ไฟร์เบส (Firebase)

ไฟร์เบส เป็นระบบฐานข้อมูลที่มีอยู่ด้วยกันสองแบบคือ ฐานข้อมูลเรียลไทม์ (Realtime Database) ที่เก็บข้อมูลเป็น JSON ซึ่งต่อไปจะเรียกสั้นว่า RTDB และคลาดไฟร์สโตร์ (Cloud Store) ซึ่งต่อไปจะเรียกสั้น ๆ ว่า CFire ดังจะเป็นเรื่องศึกษาการใช้งานทั้งสองชนิดฐานข้อมูล โดยมีไลบรารีแองกูล่าร์ไฟร์ (AngularFire) ทำงานกับฐานข้อมูลไฟร์เบส

แองกูล่าร์ไฟร์

แองกูล่าร์ไฟร์ เป็นไลบรารีที่ใช้ แองกูล่าร์ไฟร์ ทำงานกับระบบไฟร์เบสได้ทั้ง CFire และ RTDB การรับข้อมูลทำงานอยู่บนพื้นฐานค่าให้สังเกตการณ์ (Observable) ใช้การผูกข้อมูลแบบเรียลไทม์ รองรับการยืนยันตัวตน (Authentication) รองรับการทำงานแบบออฟไลน์ (Offline) คือสามารถเก็บข้อมูลอัตโนมัติแม้ไม่ได้ต่อเชื่อมกับระบบไฟร์เบส

ฐานข้อมูลเรียลไทม์

แองกูล่าร์ ใช้ AngularFireDatabase ทำงานกับ RTDB เป็นระบบฐานข้อมูลเริ่มแรกที่พัฒนาขึ้นมา ทำงานในลักษณะทันทีทันใดที่มีการต่อเชื่อมกับเครื่องลูกข่าย ไม่ว่าจะเป็น iOS, Android หรือ JavaScript

ระบบฐานข้อมูลของ RTDB เก็บในรูปแบบ JSON ซึ่งเป็นออบเจ็กต์อย่างหนึ่งของ JavaScript ข้อมูลที่เป็นออบเจ็กต์นี้เก็บได้ในรูปแบบอาร์เรย์ หรือ List การอ่าน การสืบค้น จึงเป็นอ่านข้อมูลผ่าน List

คลาดไฟร์สโตร์

แองกูล่าร์ ใช้ AngularFirestore ในการทำงานกับ CFire ด้วยคุณสมบัติใหม่ที่ทำให้สืบค้นได้รวดเร็ว ขยายระบบฐานข้อมูลได้ดีกว่า RTDB

ระบบฐานข้อมูลของ CFire เก็บข้อมูลในรูปแบบเอกสาร (Documents) ไม่ใช้งาน SQL (NoSQL) ไม่มีแถว ไม่มีตาราง แต่ละเอกสาร จะมีชื่อกำกับ ซึ่งจะรวมรวมเป็นชุดข้อมูล(Collections) อยู่ในรูปแบบ คีย์-ข้อมูล (Key-Value) รูปแบบข้อมูลนี้อยู่ในรูป JSON การสืบค้นใช้งานผ่าน ชุดข้อมูล

จากใช้เอกสารเพื่อเก็บข้อมูล JSON จึงทำให้ระบบฐานข้อมูล CFire มีความยืดหยุ่นกว่าระบบฐานข้อมูล RTDB ดังการขยายระบบฐานได้สะดวกกว่าใช้ข้อมูลเป็น JSON อย่างเดียว

ตาราง 1 ความแตกต่างระหว่าง Realtime Databaseกับ Cloud Firestore

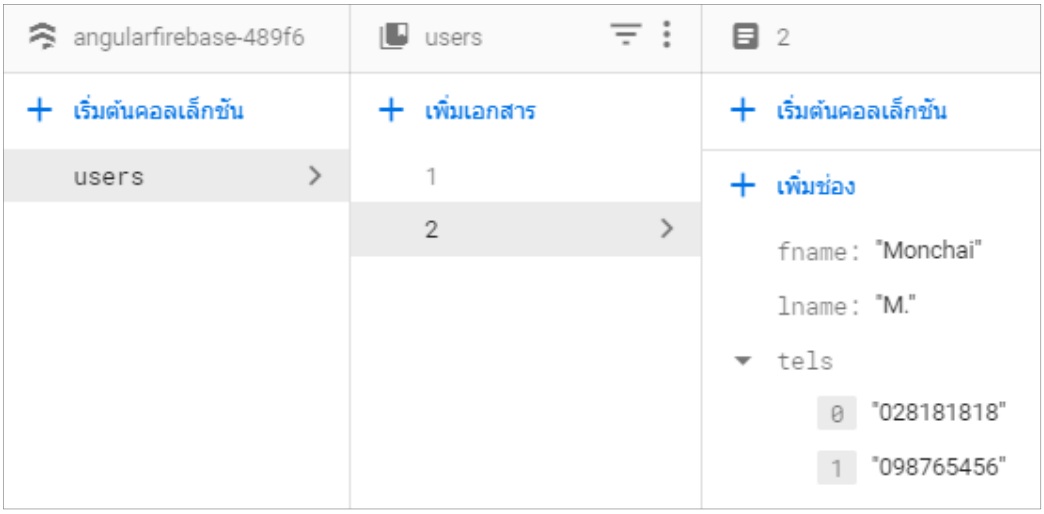
คุณสมบัติ	Realtime Database	Cloud Firestore
ชนิดฐานข้อมูล	NoSQL (JSON tree)	NoSQL (Document)
การสืบค้น	ได้ข้อมูลทั้งต้นไม้ JSON	ไม่ส่งค่าระดับย่อยของเอกสาร
ความน่าเชื่อถือ	จำกัดบางพื้นที่ที่ใช้ได้	ใช้ได้ทั่วโลก
การยายขนาด	ต่อเชื่อมได้ถึง 2 แสน และเขียนได้ 1 พันครั้งต่อวินาที	ต่อเชื่อมได้ 1 ล้าน เขียนได้ 1 หมื่นครั้งต่อวินาที
ความปลอดภัย	สร้างกฎได้	สร้างกฎได้
ราคา	เรียกเก็บตามขนาดพื้นที่เก็บและขนาดการเรียกใช้	เรียกเก็บตามจำนวนการดำเนินการต่อครั้ง (อ่าน เขียน ลบ)

ที่มา : https://firebase.google.com/docs/firestore/rtdb-vs-firestore (เข้าดูเมื่อ 11 กรกฎาคม 2563).

เริ่มต้นใช้งาน CFire

เพื่อให้เห็นภาพการใช้งานเบื้องต้น การใช้แองกูล่าร์ไฟร์ กับ CFlire ให้ทำการสมัครใช้งานฐานข้อมูล CFire ก่อน ฐานข้อมูลอยู่ในรูปเอกสาร เริ่มจากสร้าง คอลเล็คชัน แล้วเพิ่มเอกสาร และคอลเล็กชันของเอกสาร เช่น กำหนดลำดับข้อมูลตามรูปแบบ collection > document > collection ดังเขียนได้ว่า:

```
users > 1 >
        {fname: 'Theerapol', lname:'L.', tels:['021234567','098765443']}
      > 2 >
        {fname: 'Monchai', lname:'M.', tels:['028181818','098765456']}
```



รูป 1 ฐานข้อมูล Users เบื้องต้น

สำหรับชนิดฟิลด์ fname และ lname เลือกไทป์เป็น string ส่วน tels ให้เลือกชนิดข้อมูลแบบ array ในขั้นตอนการตรวจสอบกฎ ให้ตั้งค่าอ่านและเขียนได้เป็น true และเมื่อมีข้อมูลพร้อมให้ทดสอบแบบ ต่อไปจะเป็นขั้นตอนการอ่านข้อมูลผ่านแองกูล่าร์ ซึ่งมีขั้นตอนต่าง ๆ ดังนี้

1) สร้างแอปพลิเคชัน เลือกแบบมีเส้นทาง

```
ng new myAngular --routing
cd myAngular
```

2) เมื่อสร้างแอปพลิเคชันเริ่มต้นแล้ว ยังไม่ต้องเปิด (ng serve) ให้เพิ่มไลบรารี่ไฟร์เบส ผ่าน CLI:

```
ng add @angular/fire
```

ในระหว่างจะมีให้ใส่รหัสยืนยันตัวตนซึ่งได้จากการสร้างโปรเจ็กต์ ในระบบไฟร์เบส ดังตัวอย่างรูปต่อไปนี้

```
D:\SBC\Courses\Angular\Code\09firebase\myAngular>ng add @angular/fire
Installing packages for tooling via npm.
Installed packages for tooling via npm.
UPDATE package.json (1520 bytes)
√ Packages installed successfully.
? Allow Firebase to collect CLI usage and error reporting information? No
? Paste authorization code here: 4/1wEaxB50xrjJ_jPpsgIw6LR1foxp7ILLI1wYZGi
```

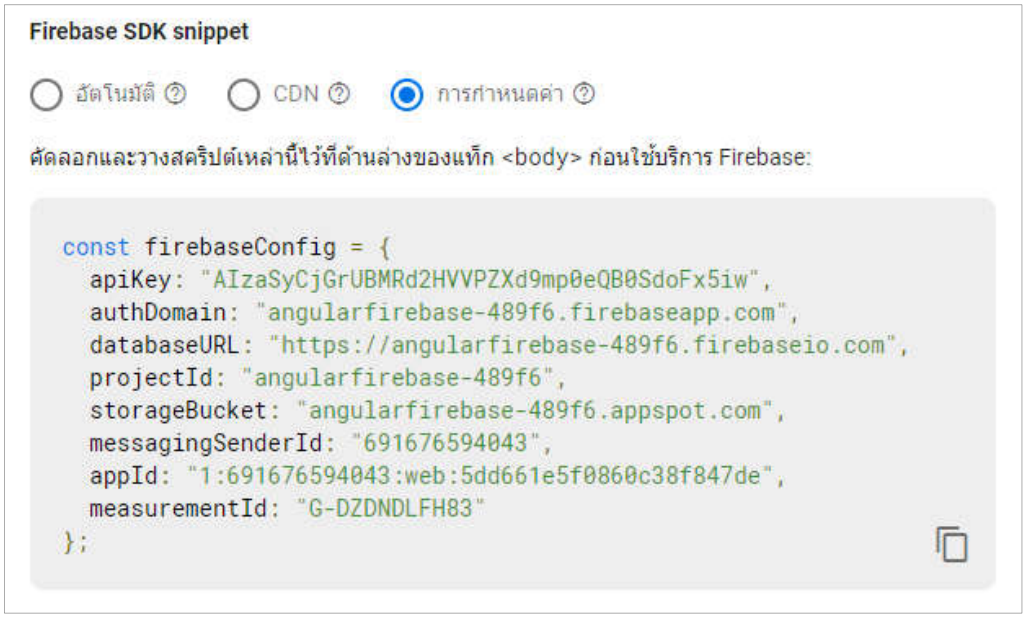
ขั้นตอนให้ใส่ authorization code ระบบจะให้เข้ารหัสอัตโนมัติผ่านการลงชื่อใช้ผ่านเบราเซอร์อัตโนมัติ นอกจากนี้ให้ตอบคำถามโปรเจ็กต์ ชื่อโปรเจ็กต์ไอดี (Project Id) ที่ได้สร้างไว้บนไฟร์เบส

```
√  Preparing the list of your Firebase projects
? Please select a project:
> AngularFirebase (angularfirebase-489f6)
  AssistantSample (assistantsample-kkpbwf)
```

3) หลังจากนให้เพิ่มข้อกำหนดในไฟล์ environment.ts ในข้อกำหนดนี้ ให้คัดลอกจาก Firebase console ซึ่งเป็นรูปเกียร์ คลิกรูปนี้ ต่อมาเลือก การตั้งค่าทั่วไป แล้วปรับแต่งบนพื้นฐานของข้อกำหนดต่อไปนี้ ในขั้นตอนนี้อาจมีให้ลงทะเบียนการใช้งานผ่านเว็บแอปฯ หรือโมบายล์แอปฯ ซึ่งในงานของเราใช้ผ่านเว็บแอปฯ

Code 1. src/environments/environment.ts

```
export const environment = {
  production: false,
  firebase: {
    apiKey: '<your-key>',
    authDomain: '<your-project-authdomain>',
    databaseURL: '<your-database-URL>',
    projectId: '<your-project-id>',
    storageBucket: '<your-storage-bucket>',
    messagingSenderId: '<your-messaging-sender-id>',
    appId: '<your-app-id>',
    measurementId: '<your-measurement-id>'
  }
};
```



4) ต่อมาให้กำหนด @NgModule สำหรับแองกูล่าร์ไฟร์เบส โดยการนำเข้าส่วนที่เกี่ยวข้อง ที่สำคัญคือ

- AngularFireModule เป็นโมดูลทำงานกับระบบ ไฟร์เบส
- AngularFirestoreModule เป็นโมดูลสำหรับทำงานกับระบบฐานข้อมูล CFire
- environment เป็นข้อกำหนดการใช้งานที่ได้ทำไว้ก่อนหน้านี้

Code 2. src/app/app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AngularFireModule } from '@angular/fire';
import { AngularFirestoreModule } from '@angular/fire/firestore';
import { environment } from '../environments/environment';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    AngularFireModule.initializeApp(environment.firebase),
    AngularFirestoreModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

5) ใส่คลาส AngularFirestore ลงในคอมโพเน้นท์หลัก อ่านข้อมูลเบื้องต้นจากไฟร์เบส

Code 3. src/app/app.component.ts

```
import { Component } from '@angular/core';
import { Observable } from 'rxjs';
import { AngularFirestore } from '@angular/fire/firestore';

@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html',
  styleUrls: ['app.component.css']
})
export class AppComponent {
  title = 'AngularFirebase';
  users: Observable<any[]>;
  constructor(firestore: AngularFirestore) {
      this.users = firestore.collection('users').valueChanges();
  }
}
```

ลำดับข้อนี้เป็นการอ่านทั้งคอลเล็คชัน users ที่มีทุกเอกสาร (1, 2) ภายในเอกสารประกอบด้วยคอลเล็คชันอีกทีหนึ่ง

6) ผูกข้อมูลเบื้องต้นกับหน้า HTML

Code 4. src/app/app.component.html

```
<ul>
      <li *ngFor='let user of users | async'>
          {{user.fname}} {{user.lname}} {{user.tels}}
      </li>
</ul>
```

7) ดูผลการทำงานผ่าน CLI จะได้เป็นรายชื่อที่มีทั้งหมดในฐานข้อมูล

```
ng serve --open
```

AngularFirestore

ฐานข้อมูล CFire สามารถจัดการได้ด้วย AngularFirestore การจัดการแยกเอกสารในลักษณะต่างๆ คือ Document, Collection และการจัดการกับข้อมูล Offline

ข้อมูลในรูปเอกสาร เก็บเป็นชุดข้อมูลหรือ คอลเล็คชัน (Collection) ซึ่งประกอบด้วยคู่ คีย์-ข้อมูล (key-value) จากตัวอย่างที่ผ่ามมา มีคอลเล็คชัน ชื่อ users ซึ่งประกอบด้วย เอกสาร มีชื่อ 1 มีคอลเล็คชัน { fname:"Theerapol", lname:"L."} และเอกสาร 2 มีคอลเล็คชัน ของคู่ของ fname และ lname ดังนั้นข้อมูลจึงมีลำดับการอ่านคือ คอลเล็คชัน เอกสาร และคอลเล็คชัน

การอ่านคอลเล็คชันของ CFire

เรามาเริ่มทำความเข้าใจ การอ่านในลำดับของเอกสาร โดยให้สร้างเป็นคอมโพเน้นท์ User เพื่อการแสดงผลผู้ใช้ (user) แต่ละราย คอมโพเน้นท์ users เพื่ออ่านทุกเอกสาร และงานบริการ users

```
ng generate component user
ng generate component users
ng generate component pageNotFound
ng generate service users
```

การอ่านเอกสารทุกเอกสารซึ่งอยู่ในรูปคอเล็คชันของเอกสาร แต่ละเอกสารก็จะมีคอลเล็คชันของตนเอง การอ่านจึงแยกได้สองแบบคือ การคอลเล็คขันแรก (ทุกเอกสาร) และการอ่านเฉพาะเอกสาร

เราจะมาทำความเข้าใจการอ่านทุกเอกสารกันก่อน การอ่านทุกเอกสารได้ทดลองทำมาบ้างแล้วในหัวข้อ **เริ่มต้นใช้งาน CFire** แต่ในลำดับต่อไปนี้จะแสดงการอ่านในรายละเอียดมากขึ้น

เพื่อการสาธิต ให้นำคอมโพเน้นท์ users ซึ่งต้องการแสดงรายการผู้ใช้ทุกรายใน app.component.html

Code 5. src/app/app.component.html

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
</div>
<router-outlet></ router-outlet>
```

เนื่องจากแอปพลิเคชันนี้ใช้ระบบเส้นทางจึงต้องเขียนเส้นทางไปยังคอมโพเน้นท์ต่าง ๆ และนำเข้าแต่ละคอมโพเน้นท์ด้วยที่มีในแต่ละเส้นทางด้วย

Code 6. src/app/app-routing.module.ts

```
const routes: Routes = [
  { path:'users', component: UsersComponent},
  { path:'user/:id', component: UserComponent},
  { path: '', redirectTo: '/users', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];
```

การอ่านทุกเอกสารถือเป็นคอลเล็คชันแรก จะต้องนำเข้าโมดูลที่จำเป็นอื่น ๆ ดังเพิ่มเติมในการนำเข้าในงานบริการ UserService และประกาศไทป์ User แทนโครงสร้างข้อมูล User

Code 7. src/app/users.service.ts

```
import {
  AngularFirestore,
  AngularFirestoreCollection,
  AngularFirestoreDocument } from '@angular/fire/firestore';
import { Observable} from 'rxjs';
import { map } from 'rxjs/operators';

export interface User { id:number; fname: string; lname:string; tels:string[] }
```

โมดูล AngularFirestoreCollection ทำงานกับชุดข้อมูลหรือคอลเล็คชันที่เป็นค่าเจอเนอร์ริก (generic) หรือข้อมูลที่ต้องการใช้ไทป์ ในที่นี้คือไทป์ User โมดูล AngularFirestoreDocument ทำงานกับเอกสาร

งานบริการนี้มีการทำงานในระดับแอปพลิเคชันแล้ว เพราะใช้การส่งต่อไปยัง root ผ่าน providedIn:root

```
@Injectable({
      providedIn: 'root'
})
```

ขั้นตอนต่อไป จะเป็นการทดลองอ่านข้อมูลทุกเอกสาร ผ่านงานบริการ ซึ่งทำได้หลายวิธี เริ่มจากวิธีแรก ซึ่งเป็นวิธีที่อย่างง่าย โดยไม่จำเป็นต้องระบุไทป์แน่นอน วิธีนี้ไม่จำเป็นต้อง นำเข้า AngularFirestoreCollection ก็ได้

Code 8. src/app/users.service.ts

```
export class UsersService {
   constructor(private firestore: AngularFirestore){  }

   getUsers():Observable<any[]>{
        return this.firestore
                   .collection('users')
                   .valueChanges({idField:'id'});
   }
}
```

จากตัวอย่างนี้ ได้เลือกคอลเล็คชัน users จากฟังก์ชัน collection(‘users’) และอ่านค่าคอลเล็คชันนี้ กับฟังก์ชัน valueChanges( ) **การคืนค่าของฟังก์ชันอยู่ในรูปข้อมูล JSON** สำหรับตัวแปรเข้าในฟังก์ชันนี้ {idField:’id’} ใช้แทนชื่อเอกสาร ในที่นี้ใช้ชื่อเป็น “id” และจะถูกจับไปเป็นคุณสมบัติในไทป์ user ในชื่อ id ดังได้กำหนดให้ไทป์ user มี Id เป็นชื่อหนึ่ง

นอกจาก fname และ lname แต่มีสิ่งที่น่าสังเกตอย่างหนึ่งคือ ค่าให้สังเกตการณ์เลือกใช้ไทป์ any[ ] ซึ่งไม่ระบุไทป์ที่แน่ชัด วิธีการนี้ก็ใช้งานสะดวก แต่อาจมีความผิดพลาดขณะทำงานได้

ต่อไปจะเป็นการนำบริการที่สร้างไปแสดผล ซึ่งใช้คอมโพเน้นท์ users แสดงผล การนำไปใช้ให้นำเข้าบริการนี้ และไทป์ user

Code 9. src/app/users/users.component.ts

```
import { Component, OnInit } from '@angular/core';
import { UsersService } from '../users.service';
import { User } from '../users.service';

@Component({
  selector: 'app-users',
  templateUrl: './users.component.html',
  styleUrls: ['./users.component.css']
})
export class UsersComponent implements OnInit {
   users: User[];
   constructor(private usersService:UsersService){  }

   ngOnInit(): void {
         this.usersService.getUsers().subscribe(users=>this.users=users);
   }
}
```

นอกจากนำเข้าส่วนที่จำเป็นแล้ว (User, UsersService) ต้องใส่บริการลงในคอนสตรักเตอร์ ให้เป็นสมาชิกหนึ่งของคลาส UsersComponent นี้ การเรียกใช้งานงานบริการ getUsers( ) ใช้งานผ่านฟังก์ชัน ngOnInit( ) โดยการสมัครรับบริการให้มีค่าได้เท่ากับ users ซึ่งกำหนดเป็นอาร์เรย์ภายในคลาสนี้

ต่อไปก็จะเป็นการผูกข้อมูลเพื่อแสดงผล การแสดงผลนี้ ใช้การแสดงทั้ง Id, fname, และ lname ใช้การวนซ้ำของ *ngFor บน <li> สำหรับ tels เป็นอาร์เรย์ จึงต้องวนซ้ำอีกรอบ

Code 10. src/app/users/users.component.html

```
<table class="table">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First Nmae</th>
      <th scope="col">Last Name</th>
        <th scope="col">Tel</th>
    </tr>
  </thead>
  <tbody>
     <tr [ngStyle]="{'cursor':'pointer'}"
        *ngFor="let user of users" [routerLink]="['/user', user.id]">
      <th scope="row">{{user.id}}</th>
      <td>{{user.fname}}</td>
      <td>{{user.lname}}</td>
        <td>
        <span [ngStyle]="{'margin':'0 4px'}"
             *ngFor="let tel of user.tels">{{tel}}</span> </td>
```

```
    </tr>
</table>
```

นอกจากการสร้างงานบริการที่ให้ค่าสังเกตการณ์ไม่ระบุไทป์แน่ชัดแล้ว ยังมีวิธีที่ต้องการระบุไทป์ที่แน่นอน แต่ไม่ใช้ใส่ไทป์ User[ ] แทน any[ ] ตามวิธีเดิมที่ได้ทำมา

วิธีการต่อไปนี้ ใช้การระบุไทป์ผ่านฟังก์ชัน collection<User> ซึ่งให้ผลเหมือนกับวิธีที่ผ่านมา แต่อย่างลืมนำเข้า AngularFirestore, และ Observable

Code 11. src/app/users/users.component.ts

```
constructor(private usersService:UsersService, private firestore: AngularFirestore
){  }

getUsers():Observable<User[]>{
        return this.firestore
                    .collection<User>('users')
                    .valueChanges({idField:'id'});
}
```

#	First Nmae	Last Name	Tel
1	theerapol	L.	021234567 098765443
2	Monchai	M.	028181818 098765456

รูป 2 ผลการอ่านจากฐานข้อมูล (ตารางนี้ใช้ CSS จาก Bootstrap 4)

อีกวิธีที่ใช้งานผ่าน AngularFirestoreCollection วิธีการนี้ต้องนำเข้าโมดูลงานบริการนี้ด้วย งานบริการนี้ใช้สำหรับคอลเล็คชันของค่าเจนเนอริกไทป์โดยเฉพาะ การกำหนดให้ usersCollection เป็นสมาชิกหนึ่งของคลาส เพื่อต้องการอ้างอิงต่อไปใช้ฟังก์ชันที่เรียกใช้งาน ซึ่งอาจมีมากกว่าฟังก์ชัน getUsers( ) ซึ่งต่อไปจะใช้ฟังก์ชัน getUsers( ) นี้แทนฟังก์ชันที่สร้างก่อนหน้านี้

Code 12. src/app/users/users.service.ts

```
private usersCollection:AngularFirestoreCollection<User>;

constructor(private firestore: AngularFirestore){
      this.usersCollection = this.firestore.collection<User>('users');
}
getUsers():Observable<User[]>{
      return this.usersCollection.valueChanges({idField:'id'});
}
```

ในบางกรณีข้อมูลที่ได้มาต้องการจะแก้ไข ให้ตรงไทป์ หรือกำหนดค่าบางส่วนใหม่ สามารถแก้ไขระหว่างทางได้ด้วยการใช้ผ่านฟังก์ชัน snapshotChanges( ) ซึ่งคืนค่าในรูปอาร์เรย์ของ DocumentChangeAction[ ] อาร์เรย์นี้มีค่าสองค่าคือ type และ payload ค่าตัวหลังจะมีค่าของเอกสาร (doc) ตั้งจะนำค่านี้ไปดำเนินการต่อไปได้ เช่น ดำเนินการกับฟังก์ชัน pipe( ) การเชื่อมต่อคำสั่งด้วย pipe( ) ทำให้อ่านค่า หรือแก้ไขค่าได้ ดังตัวอย่างต่อไปนี้

Code 13. src/app/users/users.service.ts

```
getUsers():Observable<User[]>{
        return this.usersCollection
                    .snapshotChanges()
                    .pipe(map(actions => {
                                return actions.map(a => {
                                const data = a.payload.doc.data() as User;
                                data.id = +a.payload.doc.id;
                                console.log(data.id);
                                return data;
                                });
                          }));
}
```

จากตัวอย่างนี้ใช้การอ่านค่า id จาก payload.doc ต่างคุณสมบัติต่าง ๆ มาจากไทป์ DocumentChagneAction ดังจะได้อธิบายต่อไป

ไทป์ DocmentChangeAction

ไทป์อินเทอร์เฟส DocumentChangeAction ชุดของอินเทอร์เฟสที่อ้างอิงกันสามอินเทอร์เฟส โดยมีจุดเริ่มต้นที่ไทป์ DocumentChangeAction มีคุณสมบัติ type บอกถึงการดำเนินการในสามลักษณะคือ เพิ่มแล้ว แก้ไขแล้ว และเอาออกแล้ว (added, modified, removed) และมีคุณสมบัติ payload บอกถึง DocumentChange ซึ่งเป็นอีกไทป์หนึ่งโดยไทป์นี้ค่าของเอกสาร ค่าเอกสารที่สำคัญคือ doc มีไทป์ DocuemntSnapshot ดังอ่านได้จาก 3 อินเทอร์เฟสต่อไปนี้

```
interface DocumentChangeAction {
  //'added' | 'modified' | 'removed';
  type: DocumentChangeType;
  payload: DocumentChange;
}

interface DocumentChange {
  type: DocumentChangeType;
  doc: DocumentSnapshot;
  oldIndex: number;
  newIndex: number;
}

interface DocumentSnapshot {
  exists: boolean;
  ref: DocumentReference;
  id: string;
  metadata: SnapshotMetadata;
  data(): DocumentData;
  get(fieldPath: string): any;
}
```

จากตัวอย่างที่ใช้อ่านค่าผ่านฟังก์ชัน getUsers( ) ที่ผ่านมาใช้การอ่านกับ snapshotChanges( ) ซึ่งคืนค่าเป็นค่าสังเกตการณ์ได้ (Observable<User[ ]) ของไทป์ DocumentChangeAction ถ้าให้แสดงผลการอ่านในลักษณะต่าง ๆ บางค่าผ่าน console.log( ) จะมีผลดังนี้

```
return actions.map(a => {
       console.log(a.payload.doc.data());
       //ค่าข้อมูลในรูปแบบ JSON ข้อมูลแรกคือ {tels: Array(2), lname: "L.", fname: "theerapol"}
       console.log(a.payload.doc.id);
       //ค่าข้อมูลชื่อเอกสาร ในรูปแบบ string
       console.log(a.type);
       //ค่าข้อมูลการเปลี่ยนแปลง added
       console.log(a.payload.newIndex);
       //ค่าข้อมูลตามลำดับ กรณีมีสองชุดเอกสาร จะได้เลขที่ 0, 1
       console.log(a.payload.doc.get('fname'));
       //ค่าข้อมูลเฉพาะ fname ซึ่งจะได้ therapol, Monchai
      const data = a.payload.doc.data() as User;
       return data;
});
```

การเพิ่มและลบเอกสารของ CFire

ในการเพิ่มเอกสารเข้าไปในคอลเล็คชัน กรณีนี้เพิ่มในคอลเล็คชัน users มีรูปแบบการเพิ่มคือ

```
this.usersCollection.doc(id).set(user);
```

โดยที่ usersCollection คืออ๊อบเจ็กต์ AngularFirestoreCollection<User> มี User เป็นไทป์ User ส่วน id คือชื่อเอกสารมีไทป์เป็น String และ user เป็นคอลเล็คชัน

id เป็นค่าที่ระบบฐานข้อมูลไฟร์เบสสร้างให้อัตโนมัติ ไม่มีลำดับอะไร เป็นค่าอักษรที่สุ่มขึ้นมา อย่างไรก็ตามในกรณีที่ต้องการสร้างค่าที่มีลำดับจะต้องสร้างชื่อเอกสารขึ้นมาเอง เช่น ในคอลเล็คชัน users มี id เป็น ลำดับ 1, 2 ตัวต่อไปต้องการให้เป็น 3 ต้องทราบว่าของเดิมมีเลขอะไรสูงสุด วิธีหนึ่งที่ทำได้คือนับว่ามีจำนวนเอกสารเท่าใด เช่น นับว่าได้ 2 ค่า id ต่อไปก็ควรเป็น 3

Code 14. src/app/users/users.service.ts

```
usersCollection: AngularFirestoreCollection<User>;

addUser(user:User){
       this.usersCollection.get().subscribe(data=>{
              const id:string = String(data.docs.length+1);
              this.usersCollection.doc(id).set(user);
       });
}
delUser(id:string){
         this.usersCollection.doc(id).delete();
}
```

ในฟังก์ชัน addUser( ) มีการอ่านค่าจำนวนเอกสาร ผ่านการสมัครของฟังก์ชัน subscribe( ) ข้อมูลที่ได้ ตั้งชื่อเป็นตัวแปรว่า data แล้วใช้คุณสมบัติ docs.length เพื่ออ่านจำนวนเอกสาร และทำการบวกไปหนึ่งค่า

เมื่อได้จำนวนเอกสารแล้วค่อยนำไปเพิ่มเอกสารตามชื่อเอกสาร (id) ให้สังเกตว่ามีไทป์เป็น string ที่ผ่านการแปลงจากไทป์ number ไปสู่ string

สำหรับอ๊อบเจ็กต์ user ก็คือคอลเล็คชันที่ต้องการสร้าง แต่ในโครงสร้างอ๊อบเจ็กต์นี้ มี id เป็นส่วนประกอบด้วย แต่คอลเล็คชันที่สร้างที่ไพร์เบส ไม่มีค่านี้ ซึ่งก็ไม่เป็นไร เพราะค่าที่ไม่มีจะไม่ถูกนำไปรวมในคอลเล็คชัน แต่ในขั้นการเพิ่มเอกสารให้กำหนดค่าให้ครบตามที่กำหนดในไทป์ เช่น

```
let user:User = {id:0, fname:"Somcard",lname:"T.", tels:[]};
```

สำหรับการลบเอกสารลบตามฟังก์ชัน delUser( ) ใช้ตัวแปรเข้าเป็น id มีไทป์เป็น string เพื่ออ้างอิงเป็นชื่อเอกสารในการลบ ซึ่งใช้เพียงฟังก์ชัน delete( )

ตัวอย่างการใช้ในฟังก์ชัน เพื่อเพิ่มและลบ ให้สังการลบตัวแปรเข้าเป็น string จึงต้องเขียนเป็น “3” แทน 3 โดยทดสอบที่ฟังก์ชัน ngOnInit( ) ซึ่งทำงานเปิดเปิดคอมโพเน้นท์นี้ และเมื่อทดสอบแล้วควรใส่เครื่องหมาย // เพื่อปิดการทำงานหรือจะลบทั้งทั้งบรรทัด

Code 15. src/app/users/users.component.ts

```
ngOnInit(): void {
        //this.getUsers().subscribe(users=>this.users=users);
        //this.usersService.addUser({id:0, fname:"Somcard",lname:"T.", tels:[]});
        //this.usersService.delUser("3");
        //this.usersService.getUsers().subscribe(users=>this.users=users);
        this.usersService.getUsers().subscribe(users=>this.users=users);
}
```

การอ่าน และปรับปรุ่งเอกสารของ CFire

ที่ผ่านมาได้ทดลองอ่านคอลเล็คชัน โดยใช้ AngularFirestoreCollection จะได้ข้อมูลทั้งคอลเล็คชัน สำหรับการอ่านเพียงเอกสารใดเอกสารหนึ่ง จะมีคลาสงานบริการ AngularFirestoreDocument ซึ่งต้องนำเข้า ดังที่ได้เพิ่มส่วนนี้แล้วในตอนต้น

ในการอ่านเอกสารของคอลเล็คชัน users รายใด ข้อมูลจะอยู่ในรูปแบบเอกสาร จากตัวอย่างที่สร้างคือ เอกสาร 1 และ 2 รูปแบบการอ่านจึงอยู่ในรูปแบบ ‘collection/document’ เช่น กรณีอ่านเอกสารที่ 2 จะเขียนในรูป ‘users/2’

ตัวอย่างต่อไปนี้เป็นการเพิ่มงานบริการ getUser(id:string) เพื่ออ่านเอกสารตาม id โดยใช้ userDocument เป็นการรับค่าเอกสาร

Code 16. src/app/users.service.ts

```
private userDocument:AngularFirestoreDocument<User>;
getUser(id:string):Observable<User>{
        this.userDocument = this.firestore.doc<User>('users/'+id);
        return this.userDocument.valueChanges();
}
```

สำหรับการปรับปรุงข้อมูล ใช้เพียงฟังก์ชัน update(user) โดยตัวแปรเข้าใช้รูปแบบข้อมูล JSON ให้สอดคล้องกับที่สร้างไว้ในฐานข้อมูล เช่น ตัวแปร user นี้มีข้อมูล

```
{ fname:'Monchai', lname:'M.', tels:['028181818', '098765456']}
```

Code 17. src/app/users.service.ts

```
updateUser(user:User) {
	this.userDocument.update(user);
}
```

จากตัวอย่างนี้ใช้อ้างอิงเอกสารเดียวกับที่ใช้ในฟังก์ชัน getUser( ) ซึ่งหมายความว่าฟังก์ชันนี้จะใช้ได้ก็ต่อเมื่อมีการเรียกฟังก์ชัน getUser( ) มาก่อนเพราะฟังก์ชัน getUser( ) มีการกำหนดเอกสารไว้ว่าชื่ออะไร หากต้องการจะสร้างการอ้างอิงเอกสารเอง ก็ต้องปรับปรุงฟังก์ชัน update( ) เสียใหม่ ดังตัวอย่างต่อไปนี้

Code 18. src/app/users.service.ts

```
updateUser(id:number, user:User) {
	this.getUser(id);
	this.userDocument.update(user);
}
```

นำมาทดสอบในฟังก์ชัน ngOnInit( ) อีกครั้ง เป็นการปรับปรุงเอกสาร 3 (ต้องมีเอกสาร 3 ก่อน ถ้าไม่มีก็ให้เพิ่มเอกสารนี้ หรือจะปรังปรุงเอกสารหมายเลขอื่นก็ได้)

Code 19. src/app/users/users.component.ts

```
ngOnInit(): void {
	let user:User ={fname:'Monchai',lname:'M.',tels:['028181818', '098765456']}
	this.usersService.updateUser('3', user);
	this.usersService.getUsers().subscribe(users=>this.users=users);
}
```

แต่ดูเหมือนว่า บางทีต้องการปรับข้อมูลบางรายการไม่ได้ต้องการทั้งหมด เช่น ปรับปรุงเฉพาะ ชื่อ วิธีการคืออ่านค่าออบเจ็กต์ที่อยู่ในเอกสารมาก่อน แล้วเปลี่ยนแปลงเฉพาะชื่อ แล้วปรับปรุงออบเจ็ตกับฟังก์ชัน updateUser( ) ดังตัวอย่างต่อไปนี้

Code 20. src/app/users/users.component.ts

```
ngOnInit(): void {
	this.userService.getUser('3').subscribe(user=>{
		user.fname = 'Vachira';
		tshis.usersService.updateUser('3', user);
	}
	this.usersService.getUsers().subscribe(users=>this.users=users);
}
```

จากตัวอย่างมีการเรียกฟังก์ชัน subscribe( ) ก่อนเพื่อให้ได้ข้อมูลออบเจ็กต์ User ที่จะนำไปปรับปรุงต่อไป

การสืบค้นบน CFire

การสืบค้นผ่านคอลเล็คชัน Angularfirebase สร้างบนพื้นฐานของ firestore.CollectionReference มีเมธอดที่สำคัญคือ where แต่มีข้อจำกัดหลายอย่าง เช่น สืนค้าได้ทีละหนึ่งฟิลด์ หรือหนึ่งค่าในคอลเล็คชัน

ในงานที่สร้างมาของคอลเล็คชัน users ถ้าต้องการสืบค้นค่าของ fname ตามเงื่อนไขของเมธอดของ where โดยมีการเปรียบเทียบในลักษณะ เท่ากัน มากกว่า น้อยกว่า (==, >=, <=, >, <) แต่ไม่มีเงื่อนไข ไม่เท่ากับ หรือไม่มากกว่า นี้ก็เป็นข้อจำกัดอีกอย่างหนึ่ง

ตัวอย่างต่อไปนี้ แสดงการสืบค้นกับเงื่อนไข == กับ fname โดยการใส่เป็นตัวแปร สามตัวในเมธอด where ผลการสืบค้นได้ค่าเป็น Obserable<any[ ]> และใช้ตัวอ้างอิง this.firestore.collection( ) โดยมีชื่อคอลเล็คชันเป็นตัวสืบค้นในตัวแปรแรก ส่วนตัวแปรที่สองเป็นแลมบ์ดา (ref)

Code 21. src/app/users.service.ts

```
getCollection(query:string):Observable<any[]>{
	return this.firestore.collection('users',
		ref=>  ref.where('fname','==',query))
			.valueChanges({idField:'id'});
}
```

จากตัวอย่างนี้จะเห็นแล้วว่ามีจำกัดในการสืบค้นที่ทำได้เพียงฟิลด์เดียว แต่ถ้าต้องการสืบค้นกับฟิลด์ lname ด้วยต้องสร้างเป็นเงื่อนไข หรือ (|) ตัวอย่างต่อไปนี้แสดงการสืบค้นของสองฟิลด์ (fname, lname) และให้ผลการสืบค้นในรูปแบบ Observable<User[ ]>

Code 22. src/app/users.service.ts

```
getCollection(query:string):Observable<User[]>{
	const q =this.usersCollection.valueChanges({idField:'id'});
	return q.pipe(
		 map((users:User[])=>users.filter((user:User)=>
		     (user.fname ==query)|| (user.lname==query)))
	);
}
```

การสืบค้นนี้ใช้การเชื่อมต่อคำสั่งด้วย pipe( ) และใช้ตัวดำเนินการ map( ) การสืบค้นนี้ไม่อยู่เงื่อนไขเมธอด where ซึ่งทำได้ง่ายกว่า

ด้วยเงื่อนไข หรือ (||) ทำให้ต่อคำสั่งได้เรื่อย ๆ แต่ถ้ามีบางฟิลด์ มีค่าเป็นอาร์เรย์ เช่น tels มีค่าเป็นอาร์เรย์ของ users อีกทีหนึ่ง การใช้เงื่อนไข หรือ ต่อไป ก็ย่อมทำได้แต่ ใช้การกรองของฟังก์ชัน filter( ) อีกครั้ง การกรองอีกครั้งจะได้ผลการกรองเป็นจำนวนอาร์เรย์ ซึ่งต้องตั้งเงื่อนไขซ้อน เช่น ถ้าได้ผลมีขนาดอาร์เรย์มากกว่าศูนย์ ดังตัวอย่างต่อไปนี้

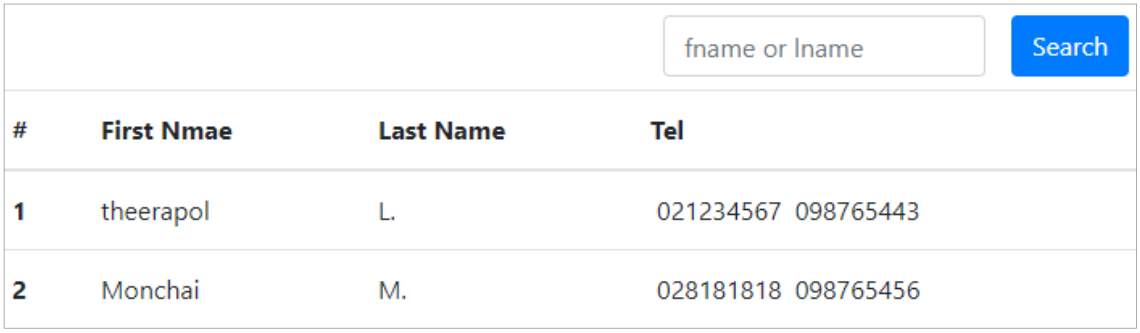
Code 23. src/app/users.service.ts

```
getCollection(query:string):Observable<User[]>{
	const q =this.usersCollection.valueChanges({idField:'id'});
	return q.pipe(
		map((users:User[])=>users.filter((user:User)=>
		(user.fname == query)
		|| (user.lname == query)
		|| (user.tels.filter((tel:string)=> tel == query).length>0 )
		)));
}
```

เพื่อการทดสอบการสืบค้นตามตัวอย่างการสืบค้น getCollection( ) สมมุติให้การสืบค้น ชื่อและสกุล ผ่านคอมโพเน้นท์ UsersComponent โดยสร้างเป็นฟอร์ม ที่มีเพียงหนึ่ง <input> โดยใช้ช่องสืบค้นนี้แทนทั้ง การสืบค้นทั้งชื่อและสกุล โดยวางตัวอย่างโปรแกรมต่อไปนี้บนตารางแสดงรายการ users

Code 24. src/app/users/users.component.html

```
<div [ngStyle]="{'float':'right','margin-right':'10px'}">
<form class="form-inline" (ngSubmit)="onQuery()">
  <div class="form-group mx-sm-3 mb-2">
    <input type="text" class="form-control" placeholder='fname or lname'
      [(ngModel)]="query" name="query" >
  </div>
  <button type="submit" class="btn btn-primary mb-2">Search</button>
</form>
</div>
```



รูป 3 แสดงฟอร์มการสืบค้น

ฟอร์มการสืบค้นนี้ทำงานกับฟังก์ชัน onQuery( ) ให้เมธอดนี้สมัครใช้บริการจาก UsersService ส่งตัวแปรคำเป็นคำค้น (query) ที่มาจากฟอร์ม เนื่องจากใช้งานฟอร์ม จะต้องนำเข้า `FromModule` ในระดับโมดูลด้วย

```
import {FromModule} from '@angalr/forms';
```

Code 25. src/app/users/users.component.ts

```
query:string;
onQuery(){
      //console.log(this.query);
      this.usersService
          .getCollection(this.query)
          .subscribe(users=>this.users=users);
}
```

ในตัวอย่าง อาจทดสอบแสดงผลคำค้น (query) ก่อนว่าทำงานได้ผลหรือไม่ผ่าน console.log( ) แล้วค่อยสมัครรับงานบริการการสืบค้น

การทำงานแบบออฟไลน์ (Offline)

การทำให้แอปพลิเคชันทำงานในโหมดออฟไลน์ ในยามที่ไม่สามารถติดต่อกับเซิร์บเวอร์ ข้อมูลใดที่เคยเก็บมาได้ก่อนการขาดการติดต่อกับเซิร์บเวอร์ จะถูกเก็บในหน่วยความจำชั่วคราว ซึ่งจะมีบางเบราเซอร์เท่านั้นที่รับรองการทำงานแบบนี้ได้

คือ Chrome, Safari, และ Firefox แต่การใช้งานถ้าเปิดเบราเซอร์หลาย ๆ แทป (tab) หน้าหลายหน้าต่าง หน้าต่างแรกเท่านั้นจะทำงานได้ถูกต้อง

วิธีการทำให้เก็บข้อมูลออฟไลน์ได้ง่ายนิดเดียวเพียงเติมการนำเข้าในโมดูลหลักเพียงรายการเดียวก็ทำได้งานได้แล้ว สำหรับการตรวจสอบว่า แอปพลิเคชันกำลังทำงานแบบออฟไลน์หรือไม่ ได้จากการตรวจสอบคุณสมบัติ fromCache จาก SnapshotMetadata ถ้าได้ข้อมูลคุณสมบัติเป็น false ถือว่ากำลังใช้ข้อมูลแบบออฟไลน์ แต่ถ้าได้ค่าคุณสมบัติเป็น true แสดงว่าข้อมูลปัจจุบันจากเซิร์บเวอร์ได้

Code 26. src/app/app.module.ts

```
@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
       AngularFireModule.initializeApp(environment.firebase),
       AngularFirestoreModule.enablePersistence(),
       AngularFirestoreModule,
       FormsModule,
  ],
```

ตัวอย่างต่อไปนี้เป็นการทดสอบการอ่านค่าว่ามาจากออฟไลน์หรือไม่ ผ่าน console.log( ) ของการอ่านข้อมูลคอลเล็คชัน users

Code 27. src/app/users.service.ts

```
getUsers():Observable<User[]>{
       return this.usersCollection
              .snapshotChanges()
              .pipe(map(actions => {
                     return actions.map(a => {
                            const data = a.payload.doc.data() as User;
                            data.id = +a.payload.doc.id;
                            console.log(a.payload.doc.metadata.fromCache);
                            return data;
                     });
              }));
}
```

AngularFireDatabase

การทำงานกับ RTDB ต้องใช้ AngularFireDatabase ที่ทำงานได้แทบจะทันที หรือ เกือบเรียลไทม์ (real time) ซึ่งเหมาะกับงานประเภทโมบายล์แอปพลิเคชัน การดำเนินการ RTDB ทำงานกับข้อมูลในลักษณะออบเจ็กต์ และคอลเล็คชันของออบเจ็กต์ หรือ ลิสท์ (List)

สร้างฐานข้อมูล RTDB

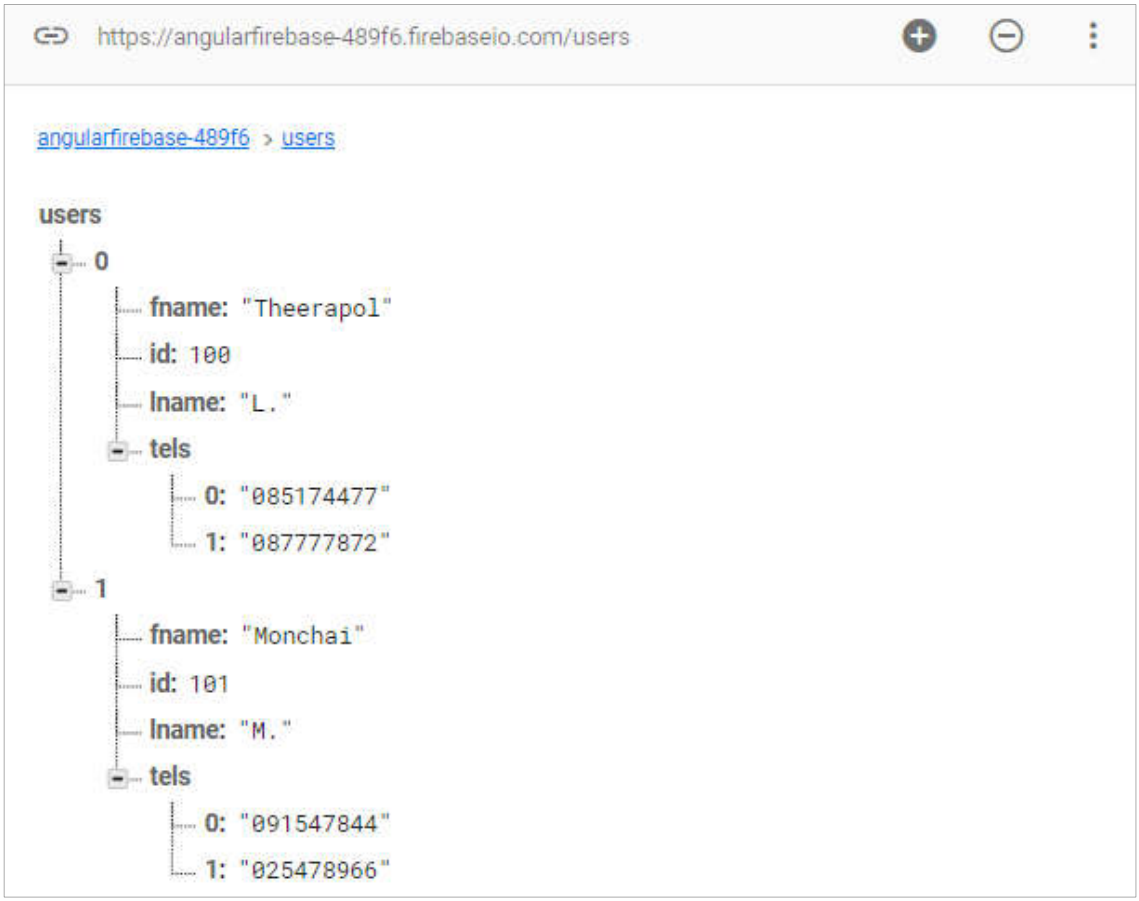
ก่อนการทำงานกับฐานข้อมูลได้ จะต้องสร้างฐานมูลของ RTDB ขึ้นมาก่อน โดยข้อมูล สมมุติให้มีลักษณะข้อมูลคล้ายกับที่เคยสร้างกับฐานข้อมูล CFire

สำหรับฐานข้อมูล RTDB มีลักษณะข้อมูลในรูปแบบ JSON บริสุทธิ์ ดังกำหนดให้มีลักษณะข้อมูลประกอบด้วยอาร์เรย์หรือลิสท์ของ JSON หรือจะสร้างให้เป็น JSON ที่ประกอบด้วยอาร์เรย์ก็ได้ หรือจะสร้างแบบ JSON ที่ประกอบด้วย JSON ขึ้นอยู่กับการออกแบบฐานข้อมูล แต่มีอยู่กหลักการอย่างนึ่ง ต้องทำให้ฐานข้อมูลแบบมากที่สุดเท่าที่ทำได้ เพราะต้องการลดความซับซ้อนในการสืบค้นให้มากที่สุด ดังสรุปเป็นสามแบบหลักคือ:

- แบบอาร์เรย์ของ JSON : `[ {}, {},..]`
- แบบออบเจ็กต์ JSON ของ JSON :`{ {}, {}, .. }`
- แบบออบเจ็กต์ JSON ของ JSON ของ JSON : `{ { {},{},..} }`

เราจะเริ่มจากการออกแบบแรกก่อน คือ ใช้รูปแบบ อาร์เรย์ของ JSON

```
[
      {
        "fname" : "Theerapol",
        "id" : 100,
        "lname" : "L.",
        "tels" : [ "085174477", "087777872" ]},
      {
        "fname" : "Monchai",
        "id" : 101,
        "lname": "M.",
        "tels" : [ "091547844", "025478966" ]}
]
```



รูป 4 ฐานข้อมูล RTDB เริ่มต้น แบบอาร์เรย์ของ JSON

เมื่อนำเข้าข้อมูล JSON เข้าสู่ RTDB จะมีลักษณะดัง **รูป 4** ให้สังเกตว่าการสร้างอาร์เรย์ จะต้องมีข้อมูลเริ่มต้นเป็น 0 เสมอ ถ้าไม่ใส่ 0 ฐานข้อมูลนี้จะกำหนดเป็นค่า NULL มาให้

การสร้างฐานข้อมูลนี้จะสร้างจากหน้าเว็บไซต์เองก็ได้ผ่านการคลิกเลือกที่ละออบเจ็กต์ หรือจะนำเข้าไฟล์ JSON ก็ได้ ผ่านรูปไข่ปลา (จุดสามจุดเรียงในแนวตั้งตาม **รูป 4**) ที่มุมขวาบนของหน้าต่างฐานข้อมูล

การอ่านข้อมูลในรูปแบบออบเจ็กต์

เองกูล่าร์ใช้บริการจาก AngularFireDatabase ซึ่งบริการนี้ผ่านการฉีดผ่านคอนสตรักเตอร์ของคอมโพเน้นท์ หรือใช้ฉีดเป็นงานบริการผ่าน @Injecting( ) ก็ได้

เริ่มจากต้องนำเข้า AngularFireDataase

```
import { AngularFireDatabase } from '@angular/fire/database';
```

ต่อมาฉีดบริการลงผ่านคอนสตรักเตอร์

```
constructor(private fireDb:AngularFireDatabase)
```

กรณีโปรแกรมเดิม มีการใช้บริการ AngualrFirestore ผ่านคอนสตรักเตอร์อยู่ จะเก็บไว้ดูเป็นตัวอย่างก็ได้ (ไม่ต้องลบของเดิมออก)

ในก่อนหน้านี้เราสร้างงานบริการ UsersService และใช้ฟังก์ชัน getUsers( ) เพื่ออ่านรายชื่อผู้ใช้ทั้งหมด มาคราวนี้เราเปลี่ยนมาใช้ฐานข้อมูล RTDB ดังนี้ฟังก์ชัน getUsers( ) จะเปลี่ยนแปลงดังนี้

Code 28. src/app/users.service.ts

```
getUsers():Observable<any[]>{
        return this.fireDb.list('users').valueChanges();
}
```

เมื่อบันทึกไฟล์นี้ เว็บแอปฯ ก็จะทำงานได้ผลเหมือนเดิมกับที่ใช้กับฐานข้อมูล CFire เพียงแต่ตอนนี้เลข id ได้แก้ไขเป็นเลข 100 และ 101 แทน

#	First Nmae	Last Name	Tel
100	Theerapol	L.	085174477 087777872
101	Monchai	M.	091547844 025478966

รูป 5 ผลการแสดงของฐานข้อมูล RTDB

นอกจากอ่านเป็น list( ) แล้วยังมีอีกวิธีคืออ่านเป็นออบเจ็กต์ เป็นการอ้างอิงชื่อในรูปออบเจ็กต์ ในฐานข้อมูลมีออบเจ็กต์ชื่อ users ซึ่งอยู่ในรูปอาร์เรย์ของ ออบเจ็กต์ JSON วิธีการต่อไปนี้ทำงานได้ผลเหมือนกับที่ใช้งานก่อนหน้านี้

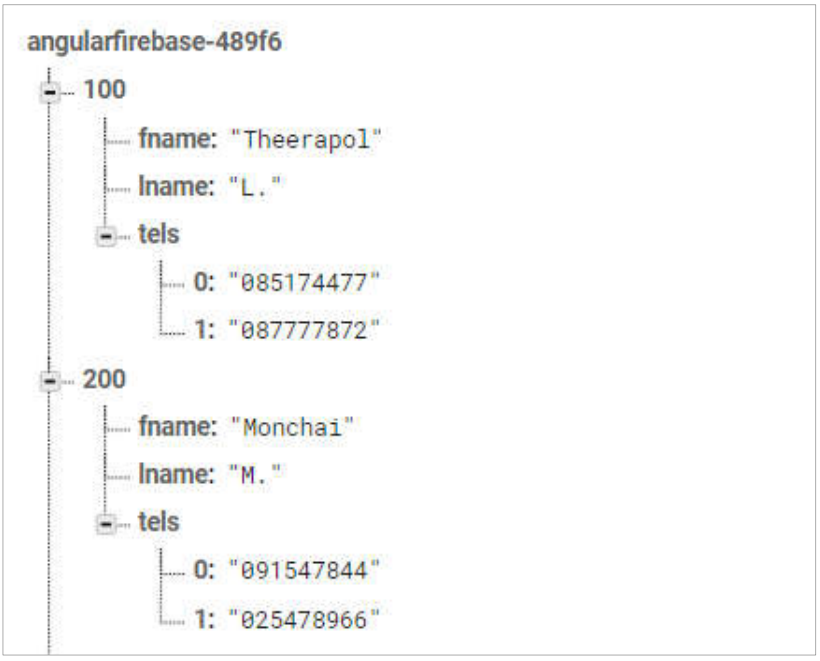
Code 29. src/app/users.service.ts

```
getUsers():Observable<User[]>{
          return this.fireDb.object<User[]>("users").valueChanges();
}
```

จากตัวอย่างนี้เป็นการอ่านแบบออบเจ็กต์ ซึ่งเหมาะที่ใช้การออกแบบฐานข้อมูลในรูปแบบออบเจ็กต์ แต่ที่ผ่านมาเป็นออกแบบอาร์เรย์ ซึ่งมีข้อเสีย กรณีที่ต้องการหาค่าเลข id หมายเลขใดต้องอ่านค่าทั้งหมดของอาร์เรย์ ทำให้สิ้นเปลี่ยงทรัพยากร

ตัวอย่างต่อไปนี้เป็นการออกแบบฐานข้อมูลในรูปแบบ ออบเจ็กต์ JSON แทนออบเจ็กต์ทั้งหมด การค้นหาจะใช้การอ่านค่าค่าคีย์ ซึ่งค่าคีย์ จะแทนแต่ละออบเจ็กต์ เช่น 100 แทน ออบเจ็กต์อีกออบเจ็กต์

```
{
  "100" : {
    "fname" : "Theerapol",
    "lname" : "L.",
    "tels" : [ "085174477", "087777872" ]
  },
  "200" : {
    "fname" : "Monchai",
    "lname" : "M.",
    "tels" : [ "091547844", "025478966" ]
  }
}
```



รูป 6 ฐานข้อมูล RTDB เริ่มต้น แบบ JSON ของ JSON

แต่ปัญหาหนึ่งของการสร้างฐานข้อมูลแบบนี้ จะไม่ตัวแปรออบเจ็กต์ใดแทนออบเจ็กต์ทุกตัว จึงนำมาซึ่งการออกแบบฐานข้อมูลใหม่

การออกแบบฐานข้อมูลแบบที่สาม คือ สร้างการอ้างอิงรวมทุกออบเจ็กต์ ในชื่อ db ซึ่งประกอบด้วย ออบเจ็กต์ JSON ของ JSON โดยให้คีย์เป็นค่า Id ที่ต้องการอ้างอิงออบเจ็กต์รายการของผู้ใช้แต่ละราย ดังตัวอย่างต่อไปนี้

```
{
  "db":{

  "100" : {
    "fname" : "Theerapol",
    "lname" : "L.",
    "tels" : [ "085174477", "087777872" ]
  },
  "200" : {
    "fname" : "Monchai",
    "lname" : "M.",
    "tels" : [ "091547844", "025478966" ]
  }
  }
}
```

รูป 7 ฐานข้อมูล RTDB เริ่มต้น แบบ JSON ของ JSON ของ JSON

จากฐานข้อมูลที่ออกแบบใหม่นี้ ใช้ชื่อออบเจ็กต์รวมในชื่อ db ทำให้ง่ายต่อการอ้างอิงฐานข้อมูล ดังตัวอย่างฟังก์ชัน getUsers( )ตัวใหม่นี้

Code 30. src/app/users.service.ts

```
getUsers():Observable<any>{
        return this.fireDb.object<any>("db").valueChanges();
}
```

เมื่อได้ออบเจ็กต์รวมทั้งฐานข้อมูล การอ่านงานบริการนี้ จะทำอย่างไร ที่จะทำให้ออบเจ็กต์รวมกลายเป็นอาร์เรย์ดังที่เคยใช้งานในก่อนหน้า

การวนซ้ำในออบเจ็กต์รวม และจัดระเบียนออบเจ็กต์ให้เป็นไปตามนิยามออบเจ็กต์ User ใช้เทคนิคพิเศษอีกนิดหน่อย คือ การอ่านค่าคุณสมบัติของคีย์ของฟังก์ชัน hasOwnProperty( ) และเนื่องจาก เราไม่ได้สร้างฐานข้อมูลให้มี id เป็นค่าหนึ่งของออบเจ็กต์ (มีเพียง fname, lname, tels) ทำให้ต้องดังแปลงออบเจ็กต์อีกเล็กน้อง (แต่ถ้าเราออกแบบใส่ id เป็นค่าคุณสมบัติหนึ่งก็ไม่ต้องดัดแปลงอะไร) ดังตัวอย่างต่อไปนี้

Code 31. src/app/users/users.components.ts

```
getUsers():void{
      this.usersService.getUsers().subscribe(users=>{
               for (var key in users) {
                    if (users.hasOwnProperty(key)) {
                          let user = { id:Number(key),
                                       fname:users[key].fname,
                                       lname:users[key].lname,
                                       tels:users[key].tels};
                          this.users.push(user);
                    }
               }
      });
}
```

สำหรับการอ่านค่า id ใดค่าหนึ่งทำได้ง่ายเพียงระบุค่าคีย์ id ที่เลือก ดังเช่นฟังก์ชัน getUser( ) ในคอมโพเน้นท์ UserComponent

Code 32. src/app/user/user.components.ts

```
getUser(): void {
        const id = this.route.snapshot.paramMap.get('id');
        this.usersService
              .getUsers()
              .subscribe(users => {
                    this.user= users[id];
              });
}
```

สำหรับฟังก์ชันนี้ ให้สังเกตุว่าเลือกใช้งานบริการ getusers( ) ซึ่งได้มาทุกออบเจ็กต์รวม แต่มาแยกเฉพาะ id ที่ต้องการ

การบันทึกข้อมูล

การในการบันทึกข้อมูลมีรูปแบบ 3 รูปแบบคือ การลบค่าเดิมแล้วแทนที่ค่าใหม่ (destructive) ใช้ฟังก์ชัน set(value:T) ส่วนการปรุงปรุงข้อมูลโดยไม่ได้ลบข้อมูลเก่าออก (non-destructive) ใช้ฟังก์ชัน update(value:T) และสุดท้ายการลบใช้ฟังก์ชัน remove( ) โดยทั้งสามฟังก์ชันใช้การอ้างอิงหรือคีย์ เช่น ในตัวอย่างใช้ db เป็นค่าคีย์ที่อ้างอิงออบเจ็กต์รวม

ตาราง 2 ฟังก์ชันในการบันทึกข้อมูล

ฟังก์ชัน	ความหมาย	ตัวอย่าง

set(value:T)	การแทนที่ข้อมูลเดิม (ลบ แล้วกำหนดค่าขึ้นมาใหม่)	const itemRef = db.object('item'); itemRef.set({name:'name'});
update(value:T)	การปรับปรุงข้อมูลเดิม	itemRef.update({name:'new name'});
remove( )	การลบ	itemRef.remove();

เมื่อใช้ db แทนการอ้างอิงออบเจ็กต์รวม ทำให้เลือกต่อไปได้ว่าจะอ้างอิงออบเจ็กต์ใดในออบเจ็กต์รวม เช่น 100 เป็นคีย์แรกในฐานข้อมูลนี้ การอ้างอิง 100 ผ่าน db จึงนำไปดำเนินการบันทึกข้อมูลได้ ดังตัวอย่างต่อไปนี้

Code 33. src/app/users.service.ts

```
updateUser(user:User):void{
          const userRef = this.fireDb.object<any>('db');
          userRef.update({'100':{fname:'Tee', lname:'E.', tels:[]});
}
```

สรุป

ได้ทดลองใช้งานฐานข้อมูลของไฟร์เบส ทั้งใน CFire และ RTDB ทำให้รู้สึกได้เลยว่าทำไมจึงต้องมี CFire เพื่อเป็นทางเลือกอีกตัวหนึ่ง ส่วนหนึ่งเพราะข้อจำกัดของลักษณะไฟล์ JSON มีการอ้างอิงในลำดับชั้นที่ทำได้ยากกว่า ยิ่งมีจำนวนชั้นลึกมาก ก็จะยิ่งทำได้ลำบาก ในขณะฐานข้อมูลในลักษณะเอกสารของ CFire มีความหลากหลายในการใช้งานมากกว่า การอ้างอิงมีชั้นของเอกสารเป็นตัวขั้นอีกต่อหนึ่งจึงเป็นทางเลือกอีกอย่างหนึ่งนอกจากการอ้างอิงรูปแบบ JSON เพียงอย่างเดียว

แบบฝึกหัด

1. สร้างหน้าแสดงผลของ UserComponent ตามผลการคลิกจากตารางของ หน้า UsersComponent
2. จากตัวอย่างที่แสดงเรื่อง คอลเล็คชัน users/2 ให้เพิ่มการปรังปรุงข้อมูลของคอลเล็คชัน tels ของหน้า user.component.ts
3. ให้สร้างการสืบค้นผ่านฟิลด์ id นอกจาก fname, lname (ข้อระวัง id เป็นไทป์ เป็นตัวเลข จะต้องแปลงให้เป็นอักษรหรือ string ก่อน)

อ้างอิง/อ่านเพิ่มเติม

- Github. angular/angularfire. https://github.com/angular/angularfire
- Firebase. Documentation. https://firebase.google.com/docs/guides