Class notes

- Regular office hours schedule this week
  - This week’s discussion is office hours, attendance is optional
- A4 is due end-of-day Wednesday
  - Partner survey is up
Class notes

- A5 is posted
  - Option 1: use a native device resource
  - Option 2: back up logged data to Firebase
  - Also evaluated on following principles of mobile design and creating an app with compelling visuals and/or features
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
Data storage

- What happens when we refresh the sleep tracking app we made in A4?
  - 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
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.mongodb.com/scale/relational-vs-non-relational-database
Databases

Relational databases

Relational

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.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.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.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.mongodb.com/scale/relational-vs-non-relational-database
Databases

Non-relational databases

```json
{
    // 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

```json
{
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "lastMessage": "ghopper: Relay malfunction found. Cause: moth."
    },
    "two": { ... }
  },
  "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]
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]
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]
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.mongodb.com/scale/relational-vs-non-relational-database
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.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.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
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: https://firebase.google.com/
- 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

Your database is ready to go. Just add data.
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/angularfire2
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

[Link to the detailed setup guide](https://github.com/angular/angularfire2/blob/master/docs/install-and-setup.md#4-add-firebase-config-to-environments-variable)
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

Accessing the database from the mobile app

● We don’t “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();
  }
}

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

```javascript
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);
  });
}
```
var citiesRef = db.collection("cities");

citiesRef.doc("SF").set(
    {name: "San Francisco", state: "CA", country: "USA",
     capital: false, population: 860000,
     regions: ["west_coast", "norcal"]
    });
citiesRef.doc("LA").set(
    {name: "Los Angeles", state: "CA", country: "USA",
     capital: false, population: 3900000,
     regions: ["west_coast", "socal"]
    });
citiesRef.doc("DC").set(
    {name: "Washington, D.C.", state: null, country: "USA",
     capital: true, population: 680000,
     regions: ["east_coast"]
    });
citiesRef.doc("TOK").set(
    {name: "Tokyo", state: null, country: "Japan",
     capital: true, population: 9000000,
     regions: ["kanto", "honshu"]
    });
citiesRef.doc("BJ").set(
    {name: "Beijing", state: null, country: "China",
     capital: true, population: 21500000,
     regions: ["jingjinji", "hebei"]
    });

var citiesRef = db.collection("cities");

citiesRef.where("state", "==", "CA");
// SF, LA

citiesRef.where("capital", "==", true);
// D.C., Tokyo, Beijing

citiesRef.where("population", "<", 1000000);
// LA, Tokyo, Beijing

citiesRef.where("name", ">=", "San Francisco");
// SF, Tokyo, D.C.

https://firebase.google.com/docs/firestore/query-data/queries
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);
    }
}
```
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 23: Databases

Professor Daniel A. Epstein
TA Jamshir Goorabian
TA Simion Padurean