INF 354 Notes

Ionic and reporting

Contents

Ionic

Ionic installation initial:

- To install CLI: npm install -g @ionic/cli
- To create a project: ionic start projectname template
- To create an angular ionic app: ionic start projectname tabs –type=angular
- Test in browser: ionic serve
- All templates available: ionic start –list

- Generate: ionic generate

Links:

Starting point:

https://ionicframework.com/docs/intro/cli

Framework components (button, card, modal…):

https://ionicframework.com/docs/components

Example

An example implementation of the `getExpensesByCategory()` function in the `expense.service.ts` file based on the provided requirements:

```typescript
import { Injectable } from '@angular/core';
import { Storage } from '@ionic/storage';
import { Observable, from } from 'rxjs';
import { map } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class ExpenseService {
  constructor(private storage: Storage) {}

  getExpensesByCategory(): Observable<Map<string, Expense[]>> {
```

```
    return from(this.storage.get('expenses')).pipe(
      map((expenses: Expense[]) => {
        const groupedExpenses = new Map<string, Expense[]>();

        if (expenses) {
          for (const expense of expenses) {
            const category = expense.category;
            if (groupedExpenses.has(category)) {
              groupedExpenses.get(category)?.push(expense);
            } else {
              groupedExpenses.set(category, [expense]);
            }
          }
        }

        return groupedExpenses;
      })
    );
  }
}
```

In the above code:

1. We inject the `Storage` service from Ionic into the `ExpenseService` class through the constructor.

2. The `getExpensesByCategory()` function retrieves all expenses from storage using `this.storage.get('expenses')`. It returns a promise, so we convert it into an observable using `from()` from the RxJS library.

3. We then use the `map()` operator to transform the array of expenses into a `Map<string, Expense[]>` object where each key is an expense category and each value is an array of expenses.

4. Inside the `map()` function, we iterate over the expenses and group them by category using a `Map`. If a category already exists in the map, we append the expense to the existing array. Otherwise, we create a new entry in the map with the category as the key and an array containing the expense.

5. Finally, we return the resultant `groupedExpenses` map as an observable.

```
<ion-header>
  <ion-toolbar>
    <ion-title>Question 2</ion-title>
  </ion-toolbar>
</ion-header>


<ion-content>
  <ion-searchbar [(ngModel)]="searchText" (ionChange)="filterExpenses()"></ion-searchbar>


  <ion-list>
    <ion-item *ngFor="let expense of filteredExpenses">
      <ion-label>{{ expense.title }}</ion-label>
      <ion-label>{{ expense.amount }}</ion-label>
    </ion-item>
  </ion-list>
</ion-content>
```

In the above code:

1. We have an ion-header component with a title displayed in an ion-toolbar.
2. Inside the ion-content component, we have an ion-searchbar component. The [(ngModel)] directive is used to bind the entered search text to the searchText property in the controller. The (ionChange) event is triggered whenever the search text changes, and it calls the filterExpenses() method in the controller.
3. The main content is displayed in an ion-list component. We use *ngFor directive to iterate over the filteredExpenses array in the controller and generate an ion-item for each expense.
4. Inside each ion-item, we display the title and amount properties of the expense using ion-label components. You can adjust the displayed properties as needed based on your Expense object structure.

```typescript
import { Component } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { ExpenseService } from 'path/to/expense.service';
import { ToastController } from '@ionic/angular';

@Component({
  selector: 'app-question3',
  templateUrl: './question3.page.html',
  styleUrls: ['./question3.page.scss'],
})
export class Question3Page {
  expenseForm: FormGroup;

  constructor(
    private formBuilder: FormBuilder,
    private expenseService: ExpenseService,
    private toastController: ToastController
  ) {
    this.expenseForm = this.formBuilder.group({
      title: ['', Validators.required],
      amount: ['', [Validators.required, Validators.min(1)]],
      category: ['', Validators.required],
    });
  }

  async addExpense() {
    if (this.expenseForm.valid) {
      const expense = {
        title: this.expenseForm.value.title,
```

```
      amount: this.expenseForm.value.amount,
      category: this.expenseForm.value.category,
    };

    this.expenseService.addExpense(expense).subscribe(
      () => {
        this.expenseForm.reset();
        this.presentToast('Expense added successfully');
      },
      (error) => {
        console.error('Error adding expense:', error);
        this.presentToast('Error adding expense. Please try again.');
      }
    );
  }
 }

 async presentToast(message: string) {
  const toast = await this.toastController.create({
    message: message,
    duration: 2000,
    position: 'bottom',
  });
  toast.present();
 }
}
```

In the above code:

1. We import the necessary modules, including `FormBuilder` and `FormGroup` from `@angular/forms`, `ExpenseService` from the appropriate path, and `ToastController` from `@ionic/angular`.

2. The `expenseForm` property is declared as a `FormGroup` to handle the reactive form with validation. We use the `formBuilder` to create the form group and define the form controls with their initial values and validation rules.

3. The `addExpense()` function is defined to handle the addition of expenses. It checks if the form is valid and then retrieves the entered form values to create an `expense` object. It calls the `addExpense()` method from the `ExpenseService` and subscribes to the observable returned by it. If the addition is successful, the form is reset and a toast message is displayed. If there's an error, an error message is logged, and an error toast is presented.

4. The `presentToast()` function is defined to create and display toast messages to the user. It uses the `toastController` to create a toast with the provided message, duration, and position.

Integrate Ionic and Chart.js:

1. Install Chart.js: In your Ionic project directory, install Chart.js using npm by running `npm install chart.js`.

2. Create a page/component: Generate a new page or component in Ionic where you want to display the chart. You can use the Ionic CLI to create a new page by running `ionic generate page ChartPage`.

3. Import Chart.js: In the component file (e.g., `chart-page.ts`), import Chart.js using the following line:

```typescript
import Chart from 'chart.js';
```

4. Create a canvas element: In the HTML template file (e.g., `chart-page.html`), add a canvas element where the chart will be rendered. Give it an `id` so that we can reference it in the component file.

```html
<ion-content>
  <canvas id="myChart"></canvas>
</ion-content>
```

5. Initialize Chart.js: In the component file (`chart-page.ts`), initialize the chart by getting the canvas element and creating a new Chart instance.

```typescript
import { Component, OnInit } from '@angular/core';
import Chart from 'chart.js';
```

```
@Component({
  selector: 'app-chart-page',
  templateUrl: './chart-page.html',
  styleUrls: ['./chart-page.scss'],
})
export class ChartPage implements OnInit {
  constructor() {}

  ngOnInit() {
    const canvas = document.getElementById('myChart') as HTMLCanvasElement;
    const ctx = canvas.getContext('2d');
    const chart = new Chart(ctx, {
      type: 'bar', // Set the chart type according to your needs
      data: {
        // Provide the data for your chart
      },
      options: {
        // Configure the options for your chart (e.g., title, axes, tooltips)
      },
    });
  }
```

6. Display the page: Navigate to the page where you want to display the chart. You can do this by updating the app-routing.module.ts file to include a route for your page and navigating to it using Ionic's navigation system.

Links:

Chart.js:

https://www.chartjs.org/docs/latest/

With a separate chart data file:

```
import { Component, OnInit } from '@angular/core';
import Chart from 'chart.js';
import { chartData } from './chart-data';

@Component({
  selector: 'app-chart-page',
  templateUrl: './chart-page.html',
  styleUrls: ['./chart-page.scss'],
})
export class ChartPage implements OnInit {
  constructor() {}

  ngOnInit() {
    const canvas = document.getElementById('myChart') as HTMLCanvasElement;
    const ctx = canvas.getContext('2d');
    const chart = new Chart(ctx, {
      type: 'bar',
      data: chartData,
      options: {
        // Configure the options for your chart
      },
    });
  }
}
```

Multiple different charts with same data:

Html:

```
<ion-header>
  <ion-toolbar>
```

```
    <ion-title>
      Chart Page
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-segment [(ngModel)]="selectedChart" (ionChange)= “renderChart()”>
    <ion-segment-button value="bar">
      Bar Chart
    </ion-segment-button>
    <ion-segment-button value="line">
      Line Chart
    </ion-segment-button>
    <ion-segment-button value="pie">
      Pie Chart
    </ion-segment-button>
  </ion-segment>

  <canvas id="myChart"></canvas>
</ion-content>
```

Typescript:

```
import { Component, OnInit } from '@angular/core';
import Chart from 'chart.js';
import { chartData } from './chart-data';

@Component({
  selector: 'app-chart-page',
  templateUrl: './chart-page.html',
```

```
  styleUrls: ['./chart-page.scss'],
})
export class ChartPage implements OnInit {
  selectedChart = 'bar';

  constructor() {}

  ngOnInit() {
    this.renderChart();
  }

  renderChart() {
    const canvas = document.getElementById('myChart') as HTMLCanvasElement;
    const ctx = canvas.getContext('2d');

    // Render the selected chart
    if (this.selectedChart === 'bar') {
      window.myChart = new Chart(ctx, {
        type: 'bar',
        data: chartData,
        options: {
          // Configure options for the bar chart
        },
      });
    } else if (this.selectedChart === 'line') {
      window.myChart = new Chart(ctx, {
        type: 'line',
        data: chartData,
        options: {
          // Configure options for the line chart
```

```
    },
  });
} else if (this.selectedChart === 'pie') {
  window.myChart = new Chart(ctx, {
    type: 'pie',
    data: chartData,
    options: {
      // Configure options for the pie chart
    },
  });
```

Downloading chart to PDF:

1. npm install html2canvas jspdf

2. ```
   import { Component, OnInit } from '@angular/core';
   import Chart from 'chart.js';
   import { chartData } from './chart-data';
   import html2canvas from 'html2canvas';
   import jsPDF from 'jspdf';
   ```

3. ```
   function openPDF(contentDataUrl: string, fileWidth: number, fileHeight: number) {
    const PDF = new jsPDF({
     orientation: 'portrait',
     unit: 'mm',
     format: 'a4',
    });

    const topPosition = 10;
    const leftPosition = 0;

    PDF.addImage(contentDataUrl, 'PNG', leftPosition, topPosition, fileWidth,
   fileHeight);
    PDF.save('Graph.pdf');
   }

   OR

    downloadChart() {
     const doc = new jsPDF();
     if (this.barChartCanvas && this.lineChartCanvas) {
      const barChartImageURI =
   this.barChartCanvas.nativeElement.toDataURL('image/png');
      const lineChartImageURI =
   this.lineChartCanvas.nativeElement.toDataURL('image/png');
      doc.addImage(barChartImageURI, 'PNG', 10, 10, 190, 100);
      doc.addPage();
      doc.addImage(lineChartImageURI, 'PNG', 10, 10, 190, 100);
      doc.save('charts.pdf');
     }
    }
   ```

4. ```
   <ion-content>
    <!-- Chart and other elements -->

    <ion-button (click)="downloadChartAsPDF()">Download as PDF</ion-button>
   </ion-content>
   ```
   ```

Reporting Example

`home.page.ts`

```typescript
import { Component } from '@angular/core';
import { ActionSheetController } from '@ionic/angular';
import { ProductData } from '../productData';

@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
export class HomePage {
  products: string[] = [];
  values: number[] = [];

  barChartData: any[] = [];
  barChartLabels: string[] = [];

  constructor(private actionSheetController: ActionSheetController) {}

  async presentActionSheet() {
    const actionSheet = await this.actionSheetController.create({
      header: 'Data Action Sheet',
      buttons: [
        {
          text: 'Show Data',
          icon: 'analytics-outline',
          handler: () => {
            this.generateReport();
          }
```

```
      },
      {
        text: 'Cancel',
        icon: 'close',
        role: 'cancel'
      }
    ]
  });

  await actionSheet.present();
}

generateReport() {
  this.products = ProductData.map((product) => product.name);
  this.values = ProductData.map((product) => product.value);

  this.barChartData.push({
    data: this.values,
    label: 'Sales',
    backgroundColor: 'rgb(0,183,255)'
  });

  this.barChartLabels = this.products;

  this.generateTable();
}

generateTable() {
  // Function to generate the table based on the data
  // Not provided in the instructions, please implement accordingly }}
```

In the above code:

1. The `presentActionSheet()` function creates an action sheet controller using the `ActionSheetController` from `@ionic/angular`. It sets the header to 'Data Action Sheet' and defines two buttons: 'Show Data' and 'Cancel'. The 'Show Data' button has an icon and a handler that calls the `generateReport()` function. The 'Cancel' button has a cancel role. Finally, the action sheet is presented.

2. The `generateReport()` function extracts the product names and values from the `ProductData` array. It assigns the names to the `products` array and the values to the `values` array. Then, it pushes an object to the `barChartData` array with the values set as the `data` property, the label set as 'Sales', and the `backgroundColor` set to 'rgb(0,183,255)'. The `barChartLabels` array is set to the `products` array. Finally, it calls the `generateTable()` function.

3. The `generateTable()` function is not provided in the instructions. You should implement it based on your specific requirements to generate the table based on the data.

Reporting Example II

To add the provided data to a radar chart and print it to a PDF using `jsPDF`, you can follow the steps outlined below:

1. Import the necessary modules and dependencies:

```typescript
import { Component, ViewChild, ElementRef } from '@angular/core';
import { Chart, registerables } from 'chart.js';
import jsPDF from 'jspdf';
import html2canvas from 'html2canvas';
```

2. Register the necessary Chart.js components:

```typescript
Chart.register(...registerables);
```

3. Define the chart options:

```typescript
const chartOptions = {
```

```
  responsive: true,
  maintainAspectRatio: false,
  scales: {
    r: {
      beginAtZero: true,
    },
  },
};
```

4. Define the chart data:

```typescript
const chartData = {
  labels: ['Point A', 'Point B', 'Point C', 'Point D', 'Point E', 'Point F'],
  datasets: [
    {
      label: 'Series A',
      data: [6, 3, 2, 2, 2, 3],
      fill: true,
      backgroundColor: 'rgba(255, 99, 132, 0.2)',
      borderColor: 'rgb(255, 99, 132)',
      pointBackgroundColor: 'rgb(255, 99, 132)',
      pointBorderColor: '#fff',
      pointHoverBackgroundColor: '#fff',
      pointHoverBorderColor: 'rgb(255, 99, 132)',
    },
    {
      label: 'Series B',
      data: [2, 3, 6, 3, 2, 2],
      fill: true,
```

```
      backgroundColor: 'rgba(54, 162, 235, 0.2)',
      borderColor: 'rgb(54, 162, 235)',
      pointBackgroundColor: 'rgb(54, 162, 235)',
      pointBorderColor: '#fff',
      pointHoverBackgroundColor: '#fff',
      pointHoverBorderColor: 'rgb(54, 162, 235)',
    },
    {
      label: 'Series C',
      data: [2, 2, 2, 3, 6, 3],
      fill: true,
      backgroundColor: 'rgba(75, 192, 192, 0.2)',
      borderColor: 'rgb(75, 192, 192)',
      pointBackgroundColor: 'rgb(75, 192, 192)',
      pointBorderColor: '#fff',
      pointHoverBackgroundColor: '#fff',
      pointHoverBorderColor: 'rgb(75, 192, 192)',
    },
  ],
};
```

5. Add the necessary code to generate the radar chart and print it to PDF:

```typescript
@Component({
  selector: 'app-radar-chart',
  templateUrl: 'radar-chart.component.html',
  styleUrls: ['radar-chart.component.scss'],
})
export class RadarChartComponent {
```

```
  @ViewChild('radarChart') radarChartRef!: ElementRef;

  generateRadarChart() {
    const canvasElement = this.radarChartRef.nativeElement;

    const chart = new Chart(canvasElement.getContext('2d'), {
      type: 'radar',
      data: chartData,
      options: chartOptions,
    });

    html2canvas(canvasElement).then((canvas) => {
      const imgData = canvas.toDataURL('image/png');
      const pdf = new jsPDF();

      const pdfWidth = pdf.internal.pageSize.getWidth();
      const pdfHeight = pdf.internal.pageSize.getHeight();

      pdf.addImage(imgData, 'PNG', 0, 0, pdfWidth, pdfHeight);
      pdf.save('radar_chart.pdf');
    });
  }
}


```

6. Make sure to include the `RadarChartComponent` in your module and template files.

In the above code, the `generateRadarChart()` function is triggered to generate the radar chart. It creates a `Chart` instance using the provided data and options. Then, it uses `html2canvas` to convert the chart canvas to an image. Finally, it creates a new `jsPDF` instance, adds the image to the PDF, and saves it as `radar_chart.pdf`.