Source Code for OnlineTestApplication

App.component.html=>

```
<div class="container">
  <app-quiz></app-quiz>
</div>
```

App.component.spec.ts=>

```
import { TestBed, waitForAsync } from '@angular/core/testing';
import { AppComponent } from './app.component';
describe('AppComponent', () => {
  beforeEach(waitForAsync(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  }));

  it('should create the app', waitForAsync(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));
  it(`should have as title 'app'`, waitForAsync(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app.title).toEqual('app');
  }));
  it('should render title in a h1 tag', waitForAsync(() => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('Welcome to app!');
  }));
});
```

App.component.ts=>

```
import { Component } from '@angular/core';

import { QuizComponent } from './quiz/quiz.component';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

```
export class AppComponent {
}
```

App.module.ts=>

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { QuizComponent } from './quiz/quiz.component';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [
    AppComponent,
    QuizComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Quiz.component.html=>

```
<div class="row">
    <div class="col-6">
        <h3>Online Test Application</h3>
    </div>
    <div class="col-6 text-right">
        Select Quiz:
        <select [(ngModel)]="quizName" (change)="loadQuiz(quizName)">
      <option *ngFor="let quiz of quizes" [value]="quiz.id">{{quiz.name}}</option>
    </select>
    </div>
</div>
<div id="quiz">
    <h2 class="text-center font-weight-normal">{{quiz.name}}</h2>
    <hr />

    <div *ngIf="mode=='quiz' && quiz">
        <div *ngFor="let question of filteredQuestions;">
            <div class="badge badge-info">Question {{pager.index + 1}} of
{{pager.count}}.</div>
            <div *ngIf="config.duration" class="badge badge-info float-right">Time:
{{ellapsedTime}} / {{duration}}</div>
            <h3 class="font-weight-normal">{{pager.index + 1}}.
                <span [innerHTML]="question.name"></span>
            </h3>
            <div class="row text-left options">
                <div class="col-6" *ngFor="let option of question.options">
                    <div class="option">
                        <label class="font-weight-normal" [attr.for]="option.id">
              <input id="{{option.id}}" type="checkbox" [(ngModel)]="option.selected"
(change)="onSelect(question, option);" /> {{option.name}}
            </label>
                    </div>
                </div>
            </div>
        </div>
        <hr />
        <div class="text-center">
            <button class="btn btn-default" *ngIf="config.allowBack"
(click)="goTo(0);">First</button>
            <button class="btn btn-default" *ngIf="config.allowBack"
(click)="goTo(pager.index - 1);">Prev</button>
            <button class="btn btn-primary" (click)="goTo(pager.index +
1);">Next</button>
            <button class="btn btn-default" *ngIf="config.allowBack"
(click)="goTo(pager.count - 1);">Last</button>
            <!--<pagination *ngIf="config.showPager" direction-links="false" total-
items="totalItems" items-per-page="itemsPerPage" ng-model="currentPage" ng-
change="pageChanged()"></pagination>-->
        </div>
    </div>

    <div class="row text-center" *ngIf="mode=='review'">
        <div class="col-4 cursor-pointer" *ngFor="let question of quiz.questions; let
index = index;">
            <div (click)="goTo(index)" class="p-3 mb-2 {{ isAnswered(question) ==
'Answered'? 'bg-info': 'bg-warning' }}">{{index + 1}}. {{ isAnswered(question)
}}</div>
        </div>
    </div>
    <div class="result" *ngIf="mode=='result'">
        <h2>Quiz Result</h2>
        <div *ngFor="let question of quiz.questions; let index = index">
            <div class="result-question">
                <h5>{{index + 1}}. {{question.name}}</h5>
                <div class="row">
                    <div class="col-6" *ngFor="let Option of question.options">
                        <input id="{{Option.id}}" type="checkbox" disabled="disabled"
[(ngModel)]="Option.selected" /> {{Option.name}}
                    </div>
                </div>
                <div class="p-1 m-2 alert {{ isCorrect(question) == 'correct'? 'alert-
success': 'alert-danger'}}">Your answer is {{isCorrect(question)}}.</div>
            </div>
        </div>
        <h4 class="alert alert-info text-center">You may close this window now.</h4>
    </div>
    <hr />
    <div *ngIf="mode!='result'">
        <button class="btn btn-warning" (click)="mode = 'quiz'">Quiz</button>
        <button class="btn btn-info" (click)="mode = 'review'">Review</button>
        <button class="btn btn-primary" (click)="onSubmit();">Submit Quiz</button>
    </div>
</div>
```

Quiz.component.spec.ts=>

```
/* tslint:disable:no-unused-variable */
import { ComponentFixture, TestBed, waitForAsync } from '@angular/core/testing';
import { By } from '@angular/platform-browser';
import { DebugElement } from '@angular/core';

import { QuizComponent } from './quiz.component';

describe('QuizComponent', () => {
  let component: QuizComponent;
  let fixture: ComponentFixture<QuizComponent>;

  beforeEach(waitForAsync(() => {
    TestBed.configureTestingModule({
      declarations: [ QuizComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(QuizComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Quiz.component.ts=>

```
import { Component, OnInit } from '@angular/core';

import { QuizService } from '../services/quiz.service';
import { HelperService } from '../services/helper.service';
import { Option, Question, Quiz, QuizConfig } from '../models/index';

@Component({
  selector: 'app-quiz',
  templateUrl: './quiz.component.html',
  styleUrls: ['./quiz.component.css'],
  providers: [QuizService]
})
```

```
export class QuizComponent implements OnInit {
  quizes: any[];
  quiz: Quiz = new Quiz(null);
  mode = 'quiz';
  quizName: string;
  config: QuizConfig = {
    'allowBack': true,
    'allowReview': true,
    'autoMove': false,  // if true, it will move to next question automatically when
answered.
    'duration': 300,  // indicates the time (in secs) in which quiz needs to be
completed. 0 means unlimited.
    'pageSize': 1,
    'requiredAll': false,  // indicates if you must answer all the questions before
submitting.
    'richText': false,
    'shuffleQuestions': false,
    'shuffleOptions': false,
    'showClock': false,
    'showPager': true,
    'theme': 'none'
  };

  pager = {
    index: 0,
    size: 1,
    count: 1
  };
  timer: any = null;
  startTime: Date;
  endTime: Date;
  ellapsedTime = '00:00';
  duration = '';

  constructor(private quizService: QuizService) { }

  ngOnInit() {
    this.quizes = this.quizService.getAll();
    this.quizName = this.quizes[0].id;
    this.loadQuiz(this.quizName);
  }

  loadQuiz(quizName: string) {
    this.quizService.get(quizName).subscribe(res => {
      this.quiz = new Quiz(res);
      this.pager.count = this.quiz.questions.length;
      this.startTime = new Date();
      this.ellapsedTime = '00:00';
      this.timer = setInterval(() => { this.tick(); }, 1000);
      this.duration = this.parseTime(this.config.duration);
    });
    this.mode = 'quiz';
  }
  tick() {
    const now = new Date();
    const diff = (now.getTime() - this.startTime.getTime()) / 1000;
    if (diff >= this.config.duration) {
      this.onSubmit();
    }
    this.ellapsedTime = this.parseTime(diff);
  }

  parseTime(totalSeconds: number) {
    let mins: string | number = Math.floor(totalSeconds / 60);
    let secs: string | number = Math.round(totalSeconds % 60);
    mins = (mins < 10 ? '0' : '') + mins;
    secs = (secs < 10 ? '0' : '') + secs;
    return `${mins}:${secs}`;
  }

  get filteredQuestions() {
    return (this.quiz.questions) ?
      this.quiz.questions.slice(this.pager.index, this.pager.index + this.pager.size)
: [];
  }

  onSelect(question: Question, option: Option) {
    if (question.questionTypeId === 1) {
      question.options.forEach((x) => { if (x.id !== option.id) x.selected = false;
});
    }

    if (this.config.autoMove) {
      this.goTo(this.pager.index + 1);
    }
  }

  goTo(index: number) {
    if (index >= 0 && index < this.pager.count) {
      this.pager.index = index;
      this.mode = 'quiz';
    }
  }

  isAnswered(question: Question) {
    return question.options.find(x => x.selected) ? 'Answered' : 'Not Answered';
  };

  isCorrect(question: Question) {
    return question.options.every(x => x.selected === x.isAnswer) ? 'correct' :
'wrong';
  };

  onSubmit() {
    let answers = [];
    this.quiz.questions.forEach(x => answers.push({ 'quizId': this.quiz.id,
'questionId': x.id, 'answered': x.answered }));
```

```
    // Post your data to the server here. answers contains the questionId and the 
users' answer.
    console.log(this.quiz.questions);
    this.mode = 'result';
  }
}
```

Help.services.spec.ts=>

```
/* tslint:disable:no-unused-variable */

import { TestBed, inject, waitForAsync } from '@angular/core/testing';
import { HelperService } from './helper.service';

describe('HelperService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [HelperService]
    });
  });

  it('should ...', inject([HelperService], (service: HelperService) => {
    expect(service).toBeTruthy();
  }));
});
```

Helper.service.ts=>

```
import { Injectable } from '@angular/core';

@Injectable()
export class HelperService {
  static toBool(val) {
    if (val === undefined || val === null || val === '' || val === 'false' || 
val === 'False') {
      return false;
    } else {
      return true;
    }
  }

  static shuffle(array) {
    let currentIndex = array.length, temp, randomIndex;

    while (0 !== currentIndex) {
      randomIndex = Math.floor(Math.random() * currentIndex);
      currentIndex -= 1;

      temp = array[currentIndex];
      array[currentIndex] = array[randomIndex];
      array[randomIndex] = temp;
    }
    return array;
  }
  static extend(out) {
    out = out || {};

    for (let i = 1; i < arguments.length; i++) {
      if (!arguments[i]) {
        continue;
      }

      for (const key in arguments[i]) {
        if (arguments[i].hasOwnProperty(key)) {
          out[key] = arguments[i][key];
        }
      }
    }
    return out;
  }
}
```

Quiz.service.ts=>

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

@Injectable()
export class QuizService {

  constructor(private http: HttpClient) { }

  get(url: string) {
    return this.http.get(url);
  }

  getAll() {
    return [
      { id: 'data/javascript.json', name: 'JavaScript' },
      { id: 'data/aspnet.json', name: 'Asp.Net' },
      { id: 'data/csharp.json', name: 'C Sharp' },
      { id: 'data/designPatterns.json', name: 'Design Patterns' }
    ];
  }

}
```

Models===

Index.ts=>

```
export * from './option';

export * from './question';
export * from './quiz';
export * from './quiz-config';
```

option.ts=>

```
export class Option {
    id: number;
    questionId: number;
    name: string;
    isAnswer: boolean;
    selected: boolean;

    constructor(data: any) {
        data = data || {};
        this.id = data.id;
        this.questionId = data.questionId;
        this.name = data.name;
        this.isAnswer = data.isAnswer;
    }
}
```

Question.ts=>

```
import { Option } from './option';

export class Question {
    id: number;
    name: string;
    questionTypeId: number;
    options: Option[];
    answered: boolean;

    constructor(data: any) {
        data = data || {};
        this.id = data.id;
        this.name = data.name;
        this.questionTypeId = data.questionTypeId;
        this.options = [];
        data.options.forEach(o => {
            this.options.push(new Option(o));
        });
    }
}
```

Quiz-config.ts=>

```
export class QuizConfig {
    allowBack: boolean;
    allowReview: boolean;
    autoMove: boolean;  // if boolean; it will move to next question
automatically when answered.
    duration: number;  // indicates the time in which quiz needs to be
completed. 0 means unlimited.
    pageSize: number;
    requiredAll: boolean;  // indicates if you must answer all the questions
before submitting.
    richText: boolean;
    shuffleQuestions: boolean;
    shuffleOptions: boolean;
    showClock: boolean;
    showPager: boolean;
    theme: string;

    constructor(data: any) {
        data = data || {};
        this.allowBack = data.allowBack;
        this.allowReview = data.allowReview;
        this.autoMove = data.autoMove;
        this.duration = data.duration;
        this.pageSize = data.pageSize;
        this.requiredAll = data.requiredAll;
        this.richText = data.richText;
        this.shuffleQuestions = data.shuffleQuestions;
        this.shuffleOptions = data.shuffleOptions;
        this.showClock = data.showClock;
        this.showPager = data.showPager;
    }
}
```

Quiz.ts=>

```
import { QuizConfig } from './quiz-config';
import { Question } from './question';
```

```
export class Quiz {
    id: number;
    name: string;
    description: string;
    config: QuizConfig;
    questions: Question[];

    constructor(data: any) {
        if (data) {
            this.id = data.id;
            this.name = data.name;
            this.description = data.description;
            this.config = new QuizConfig(data.config);
            this.questions = [];
            data.questions.forEach(q => {
                this.questions.push(new Question(q));
            });
        }
    }
}
```

Data===

Javascripts.json=>

```
{
    "id": 1,
    "name": "JavaScript Quiz",
    "description": "JavaScript Quiz (Basic Multiple Choice Questions for
JavaScript Developers)",
    "questions": [
        {
            "id": 1010,
            "name": "Which HTML tag do we use to put the JavaScript?",
            "questionTypeId": 1,
            "options": [
                {
                    "id": 1055,
                    "questionId": 1010,
                    "name": "<javascript>",
                    "isAnswer": false
                },
                {
                    "id": 1056,
                    "questionId": 1010,
                    "name": "<script>",
                    "isAnswer": true
                },
```

```json
                {
                    "id": 1057,
                    "questionId": 1010,
                    "name": "<js>",
                    "isAnswer": false
                },
                {
                    "id": 1058,
                    "questionId": 1010,
                    "name": "None of the above",
                    "isAnswer": false
                }
            ],
            "questionType": {
                "id": 1,
                "name": "Multiple Choice",
                "isActive": true
            }
        },
        {
            "id": 1011,
            "name": "Which built-in method calls a function for each element in the array?",
            "questionTypeId": 1,
            "options": [
                {
                    "id": 1055,
                    "questionId": 1010,
                    "name": "while()",
                    "isAnswer": false
                },
                {
                    "id": 1057,
                    "questionId": 1010,
                    "name": "loop",
                    "isAnswer": false
                },
                {
                    "id": 1056,
                    "questionId": 1010,
                    "name": "forEach",
                    "isAnswer": true
                },
                {
                    "id": 1058,
                    "questionId": 1010,
                    "name": "takeUntil",
                    "isAnswer": false
                }
            ],
            "questionType": {
                "id": 1,
                "name": "Multiple Choice",
                "isActive": true
            }
        },
        {
            "id": 1012,
            "name": "What is the difference between let and var?",
            "questionTypeId": 1,
            "options": [
                {
                    "id": 1055,
                    "questionId": 1010,
                    "name": "let has local scope",
                    "isAnswer": true
                },
                {
                    "id": 1057,
                    "questionId": 1010,
                    "name": "Both are same",
                    "isAnswer": false
                },
                {
                    "id": 1056,
                    "questionId": 1010,
                    "name": "var is new data type",
                    "isAnswer": false
                },
                {
                    "id": 1058,
                    "questionId": 1010,
                    "name": "let consumes more cpu and ram",
                    "isAnswer": false
                }
            ],
            "questionType": {
                "id": 1,
                "name": "Multiple Choice",
                "isActive": true
            }
        },
        {
            "id": 1013,
            "name": "What is TypeScript?",
            "questionTypeId": 1,
            "options": [
                {
                    "id": 1055,
                    "questionId": 1010,
                    "name": "A Language based on Javascript",
                    "isAnswer": true
                },
                {
                    "id": 1057,
                    "questionId": 1010,
                    "name": "script that runs on browser",
                    "isAnswer": false
                },
                {
                    "id": 1056,
                    "questionId": 1010,
                    "name": "A DataType Collection of Javascript",
                    "isAnswer": false
                },
                {
                    "id": 1058,
                    "questionId": 1010,
                    "name": "None of the above",
                    "isAnswer": false
                }
            ],
            "questionType": {
                "id": 1,
                "name": "Multiple Choice",
                "isActive": true
            }
        },
        {
            "id": 1014,
            "name": "Which of the following is right syntex for arrow
function?",
            "questionTypeId": 1,
            "options": [
                {
                    "id": 1055,
                    "questionId": 1010,
                    "name": "a -> { return b; }",
                    "isAnswer": false
                },
                {
                    "id": 1057,
                    "questionId": 1010,
                    "name": "x <= x + y;",
                    "isAnswer": false
                },
                {
                    "id": 1056,
                    "questionId": 1010,
                    "name": "x <- x + 5;",
                    "isAnswer": false
                },
                {
                    "id": 1058,
                    "questionId": 1010,
                    "name": "x => x + 5;",
                    "isAnswer": true
                }
            ],
            "questionType": {
                "id": 1,
                "name": "Multiple Choice",
                "isActive": true
            }
        },
        {
            "id": 1015,
            "name": "Which new ES6 syntax helps with formatting output text - 
mixing variables with string literals, for example.",
            "questionTypeId": 1,
            "options": [
                {
                    "id": 1055,
                    "questionId": 1010,
                    "name": "Generator Functions",
                    "isAnswer": false
                },
                {
                    "id": 1057,
                    "questionId": 1010,
                    "name": "Arrow Functions",
                    "isAnswer": false
                },
                {
                    "id": 1056,
                    "questionId": 1010,
                    "name": "Template Strings",
                    "isAnswer": true
                },
                {
                    "id": 1058,
                    "questionId": 1010,
                    "name": "Set Data Structure",
                    "isAnswer": false
                }
            ],
            "questionType": {
                "id": 1,
                "name": "Multiple Choice",
                "isActive": true
            }
        },
        {
            "id": 1016,
            "name": "Which ES6 feature helps in merging of a number of changed properties into an existing object?",
            "questionTypeId": 1,
            "options": [
                {
                    "id": 1055,
                    "questionId": 1010,
                    "name": "Class syntex",
                    "isAnswer": false
                },
                {
                    "id": 1056,
                    "questionId": 1010,
                    "name": "Object.assign()",
                    "isAnswer": true
                },
                {
                    "id": 1057,
                    "questionId": 1010,
                    "name": "map data structure",
                    "isAnswer": false
                },
                {
                    "id": 1058,
                    "questionId": 1010,
                    "name": "Array.includes(obj);",
                    "isAnswer": false
                }
            ],
            "questionType": {
                "id": 1,
                "name": "Multiple Choice",
                "isActive": true
            }
        },
        {
            "id": 1017,
            "name": "What is the difference between == and === ?",
            "questionTypeId": 1,
            "options": [
                {
                    "id": 1055,
                    "questionId": 1010,
                    "name": "=== throws syntex error",
                    "isAnswer": false
                },
                {
                    "id": 1056,
                    "questionId": 1010,
                    "name": "== checks values only, === checks types as well",
                    "isAnswer": true
                },
                {
                    "id": 1057,
                    "questionId": 1010,
                    "name": "=== is reference type check only",
                    "isAnswer": false
                },
                {
                    "id": 1058,
                    "questionId": 1010,
                    "name": "Both are same",
                    "isAnswer": false
                }
            ],
            "questionType": {
                "id": 1,
                "name": "Multiple Choice",
                "isActive": true
            }
        },
        {
            "id": 1018,
            "name": "Which of the following is NOT the method of an Array?",
            "questionTypeId": 1,
            "options": [
                {
                    "id": 1055,
                    "questionId": 1010,
                    "name": ".map()",
                    "isAnswer": false
                },
                {
                    "id": 1057,
                    "questionId": 1010,
                    "name": ".includes()",
                    "isAnswer": false
                },
                {
                    "id": 1056,
                    "questionId": 1010,
                    "name": ".subscribe()",
                    "isAnswer": true
                },
                {
                    "id": 1058,
                    "questionId": 1010,
                    "name": ".flatMap()",
                    "isAnswer": false
                }
            ],
            "questionType": {
                "id": 1,
                "name": "Multiple Choice",
                "isActive": true
            }
        },
        {
            "id": 1019,
            "name": "What will be the output of the following code: ['a', 'b', 'c'].fill(7, 1, 2);?",
            "questionTypeId": 1,
            "options": [
                {
                    "id": 1055,
                    "questionId": 1010,
                    "name": "['a', 7, 'c']",
                    "isAnswer": true
                },
                {
                    "id": 1056,
                    "questionId": 1010,
                    "name": "['a', 7, 7, 'b', 'c']",
                    "isAnswer": false
                },
                {
                    "id": 1057,
                    "questionId": 1010,
                    "name": "['a', 'b', 'c']",
                    "isAnswer": false
                },
                {
                    "id": 1058,
                    "questionId": 1010,
                    "name": "['7', 7, 'c']",
                    "isAnswer": false
                }
            ],
            "questionType": {
                "id": 1,
                "name": "Multiple Choice",
                "isActive": true
            }
        }
    ]
}
```