IN4MATX 133: User Interface Software

Lecture 19:
Databases and Local Storage

Professor Daniel A. Epstein
TA Lucas de Melo Silva
TA Jong Ho Lee

Today's goals

By the end of today, you should be able to…

- Differentiate relational from non-relational databases

- Explain the advantages of each style of database

- Use Firebase to implement a non-relational database

Today is a crash course in databases CS 122A and 122B provide substantially more depth

Data storage

- What happens when we refresh the A4 sleep tracking app?

  - We lose all of the data we logged

- This is obviously not ideal

  - We have to tell the browser, app, etc. to store it

Data storage

- Data can be stored locally on a device

  - Android and iOS allow apps to store some data

  - Ionic Native provides (good) libraries for using local storage

Local Storage

- In Ionic, can store key-value pairs

  - Keys must be strings, values can be any type

- This is actually a non-relational database!

  - More on this in a few slides

https://ionicframework.com/docs/building/storage#ionic-storage

Local Storage

```
ionic cordova plugin add cordova-sqlite-storage

npm install --save @ionic/storage
```
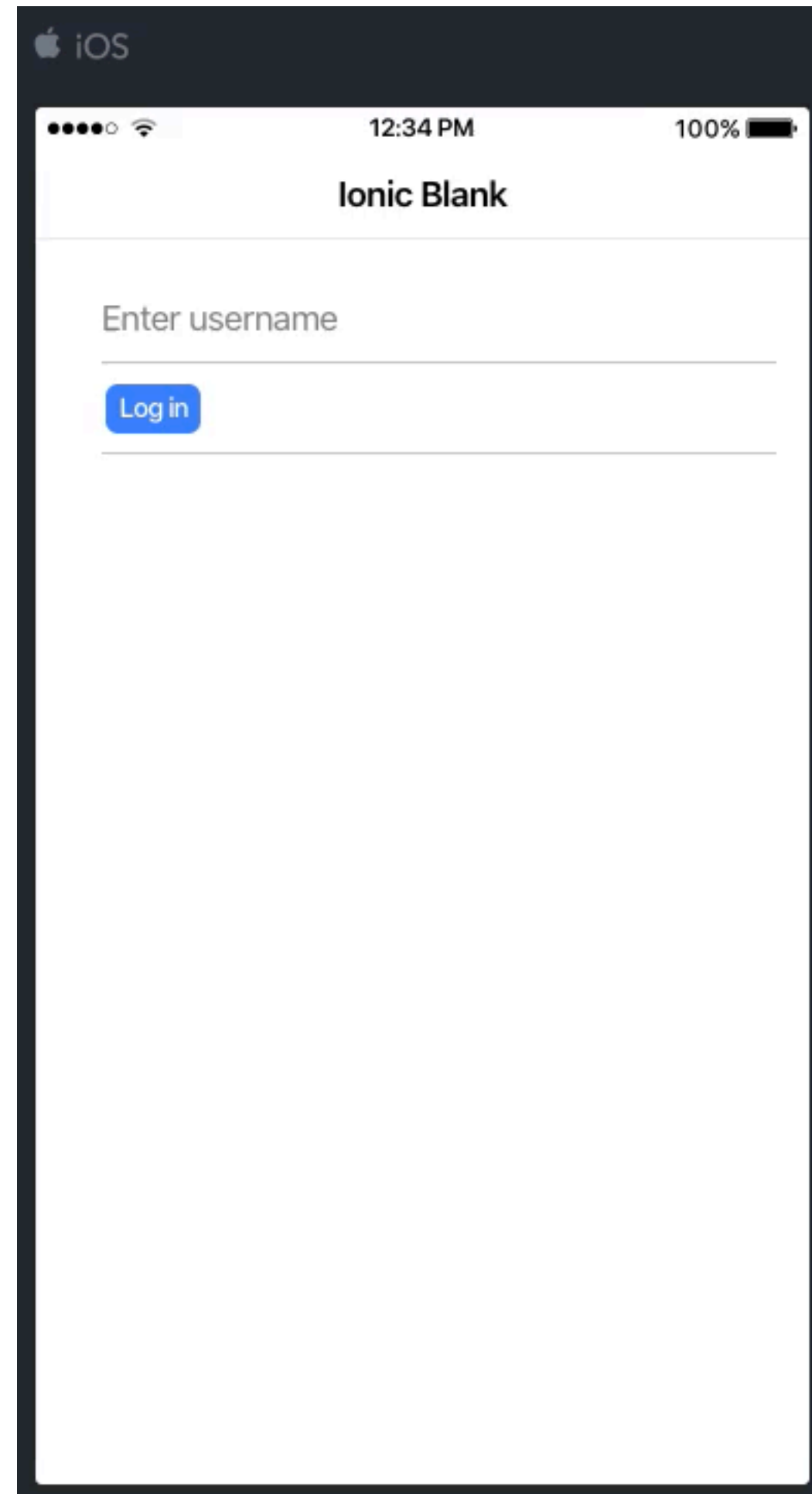
● Don't forget to add it to your module and inject it!

```
storage.set('name', 'Max');

  // Or to get a key/value pair
  storage.get('age').then((val) => {
    console.log('Your age is', val);
  });
```

https://ionicframework.com/docs/building/storage#ionic-storage

Local Storage

If we can store data on devices, why do we need databases?

Databases

- Provide reliability

  - You can get your data back if your phone dies or you get a new phone

- Provide cross-device support

  - Allow you to see and modify the same data across a phone and a desktop, for example

Databases

- Are more than files stored in the cloud

  - Can be "queried" efficiently to get subsets of data

- Two main approaches to making databases

  - Relational databases: MySQL, Postgres

  - Non-relational databases: MongoDB, Firebase

- Transaction: any add/delete/update/etc. made to a database

Databases

Relational databases

- Everything is organized into tables

- Tables contain columns with predefined names and data types

- Tables "relate" to one another by having overlapping or similar columns

  - Minimizes redundancy and keeps order

- Every data entry is a row of a table

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/
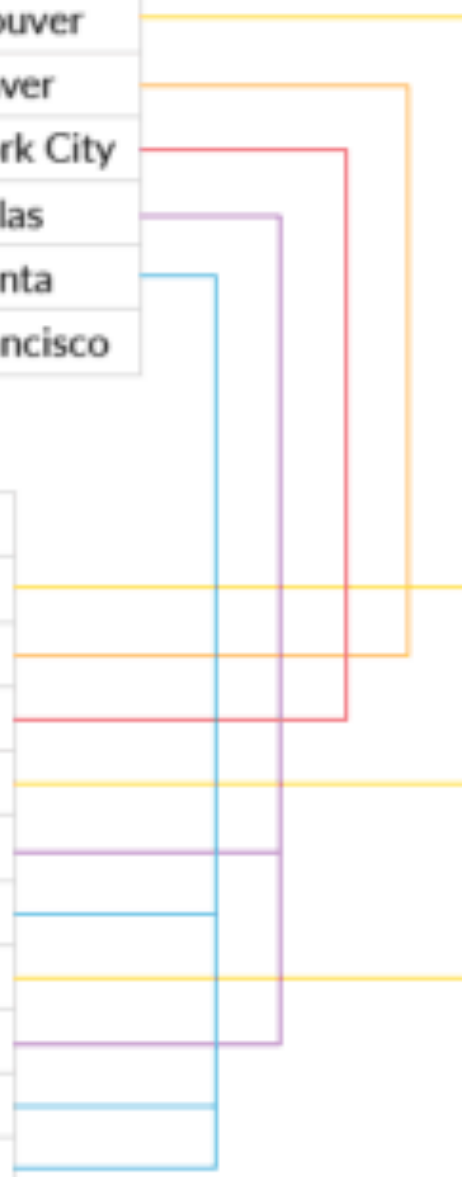https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Relational databases

Relational

Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/
https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Relational databases

```sql
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE,
    status TINYINT NOT NULL,
    priority TINYINT NOT NULL,
    description TEXT,
    PRIMARY KEY (task_id)
)  ENGINE=INNODB;
```

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/
https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Non-relational databases

- Everything is organized into objects

- There are no restrictions on how objects are structured

- Every data entry is an object, or "document"

  - Documents may be structured differently from one another

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/
https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Non-relational databases

```
MongoDB
Document

{
    first_name: 'Dexter',
    last_name: 'Lanas'
    city: 'Vancouver'
    location: [45.123,47.232],
    phones: [
        { phone_number: '111-111-1111',
          type: mobile,
          person_id: 1, ... },
        { phone_number: '444-444-4444',
          type: home,
          person_id: 1, ... },
        { phone_number: '777-777-7777',
          type: office,
          person_id: 1, ... },
    ]
}
```

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/
https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Non-relational databases

- There is no well-defined enforced structure

- That said, flatter structures are generally better

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/
https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Non-relational databases

```
{
  // This is a poorly nested data architecture, because iterating the children
  // of the "chats" node to get a list of conversation titles requires
  // potentially downloading hundreds of megabytes of messages
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "messages": {
        "m1": { "sender": "ghopper", "message": "Relay malfunction found. Cause: moth." },
        "m2": { ... },
        // a very long list of messages
      }
    },
    "two": { ... }
  }
}
```

https://firebase.google.com/docs/database/ios/structure-data

Databases

Non-relational databases

```
{
  // Chats contains only meta info about each conversation stored under the chats's unique ID
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "lastMessage": "ghopper: Relay malfunction found. Cause: moth."
    },
    "two": { ... }
  },
  // Messages are separate from data we may want to iterate quickly but still easily paginated and queried,
  // and organized by chat conversation ID
  "messages": {
    "one": {
      "m1": {
        "name": "eclarke",
        "message": "The relay seems to be malfunctioning."
      },
      "m2": { ... }
    },
    "two": { ... }
  }
}
```

https://firebase.google.com/docs/database/ios/structure-data

Question

Which database structure will be best for retrieving all first names?

A The relational database

B The non-relational database

C They will be about the same
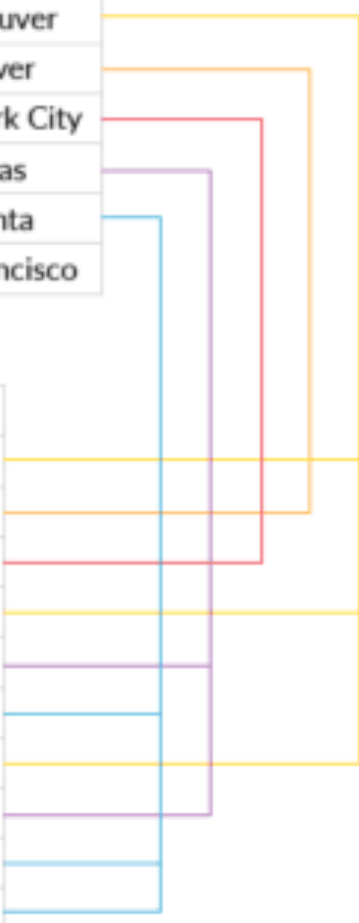
D I'm not sure

E [space intentionally left blank]

Relational

Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational

```
{
    first_name: 'Dexter',
    last_name: 'Lanas'
    city: 'Vancouver'
    location: [45.123,47.232],
    phones: [
        { phone_number: '111-111-1111',
          type: mobile,
          person_id: 1, ... },
        { phone_number: '444-444-4444',
          type: home,
          person_id: 1, ... },
        { phone_number: '777-777-7777',
          type: office,
          person_id: 1, ... },
    ]
}
```

Question 📱

Which database structure will be best for retrieving **all first names**?

A The relational database

B The non-relational database

C They will be about the same

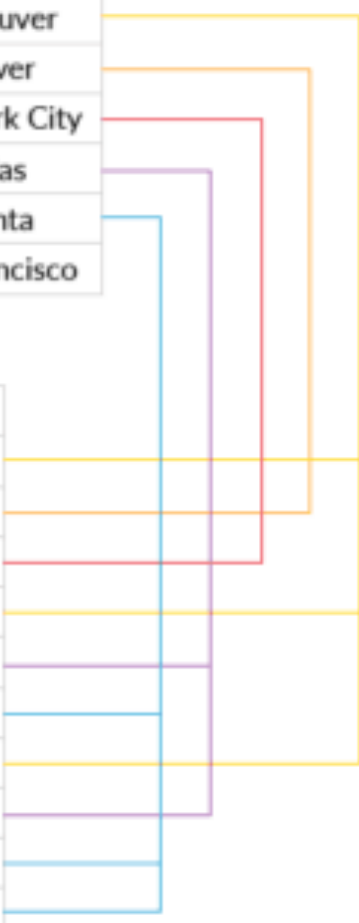**D** I'm not sure

E [space intentionally left blank]

Relational

Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational

```
{
    first_name: 'Dexter',
    last_name: 'Lanas'
    city: 'Vancouver'
    location: [45.123,47.232],
    phones: [
        { phone_number: '111-111-1111',
          type: mobile,
          person_id: 1, ... },
        { phone_number: '444-444-4444',
          type: home,
          person_id: 1, ... },
        { phone_number: '777-777-7777',
          type: office,
          person_id: 1, ... },
    ]
}
```

Question 📱

Which database structure will be best for retrieving all phone numbers?

A The relational database

B The non-relational database

C They will be about the same
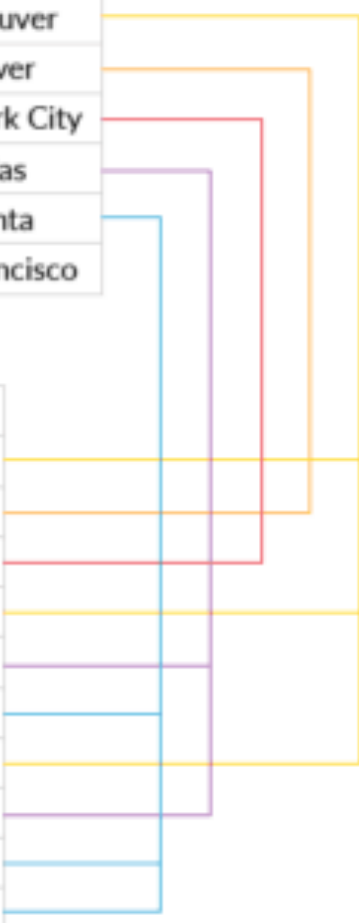
D I'm not sure

E [space intentionally left blank]

Relational

Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational

```
{
    first_name: 'Dexter',
    last_name: 'Lanas'
    city: 'Vancouver'
    location: [45.123,47.232],
    phones: [
        { phone_number: '111-111-1111',
            type: mobile,
            person_id: 1, ... },
        { phone_number: '444-444-4444',
            type: home,
            person_id: 1, ... },
        { phone_number: '777-777-7777',
            type: office,
            person_id: 1, ... },
    ]
}
```

Question 📱

Which database structure will be best for retrieving all phone numbers?

A The relational database

B The non-relational database

C They will be about the same

D I'm not sure

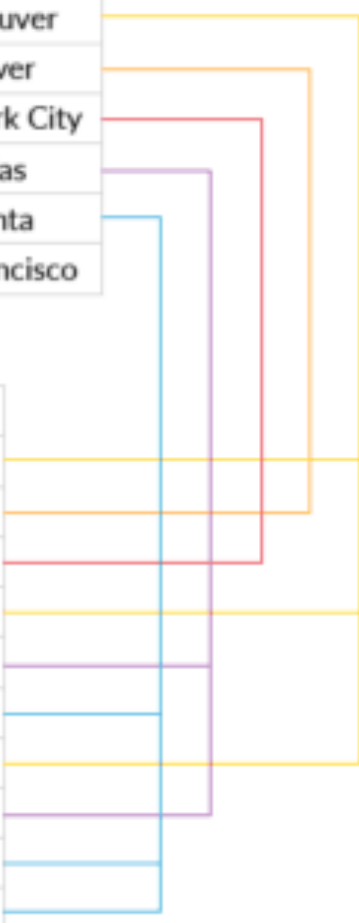**E** [space intentionally left blank]

Relational

Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational

```
{
    first_name: 'Dexter',
    last_name: 'Lanas'
    city: 'Vancouver'
    location: [45.123,47.232],
    phones: [
        { phone_number: '111-111-1111',
          type: mobile,
          person_id: 1, ... },
        { phone_number: '444-444-4444',
          type: home,
          person_id: 1, ... },
        { phone_number: '777-777-7777',
          type: office,
          person_id: 1, ... },
    ]
}
```

Question 📱

Which database structure will be best for retrieving all data?

A The relational database

B The non-relational database

C They will be about the same

D I'm not sure

E [space intentionally left blank]

Relational
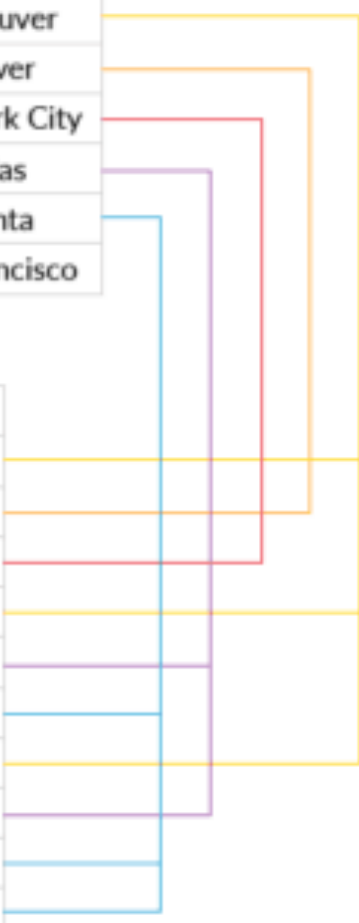
Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational

```
{
    first_name: 'Dexter',
    last_name: 'Lanas'
    city: 'Vancouver'
    location: [45.123,47.232],
    phones: [
        { phone_number: '111-111-1111',
          type: mobile,
          person_id: 1, ... },
        { phone_number: '444-444-4444',
          type: home,
          person_id: 1, ... },
        { phone_number: '777-777-7777',
          type: office,
          person_id: 1, ... },
    ]
}
```

Question 📱

Which database structure will be best for retrieving **all data**?

A The relational database

B The non-relational database

C They will be about the same

D I'm not sure

E [space intentionally left blank]

Relational
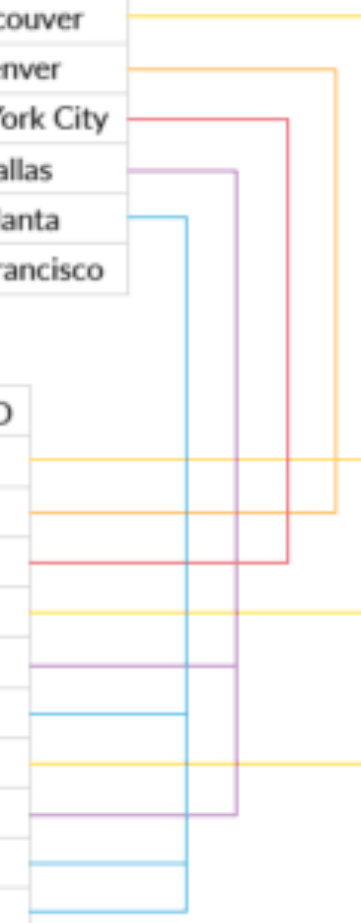
Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational

```
{
    first_name: 'Dexter',
    last_name: 'Lanas'
    city: 'Vancouver'
    location: [45.123,47.232],
    phones: [
        { phone_number: '111-111-1111',
          type: mobile,
          person_id: 1, ... },
        { phone_number: '444-444-4444',
          type: home,
          person_id: 1, ... },
        { phone_number: '777-777-7777',
          type: office,
          person_id: 1, ... },
    ]
}
```

Databases

Advantages of relational databases

- Relational databases support better querying

  - Provide *languages* for querying, such as Structured Query Language (SQL)

  - Those languages can be used to ask for specific tables or even join data across tables

  - "Give me the first name of every user whose phone number starts with 949"

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/
https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Advantages of relational databases

- Relational databases are more organized

  - Because field types are defined, data reliably follows that structure

- Relational databases are more reliable

  - Structure is enforced when new data is added

  - Transactions are atomic, so it's easy to "get" the current state of the database

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/
https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Advantages of non-relational databases

- Non-relational databases support more flexibility

    - Structure imposes restrictions

    - Adding a new field (column) can mess up a relational database

- Non-relational databases are faster for simple operations

    - It's much easier to "watch all the files" than to query and index many rows across multiple tables

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/
https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Relational vs. Non-relational

- Relational databases tend to be used in Enterprise, large-scale applications

  - It's important that data conforms to standards

  - It's important to robustly query large amounts of data

- Non-relational databases tend to be used in smaller applications

  - Data flexibility is valuable

  - Data is small enough to reliably retrieve and parse

- That said, plenty of large apps use non-relational databases and vice versa

Databases vs. Local Storage

- Who needs access to the data?

  - Just the user, or others?

  - As a developer, do you need access?

- Is the data sensitive?

- Is the data valuable enough that it should not be lost?

Databases vs. Local Storage

- Databases are crucial if more than the local device needs access

  - Cross-device app: <u>facebook.com</u> and the mobile app need your profile information

  - Developer: to understand habits across users or provide a data-driven service

- Some privacy can be preserved if data is only stored locally

- Which to use depends on the type of data and context

One non-relational database: Firebase

Firebase

- First released in 2011

- Acquired by Google in 2014

- Has features besides databases

  - Media storage

  - Authentication

  - Analytics

# Firebase

Setting up the database

- Create a new project: <u>https://firebase.google.com/</u>

- Create a database

Firebase
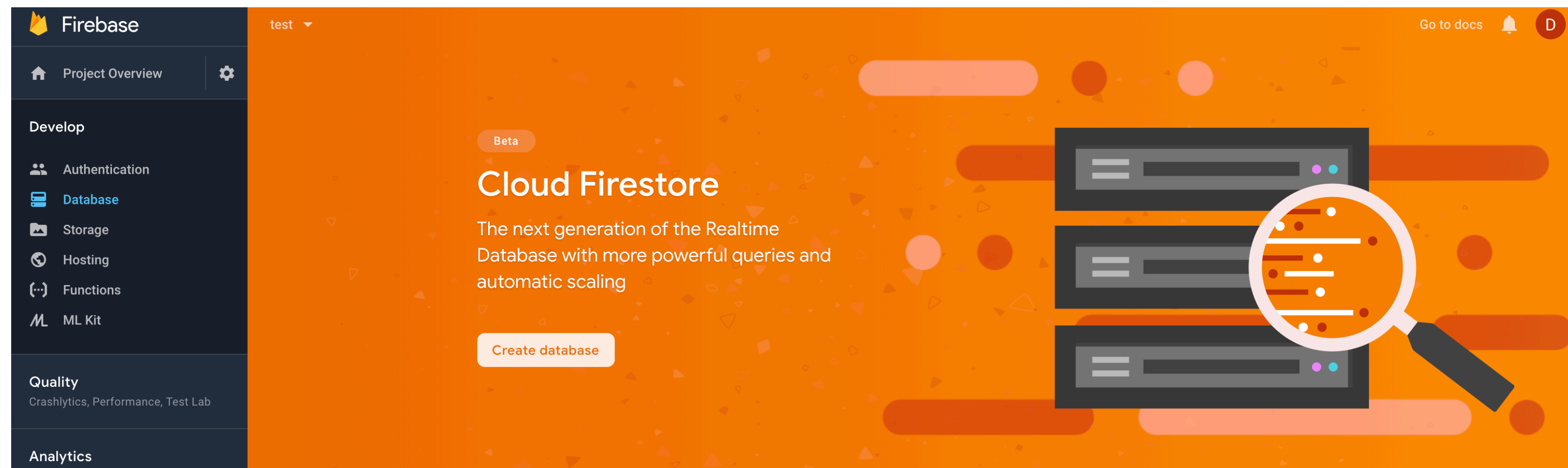
Setting up the database

- Start your database in "test mode"

  - Anyone can read or write
    to your database

  - This means *anyone*, even localhost

  - Gets around browser's origin restriction

  - This is bad practice, of course.
    It's better to allow specific users

  - Take a databases class
    to learn about permissions
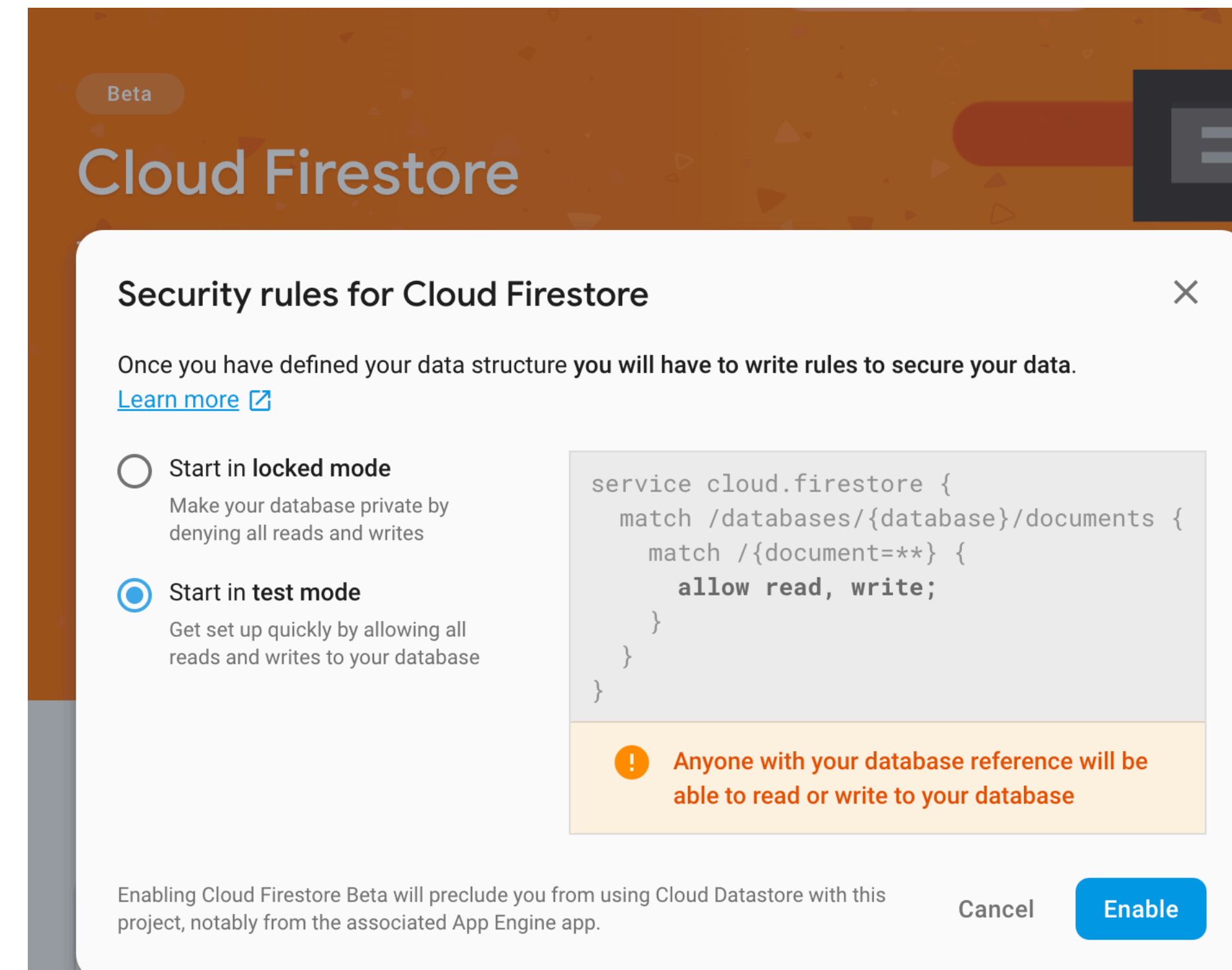
Firebase

Setting up the database

Firebase
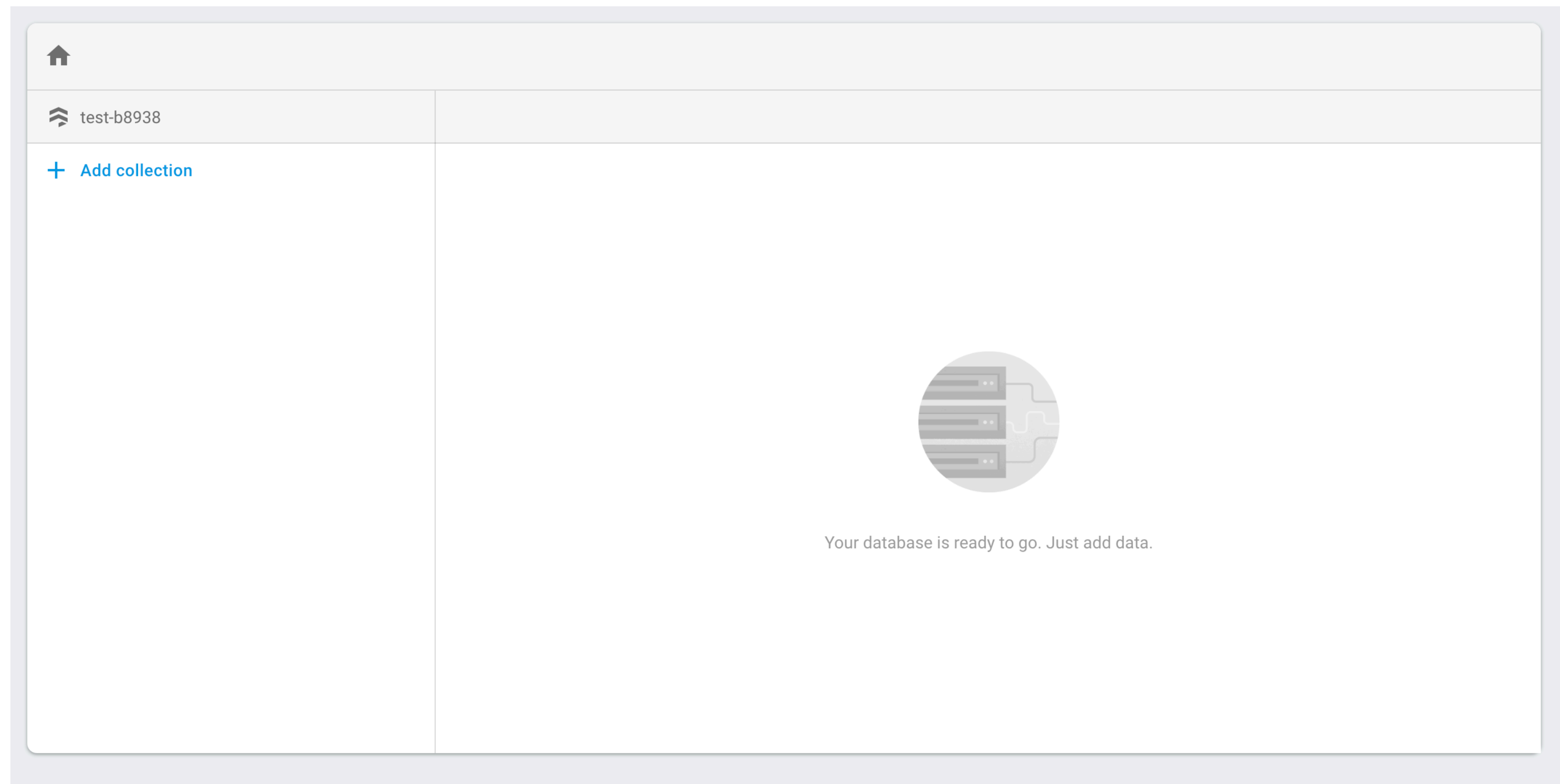
Setting up the database

- Firebase documents (objects) are organized into *collections*

- Collections are somewhat like tables in relational databases

- But Firebase is non-relational and has no structure requirement

- Multiple documents in the same collection may have different structure

- Example collections: users, sleepdata

https://firebase.google.com/docs/firestore/data-model

Firebase

Setting up the mobile app

- Angular officially supports a Firebase library

  - It works with Ionic since Ionic builds on Angular

- `npm install firebase`

- `npm install @angular/fire`

https://github.com/angular/angularfire

Firebase

Setting up the mobile app

- Add configuration information for your Firebase app to `environments.ts` file in Ionic

- Edit Ionic's `module.ts` to point to this environment information

- Also add AngularFirestoreModule to the `module.ts`

https://github.com/angular/angularfire/blob/master/docs/install-and-setup.md

Firebase
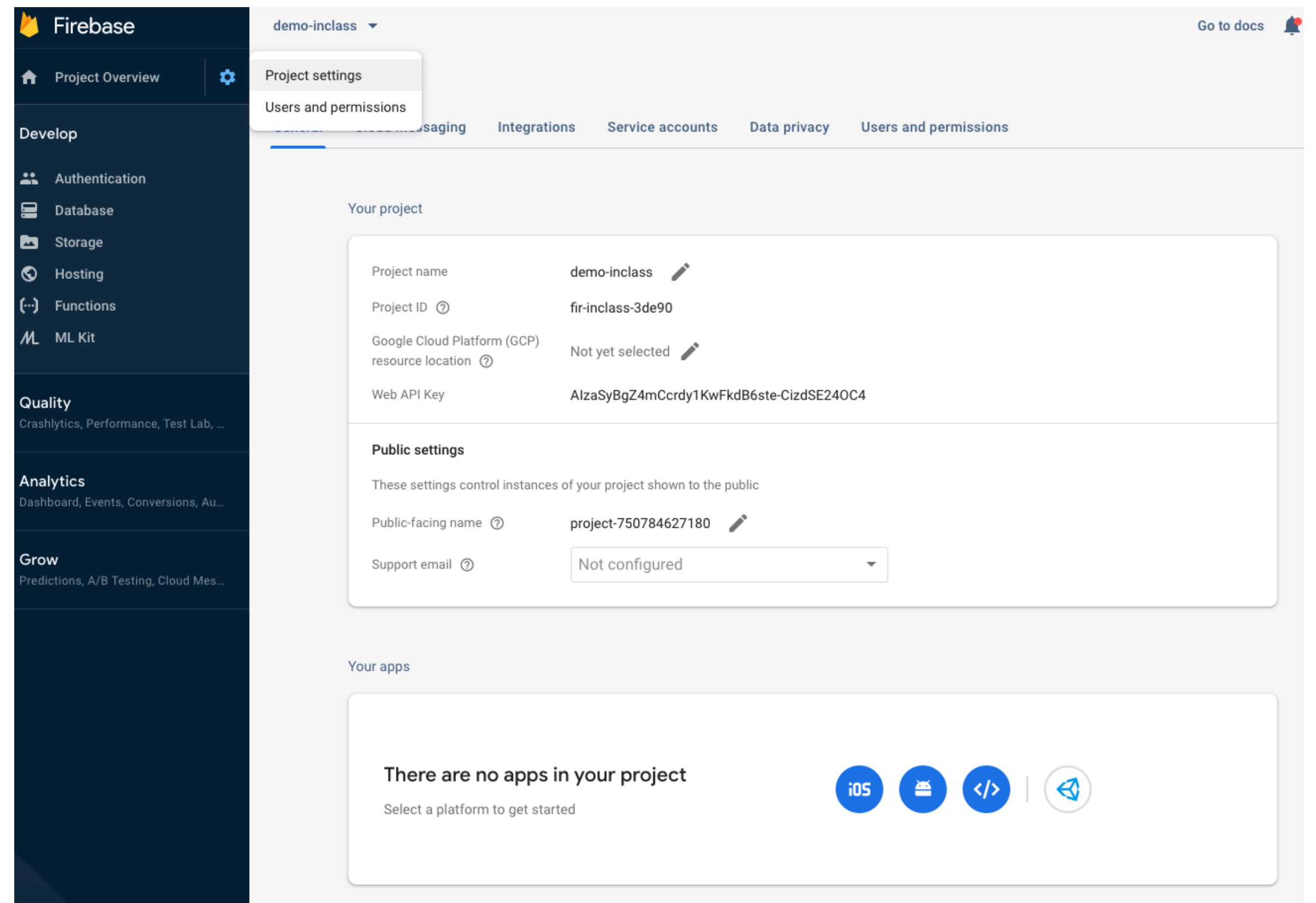
Accessing the database from the mobile app

- `AngularFirestore` is a service and is injected like any other service

 - Can retrieve a collection by its name

```typescript
import { AngularFirestore, AngularFirestoreCollection, DocumentData } from '@angular/fire/firestore';
import { Observable } from 'rxjs';

export class FirebaseService {
  collection:AngularFirestoreCollection;

  constructor(db:AngularFirestore) {
    this.collection = db.collection('test-collection');
  }
}
```

Firebase

Getting some data

Firebase
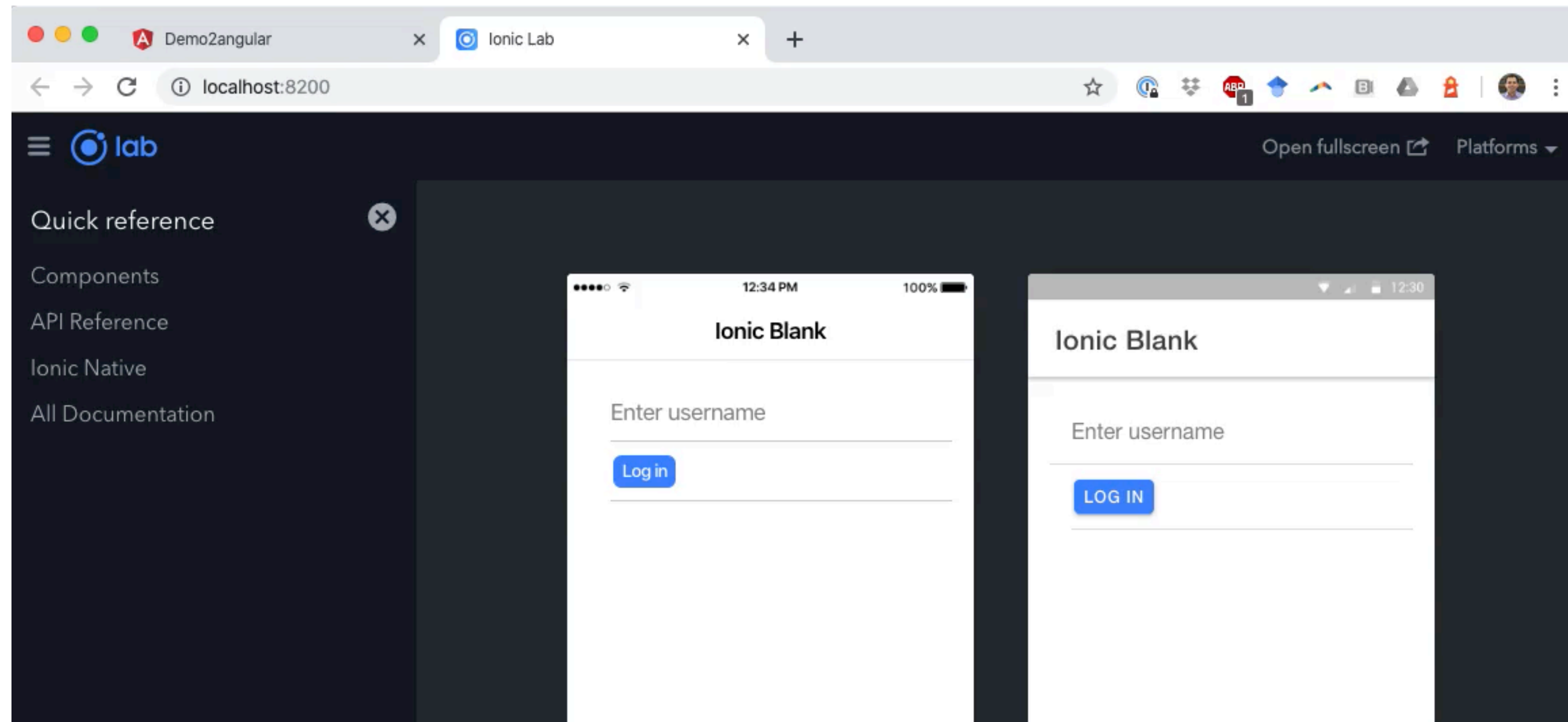
Accessing the database from the mobile app

- We probably don't want to "get" data once

  - What if someone logged their sleep from their desktop?

  - Documents can be large, it takes some time for a transaction to complete

  - Instead of "getting", we use an `Observable` to listen for any time the data changes

  - Same as listening for new accelerometer data every second with Ionic Native

Firebase

Listening for changes

```typescript
/* .component.ts */
export class MyApp {
  testItems: Observable<any[]>;
  constructor(db: AngularFirestore) {
    this.testItems = db.collection('test-collection').valueChanges();
  }
}
```

```html
<!--.component.html -->
<ul>
  <li *ngFor="let item of testItems | async">
    {{ item.name }}
  </li>
</ul>
```

Firebase

Add

- New objects can be added asynchronously

```typescript
export class FirebaseService {
  collection:AngularFirestoreCollection;

  constructor(db:AngularFirestore) {
    this.collection = db.collection('test-collection');
  }

  addData(data:{}) {
    this.collection.add(data).then((reference) => {
      console.log("Reference to added data, kind of like a URL");
      console.log(reference);
    });
  }
}
```

Firebase

Delete and Update

● The string reference can be used to delete or update documents

```
deleteDocument(reference:string) {
  this.collection.doc(reference).delete().then(() => {
    console.log('The document at ' + reference + 'no longer exists');
  });
}

updateDocument(reference:string, newData:{}) {
  this.collection.doc(reference).update(newData).then(() => {
    console.log('The document at ' + reference + 'is now ' + newData);
  });
}
```

Firebase

Converting TypeScript objects to and from JSON

- Firebase expects JSON rather than a TypeScript object

- TypeScript classes need to be converted to and from JSON

```typescript
export class DataLog {
  id:string;
  values:number[];

  toObject():{} {
    return {'id':this.id,
    'value':this.values};
  }

  fromObject(object:{}) {
    this.id = object['id'];
    this.values = object['value'];
  }
}
```

Firebase

Converting TypeScript objects to and from JSON

● Non-primitive fields, like Date, may need extra conversion

```typescript
export class DataLog {
  date:Date;

  toObject():{} {
    return {'date':this.date};
  }

  fromObject(object:{}) {
    //Stored as number of milliseconds
    this.date = new Date(object['date'].seconds*1000);
  }
}
```

Final exam

- In this room from 8-10am on Tuesday, December 10 (one week from today)

- <u>We will start right at 8</u>, you'll have the full 2 hours

  - Exam will be a little longer than the midterms

- ~50% "new" content (since midterm 2), ~50% "old" content

- Same rules as other exams: closed-note, closed-neighbor, closed-device, etc.

- <u>There is no make-up exam or late exam</u>

Today's goals

By the end of today, you should be able to…

- Differentiate relational from non-relational databases

- Explain the advantages of each style of database

- Use Firebase to implement a non-relational database

IN4MATX 133: User Interface Software

Lecture 19:
Databases and Local Storage

Professor Daniel A. Epstein
TA Lucas de Melo Silva
TA Jong Ho Lee