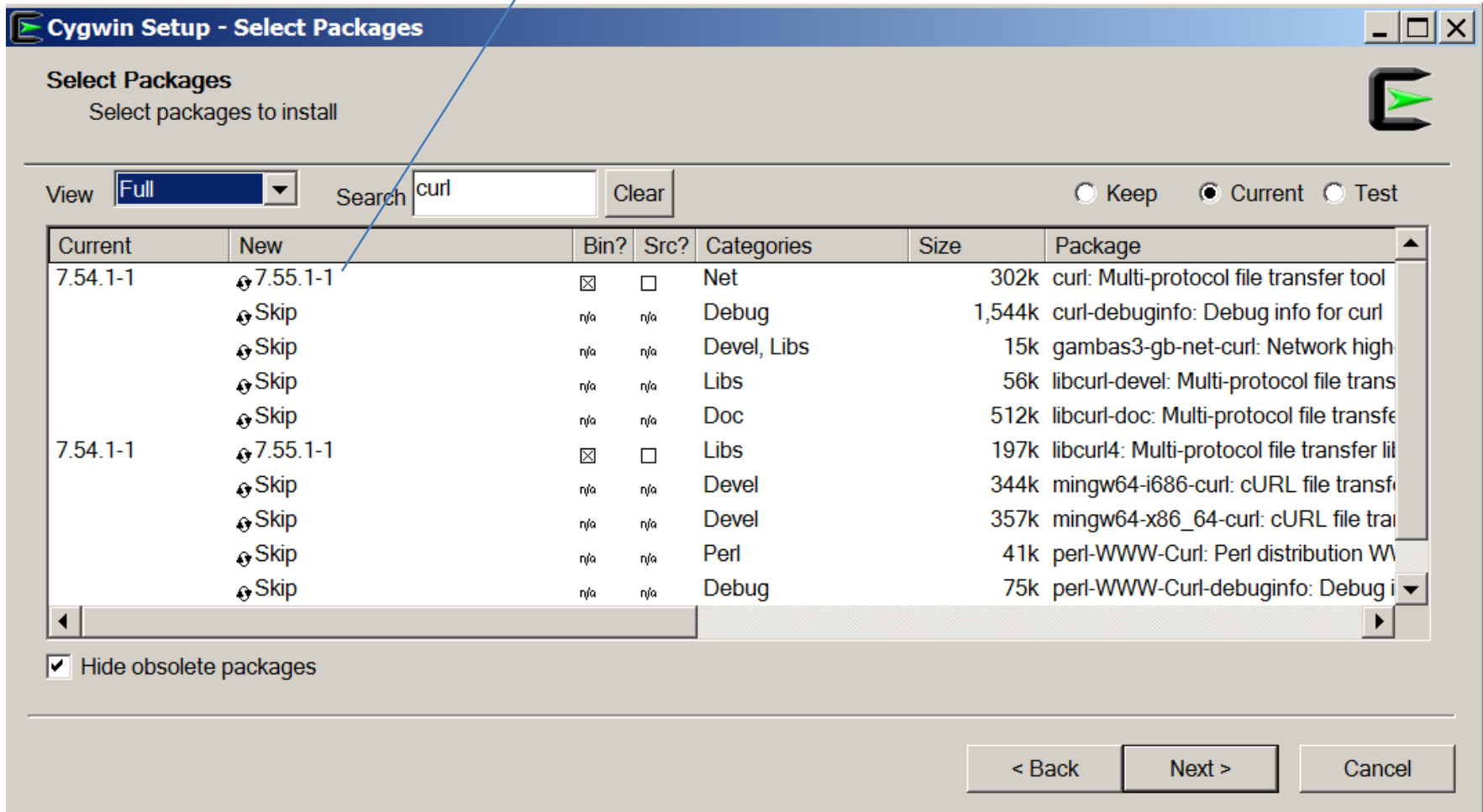
# curl

- Command line tool for data transfer
- Download from here (has Windows & Mac OS versions):
  - <https://curl.haxx.se/download.html>
- You may easily grab a copy of curl in Cygwin (see next slide)



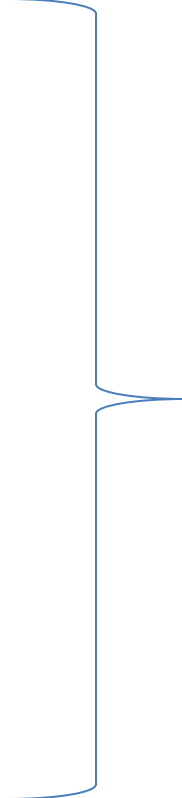
# Install curl in Cygwin

Select to install this one



# Firebase REST API

- PUT & POST (C in CRUD)
- GET (R)
- PATCH (U)
- DELETE (D)



All request commands  
are case sensitive (all uppercases)

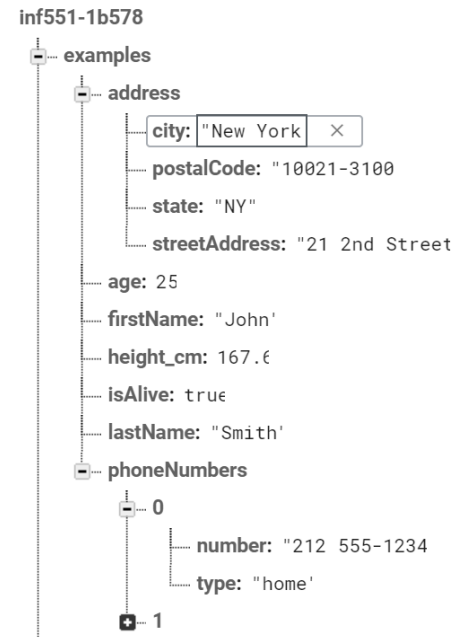
# GET

- `curl 'https://inf551-1b578.firebaseio.com/weather.json'`
- Or
  - `curl -X GET 'https://inf551-1b578.firebaseio.com/weather.json'`

# Another example

- `curl -X GET 'https://inf551-1b578.firebaseio.com/examples/phoneNumbers/0.json'`
  - `{"number": "212 555-1234", "type": "home"}`

Note: refer to array element by index



# PUT

- `curl -X PUT 'https://inf551-1b578.firebaseio.com/weather.json' -d '"hot"'`
  - `"hot"`
- PUT: write a given value (e.g., `"hot"`) to the specify node (e.g., `"weather"`)
  - Overwrite if node already has value

# PUT

- `curl -X PUT 'https://inf551-1b578.firebaseio.com/users/100.json' -d '{"name": "john"}'`
- This will add a new node "users" (assuming it does not exist yet) and a child of this node with key "100" and content: `{"name": "john"}`

# Example

- Is the previous command the same as this?
  - `curl -X PUT -d '{"100": {"name": "John"}}'`  
<https://inf551-1b578.firebaseio.com/users.json>



Note we now write to the "users" node

- Can you think of a situation where two commands give different results?

# POST

- `curl -X POST -d '{"name": "John"}'`  
<https://inf551-1b578.firebaseio.com/users.json>
- Note post automatically generates a new key & then stores the value for the new key
  - In contrast, PUT will simply overwrite the value



# PATCH

NOTE: patch expects an JSON object in the value

- `curl -X PATCH -d '{"name": "John Smith", "age": 25}' 'https://inf551-1b578.firebaseio.com/users/100.json'`
- PATCH performs the update if value already exists (e.g., name) ; otherwise, it inserts the new value (e.g., age)
  - So... an upsert

# DELETE

- `curl -X DELETE 'https://inf551-1b578.firebaseio.com/users/100.json'`
- What does this do?
  - `curl -X DELETE 'https://inf551-1b578.firebaseio.com/users.json'`


# Query: filtering by key

- `curl 'https://inf551-1b578.firebaseio.com/users.json?orderBy="$key"&equalTo="200"'`

→ Must be a string. Why?

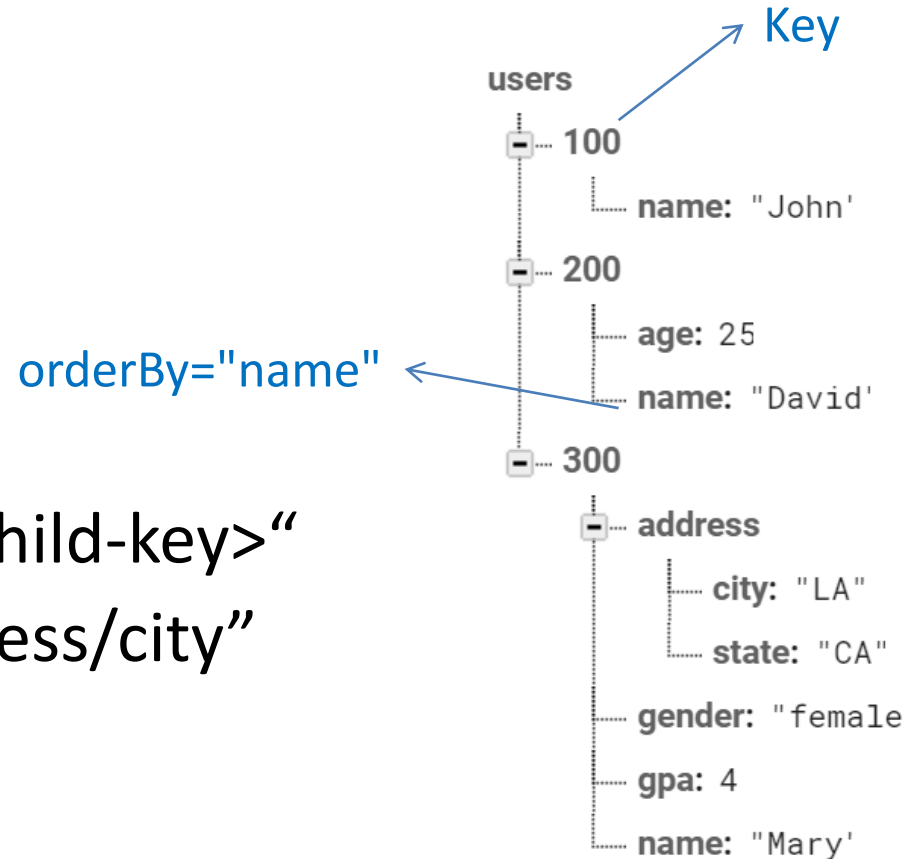
- This returns:
  - `{"200":{"age":25,"name":"David"}}`

# Another example

- `curl 'https://inf551-1b578.firebaseio.com/users.json?orderBy="$key"&startAt="200"'`  
 → Users with keys  $\geq$  "200"
- This returns:
  - `{"200":{"age":25,"name":"David"},"300":{"gender":"female","gpa":4.0,"name":"Mary"}}`

# Ways of filtering data

- By key:
  - orderBy="\$key"
- By child key:
  - orderBy="<path-to-child-key>"
  - E.g., orderBy = "address/city"
- By value:
  - orderBy="\$value"



# Parameters

- startAt
- endAt
- equalTo
- limitToFirst
- limitToLast

# Example: filtering by child key

- `curl 'https://inf551-1b578.firebaseio.com/users.json?orderBy="name"&limitToFirst=1&print=pretty'`
- What will this return?

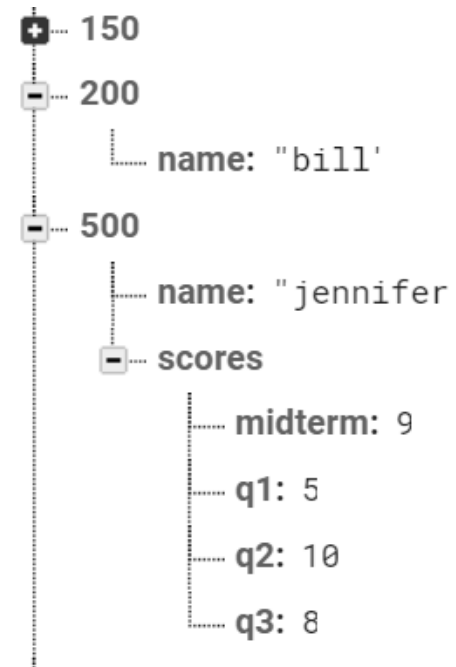
# Example for orderBy="\$value"

- `curl -X PUT 'https://inf551-1b578.firebaseio.com/users/500.json' -d '{"name": "jennifer", "scores": {"q1": 5, "q2": 10, "q3": 8, "midterm": 9}}'`



# Example: filtering by value

- `curl 'https://inf551-1b578.firebaseio.com/users/500/scores.json?orderBy="$value"&limitToFirst=1&print=pretty'`
- What will this return?



# Creating index for value/child key

- Specified in database rules:
  - <https://firebase.google.com/docs/database/security/indexing-data>
- Only required for REST API
- No need to index key
  - Done automatically by Firebase

```
{
  "rules": {
    ".read": true,
    ".write": true,
    "users": {
      ".indexOn": ["name", "age"],
      "500": {
        "scores": { ".indexOn": ".value" }}
    }
  }
}
```

# Ordering

## orderBy

When using `orderBy` with the name of a child key, data that contains the specified child key will be ordered as follows:

1. Children with a `null` value for the specified child key come first.
2. Children with a value of `false` for the specified child key come next. If multiple children have a value of `false`, they are sorted *lexicographically* by key.
3. Children with a value of `true` for the specified child key come next. If multiple children have a value of `true`, they are sorted lexicographically by key.
4. Children with a numeric value come next, sorted in ascending order. If multiple children have the same numerical value for the specified child node, they are sorted by key.
5. Strings come after numbers, and are sorted lexicographically in ascending order. If multiple children have the same value for the specified child node, they are ordered lexicographically by key.
6. Objects come last, and sorted lexicographically by key in ascending order.

The filtered results are returned unordered. If the order of your data is important you should sort the results in your application after they are returned from Firebase.

# Ordering

`orderBy="$key"`

When using the `orderBy="$key"` parameter to sort your data, data will be returned in ascending order by key as follows. Keep in mind that keys can only be strings.

1. Children with a key that can be parsed as a 32-bit integer come first, sorted in ascending order.
2. Children with a string value as their key come next, sorted lexicographically in ascending order.

`orderBy="$value"`

When using the `orderBy="$value"` parameter to sort your data, children will be ordered by their value. The ordering criteria is the same as data ordered by a child key, except the value of the node is used instead of the value of a specified child key.

# Watch out...

- <https://firebase.google.com/docs/database/rest/retrieve-data>



**Filtered data is returned unordered:** When using the REST API, the filtered results are returned in an undefined order since JSON interpreters don't enforce any ordering. If the order of your data is important you should sort the results in your application after they are returned from Firebase.

# Using REST in Python

- import requests
  - May need to "pip install requests" first
- url = 'https://inf551-1b578.firebaseio.com/users.json'
- response = requests.get(url)
- response.json()
  - {u'200': {u'age': 25, u'name': u'David'},...

# Writing

- `url1 = 'https://inf551-1b578.firebaseio.com/users/888.json'`
- `data = '{"name": "jimmy", "gender": "male"}'`
- `response = requests.put(url1, data)`

# Update, delete & post

- Updating
  - `requests.patch(url, data)`
- Deleting
  - `requests.delete(url)`
- Posting
  - `Requests.post(url, data)`



# Pretty printing

- `import json`
- `print json.dumps(response.json(), indent=4)`

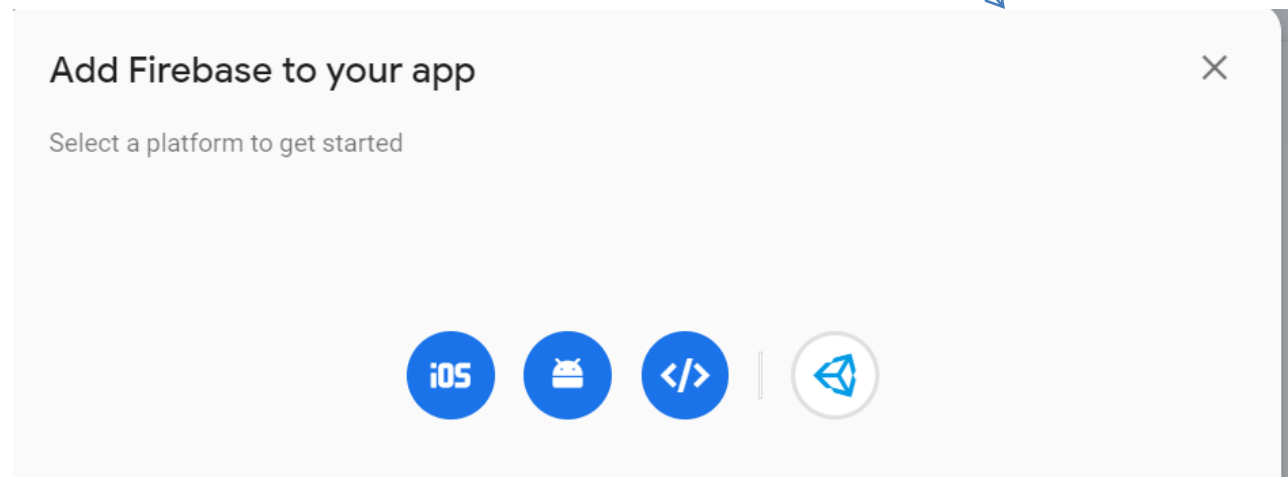
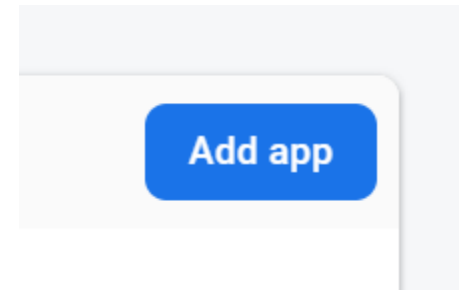
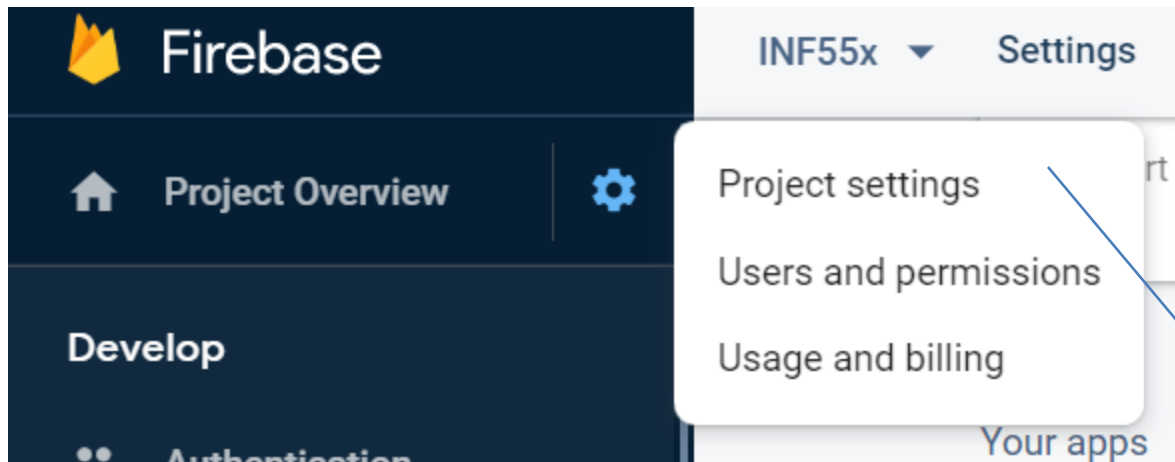


```
{  
  "200": {  
    "age": 25,  
    "name": "David"  
  },  
  ...  
}
```

# Roadmap

- Firebase REST API
- **Firebase Javascript/Web API**
  - Useful for your project





# Demo html page

```
<html>
<head><title>Test Firebase</title></head>
<body>
```

It is `<span id="value"></span>` today!!

```
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/7.7.0/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/7.7.0/firebase-database.js"></script>
```

```
<script>
  // Your web app's Firebase configuration
  var firebaseConfig = {
    apiKey: "AIzaSyDSyDdbCB6m9JXqgCDvMMHbNqY0L8WixiI",
    authDomain: "inf55x.firebaseio.com",
    databaseURL: "https://inf55x.firebaseio.com",
    projectId: "inf55x",
    storageBucket: "inf55x.appspot.com",
    messagingSenderId: "163182188596",
    appId: "1:163182188596:web:ca7ccfc2221ef4f4db5261"
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);

  var value = document.getElementById("value");
  var dbRef = firebase.database().ref().child("weather");

  // query example: a single value
  dbRef.on('value', function(snapshot) {
    console.log("weather value" + ": " + JSON.stringify(snapshot.val()));
    value.innerText = snapshot.val();
  });
```

`val()` returns a Javascript object  
representing content of snapshot

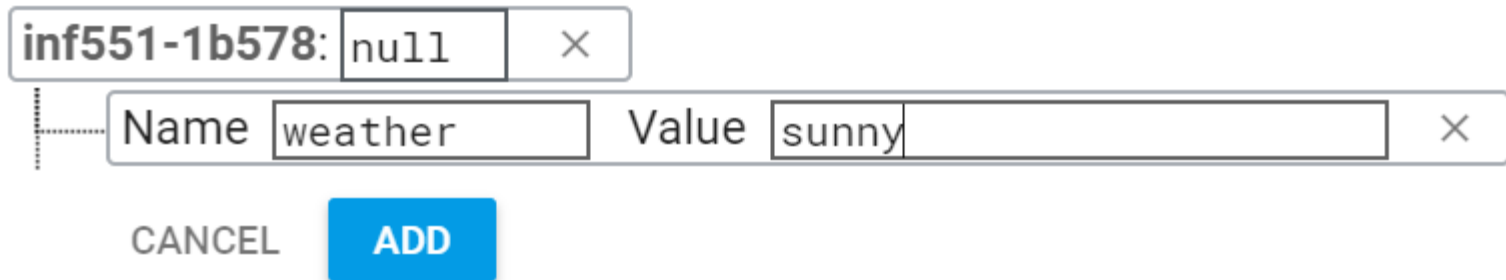


# Database reference

- `firebase.database()` returns a reference to the firebase database as specified by `"firebaseConfig"`
- `ref()`: returns a reference to the root node of the database
- `ref("weather")` returns a reference to the `"weather"` child of the root
  - same as `ref().child("weather")`

# Modify the data in database

- Observe the data automatically changed in the browser




The screenshot shows a database browser interface. At the top, there is a header bar with the text "inf551-1b578:" followed by a "null" value and a close button (X). Below this, there is a form with two main sections: "Name" and "Value". The "Name" section has a text input field containing "weather". The "Value" section has a text input field containing "sunny" and a close button (X). Below the form, there are two buttons: "CANCEL" and "ADD".

| Name    | Value |
|---------|-------|
| weather | sunny |

# Write data using set()

- ```
function writeUserData(userId, name, email) {  
  firebase.database().ref("users/" + userId).set({  
    name: name,  
    email: email  
  });  
}
```



Setting/overwriting the data of user 123
- ```
writeUserData("123", "John", "john@usc.edu");
```

# Write data using push() and set()

- `firebase.database().ref("users").push().set({name: "John", email: "john@usc.edu"});`
- `push()` will automatically generate a key
  - In this case, id for the new user
- Which REST command is this similar to?



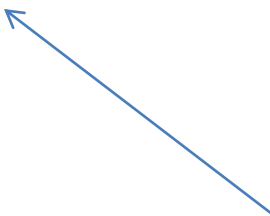
# Update data

- ```
function updateUserData(userId, phone) {  
  firebase.database().ref("users/"+userId).update({  
    phone: phone  
  });  
}
```

Note this does not remove other data of user 123  
What if you replace "update" with "set"?

- ```
updateUserData("123", "(626)123-0000");
```

# Retrieve a list of values

- `userRef = firebase.database().ref("users");`
  - `userRef.on("value", function(snapshot) {  
 snapshot.forEach(function(child) {  
 console.log(child.key + ": " + child.val());  
 });  
});`
- 

Press Ctrl+Shift+J in Chrome for console window

# Listening to child events instead

- `userRef.on("value", function(snapshot) {...`
  - Will retrieve a list of values in the path specified by `userRef`
  - Not efficient, since entire list will be retrieved whenever changes occur
- `userRef.on("child_added", function(...)) {...`
  - Firebase will callback for every existing child and new child added to the path `userRef`
  - Other events: `child_changed`, `child_removed`

# Filtering data

- queryRef =  
 firebase.database().ref("users").orderByChild(  
 "name").equalTo("David");
- queryRef.on("value", function(snapshot) {  
 snapshot.forEach(function(child) {  
 console.log(child.key + ": " + child.val());  
 });  
});

# Filtering data

- It also supports:
  - `orderByKey()`
  - `orderByValue()`

# orderByValue() example

```
queryRef = firebase.database().ref("users/500/scores")
    .orderByValue();
queryRef.on("value", function(snapshot) {
    snapshot.forEach(function(child) {
        console.log(child.key + ": " + child.val());
    });
});
```



---

q1: 5

---

q3: 8

---

midterm: 9

---

q2: 10

---

# Resources

- Add Firebase to your JavaScript Project
  - <https://firebase.google.com/docs/web/setup>
- Getting Started with Firebase on the Web
  - [https://www.youtube.com/watch?v=k1D0\\_wFlXgo&feature=youtu.be](https://www.youtube.com/watch?v=k1D0_wFlXgo&feature=youtu.be)
- Realtime Database: Installation & Setup in JavaScript, Read & Write Data ...
  - <https://firebase.google.com/docs/database/web/start>

# Resources

- Firebase REST API
  - <https://firebase.google.com/docs/reference/rest/database/>
- Requests for Python
  - <http://docs.python-requests.org/en/master/user/quickstart/#make-a-request>



# Resources

- Firebase Firestore Tutorial
  - <https://www.youtube.com/watch?v=4d-gIPGzmK4>
  - On how to develop a web app using Firestore