

订阅DeepL Pro以编辑此演示文稿。  
访问[www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document)，了解更多信息。

**第 4 号实验室工作**

**TypeScript、Angular**

**1 部分**。根据你选择的主题领域设计一个 **Angular** 应用程序。

您可以使用 angular cli 命令生成应用程序模板-prep 模板：

***ng*** new <application\_name>

应用程序的根组件是 app.component（应用程序创建时自动生成，并放置在 src\app 文件夹中）。在 app.component.html（根组件的 html 模板）中放置 bootstrap 组件 jumbotron，并说明应用程序的功能（还需要在 package.json 文件中添加 bootstrap 依赖关系）。

在 src\app 项目文件夹中，根据分配选项生成一个**模块**。这可以通过 angular cli 命令完成：

***ng*** generate module <module\_name> --module=app

进入 src\app/module\_name 文件夹，根据分配变量生成**组件**：

***ng*** 生成组件 <component\_name>

要与数据交互，请在 src\app/module name 文件夹中创建一个服务文件夹，并使用命令在其中生成一个**服务**：

***ng*** 生成服务 <模块名称

要在应用程序中实施**路由，请将** AppRoutingModule 模块添加到项目根目录（与 app.component 处于同一级别）。可以使用命令

***ng*** generate module app-routing --flat --module=app

在 app.component.html 中，放置 <router-outlet> </router-outlet> 指令（被选中处理请求的组件将代替 <router-outlet> 元素呈现）。

要处理 src\app/module\_name 文件夹中的子路由，请使用以下命令添加模块 module\_nameRoutingModule

ng generate module\_name-module-routing --flat --module= module\_name

**2 部分**。开发一个模板驱动的表单，用于添加新项目（根据变量）、更新和删除项目。在应用程序中添加表单链接。

**第三部分**。在 Firebase 云平台上部署应用程序（参见 Firebase 指南）。

将 Cloud Firestore 数据库添加到 Firebase NoSQL 项目（请参阅 Cloud Firestore 手册）。在 Cloud Firestore 数据库中根据任务的变体创建一个集合，例如列表-主题。 将字段与变量相对应的文档添加到集合中。

|  |  |  |
| --- | --- | --- |
|  | 作业截止日期： | 作业到期日： |
| 12 组 | 04.09.23 | 14.09.23 - 应用程序结构、导航；  21.09.23 - 编辑表格；  28.09.23 - Firebase 上的项目 |
| 13 组 | 04.09.23 | 11.09.23 - 应用程序结构、导航；  18.05.23 - 增列项目表；  25.09.23 - Firebase 上的项目 |
| 14 组 | 04.09.23 | 11.09.23 - 应用程序结构、导航；  18.05.23 - 增列项目表；  25.09.23 - Firebase 上的项目 |

论文的最高分是 40 分。

交货日期

截至指定日期 - 系数 1，截至 24.09 - 系数 0.6，之后 - 系数 0.4。

选项：

1. **可选**。创建一个名为 Subjects 的模块。在该模块中放置组件（元件）SubjectCenter、SubjectList、SubjectDetails。添加导航：打开项目时，SubjectCenter 组件将显示在 <router-outlet> 元素的位置，SubjectList 组件将作为 <router-outlet> 元素的子元素加载。当点击 SubjectList 列表中的一个项目时，相应的 SubjectDetails 组件将被加载。

在文件 mock-subject-list.ts 中以数组形式存储应用程序中的课程列表：   
Subjects: Subject [] = [ { { id: 1, name: 'Math', teacher: 'Drozd' }, ... ]

1. **付款。**创建一个名为 "账户 "的模块。在该模块中放置组件（元件）AccountCenter、AccountList、AccountDetails。添加导航：打开项目时，AccountCenter 组件将显示在 <router-outlet> 元素的位置，AccountList 组件将作为 <router-outlet> 元素的子元素加载。点击 AccountList 项目之一时，相应的 AccountDetails 组件将被加载。账户列表以数组形式存储在应用程序的文件 mock-account-list.ts 中：

账户： 账户 [] = [ { { id: 1, sum: 'Math', owner: 'Drozd' }, ... ]

1. **医院**。创建一个名为 "处方 "的模块。在该模块中放置 PrescriptionCenter、PrescriptionList 和 PrescriptionDetails 组件。添加导航：打开项目时，PrescriptionCenter 组件将显示在 <router-outlet> 元素的位置，而 PrescriptionList 组件将作为 <router-outlet> 元素的子元素加载。当点击处方列表中的一个项目时，相应的 PrescriptionDetails 组件将被加载。

将应用程序中的处方列表以数组形式存储在 mock-prescription-list.ts 文件中：   
Prescriptions: Prescription [] = [ { { id: 1, description: 'pill', doctor: 'Drozd', patient: 'Linev' }, ... ]

1. **入学考试**。创建一个名为 Exams 的模块。在该模块中放置 ExamCenter、ExamList 和 ExamDetails 组件。添加导航：打开项目时，ExamCenter 组件将代替 <router-outlet> 元素显示，ExamList 组件将作为子组件加载。点击 ExamList 项目之一时，相应的 ExamDetails 组件将被加载。

将应用程序中的考试成绩列表以数组形式存储在文件 mock-exam-list.ts 中：   
Exams: Exam [] = [ { id: 1, name: 'Math', mark: '10', enrollee: 'Rudnev' }, ... ]

1. **图书馆**创建一个名为 Books 的模块。在该模块中放置 BookCenter、BookList 和 BookDetails 组件。添加导航：打开项目时，BookCenter 组件将显示在 <router-outlet> 元素的位置，而 BookList 组件将作为 <router-outlet> 元素的子元素加载。当点击 BookList 列表中的一个元素时，相应的 BookDetails 组件将被加载。

将应用程序中的书籍列表以数组形式存储在 mock-book-list.ts 文件中：   
Books: Book [] = [ { { id: 1, name: 'Angular', author: 'Drozd' }, ... ]

1. **设计办公室**。创建名为项目的模块。在该模块中放置组件（元件）ProjectCenter、ProjectList 和 ProjectDetails。添加导航功能：打开项目时，ProjectCenter 组件将代替 <router-outlet> 元素显示，ProjectList 组件将作为 <router-outlet> 元素的子元素加载。点击 ProjectList 项目之一时，相应的 ProjectDetails 组件将被加载。

在 mock-project-list.ts 文件中以数组形式存储应用程序中的项目列表：   
项目： 项目 [] = [ { id: 1, description: 'progr', customer: 'Drozd', price: 123 }, ... ]

1. **电话交换机**。创建名为 "服务 "的模块。在该模块中放置组件（元件）ServiceCenter、ServiceList 和 ServiceDetails。添加导航：打开项目时，ServiceCenter 组件将显示在 <router-outlet> 元素的位置，ServiceList 组件将作为该元素的子元素加载。单击其中一个 ServiceList 项目时，会加载相应的 ServicetDetails 组件。

将应用程序中的服务列表以数组形式存储在 mock-service-list.ts 文件中：   
服务： 服务 [] = [ { { id: 1, description: 'Internet', price: 123 }, ... ]

1. **Autobase**.创建一个名为 "Trips "的模块。在该模块中放置 TripCenter、TripList 和 TripDetails 组件（元件）。添加导航：打开项目时，TripCenter 组件将显示在 <router-outlet> 元素的位置，TripList 组件将作为该元素的子元素加载。单击其中一个 TripList 项目时，将加载相应的 TripDetails 组件。

将应用程序中的航班列表以数组形式存储在 mock-trip-list.ts 文件中：   
Trips: Trip [] = [ { id: 1, description: 'freight', driver: 'Drozd' }, ... ]

1. **网上商店**。创建一个名为 "商品 "的模块。在该模块中放置 GoodCenter、GoodList 和 GoodDetails 组件。添加导航：打开项目时，GoodCenter 组件将显示在 <router-outlet> 元素的位置，GoodList 组件将作为 <router-outlet> 元素的子元素加载。点击其中一个 GoodList 项目时，相应的 GoodDetails 组件就会加载。

将应用程序中的商品列表以数组形式存储在 mock-good-list.ts 文件中：   
Goods: Good [] = [ { { id: 1, description: 'pets', price: '123' }, ... ]

1. **餐厅**系统。创建一个名为菜单的模块。在该模块中放置 MenuCenter、MenuList 和 MenuDetails 组件。添加导航：打开项目时，MenuCenter 组件将显示在 <router-outlet> 元素的位置，MenuList 组件将作为子元素加载。单击其中一个 MenuList 项目时，会加载相应的 MenuDetails 组件。

将应用程序中的菜单以数组形式存储在文件 mock-menu-list.ts 中：   
菜单：菜单 [] = [ { { id: 1, description: 'pie', price: 'Drozd' }, ... ]

1. **航空公司。**创建一个名为 Flights 的模块。在该模块中放置 FlightCenter、FlightList 和 FlightDetails 组件。添加导航：打开项目时，FlightCenter 组件将显示在 <router-outlet> 元素的位置，FlightList 组件将作为该元素的子元素加载。点击 FlightList 项目之一时，将加载相应的 FlightDetails 组件。

将应用程序中的航班列表以数组形式存储在 mock-flight-list.ts 文件中：   
Flights: Flight [] = [ { id: 1, from: 'Minsk', to: 'London', crew: 'asd' }, ... ]

1. **期刊**。创建名为期刊的模块。在该模块中放置组件 PeriodicalCenter、PeriodicalList 和 PeriodicalDetails。添加导航：打开项目时，PeriodicalCenter 组件将代替 <router-outlet> 元素显示，PeriodicalList 组件将作为 <router-outlet> 元素的子元素加载。点击其中一个 PeriodicalList 项目时，相应的 PeriodicalDetails 组件将被加载。

将应用程序中的期刊列表以数组形式存储在文件 mock-periodical-list.ts 中：   
Periodicals: Periodical [] = [ { id: 1, name: 'News', publisher: 'Drozd', price: 123 }, ... ]

1. **酒店订购**。创建一个名为 "公寓 "的模块。在该模块中放置 ApartmentCenter、ApartmentList 和 ApartmentDetails 组件。添加导航：打开项目时，ApartmentCenter 组件将显示在 <router-outlet> 元素的位置，而 ApartmentList 组件将作为该元素的子元素加载。当点击 ApartmentList 列表中的一个元素时，就会加载相应的 ApartmentDetails 组件。

将应用程序中的房间列表以数组形式存储在 mock-apartment-list.ts 文件中：   
Apartments: Apartment [] = [ { { id: 1, description: 'vip', price: 123 }, ... ]

1. **住房和公用设施**。创建一个名为 "住房 "的模块。在该模块中放置 HousingCenter、HousingList 和 HousingDetails 组件。添加导航：打开项目时，HousingCenter 组件将显示在 <router-outlet> 元素的位置，HousingList 组件将作为该元素的子元素加载。当点击 HousingList 项目之一时，相应的 HousingDetails 组件将被加载。

将应用程序中的工作列表以数组形式存储在文件 mock-housing-list.ts 中：   
Housings: Housing [] = [ { { id: 1, description: 'electricity', worker: 'Drozd' }, ... ]

1. **汽车租赁**。创建一个名为 "汽车 "的模块。在该模块中放置 CarCenter、CarList 和 CarDetails 组件（元件）。添加导航：打开项目时，CarCenter 组件将显示在 <router-outlet> 元素的位置，CarList 组件将作为子元素加载。点击 CarList 项目之一时，相应的 CarDetails 组件将被加载。

将应用程序中的汽车列表以数组形式存储在 mock-car-list.ts 文件中：   
Cars: Car [] = [ { id: 1, model: 'BMW', state: 'good', price: 123 }, ... ]

1. **种族**创建一个名为 Races 的模块。在该模块中放置 RaceCenter、RaceList 和 RaceDetails 组件。添加导航：打开项目时，RaceCenter 组件将显示在 <router-outlet> 元素的位置，而 RaceList 组件将作为 <router-outlet> 元素的子元素加载。单击其中一个 RaceList 项目时，将加载相应的 RaceDetails 组件。

将应用程序中的比赛列表以数组形式存储在 mock-race-list.ts 文件中：   
Races: Race [] = [ { { id: 1, date: 1.03, winner: 'Luch' }, ... ]

可对**比赛中的**不同**马匹下注**。一匹**马**不能

1. **测试**创建一个名为测试的模块。在该模块中放置 TestCenter、TestList 和 TestDetails 组件（元件）。添加导航：打开项目时，TestCenter 组件将显示在 <router-outlet> 元素的位置，TestList 组件将作为该元素的子元素加载。单击 TestList 项目时，相应的 TestDetails 组件将被加载。

将应用程序中的测试列表以数组形式存储在文件 mock-test-list.ts 中：   
Tests: Test [] = [ { id: 1, subject: 'Math', complexity: 'medium' }, ... ]

1. **咖啡机**创建一个名为 "饮料 "的模块。在该模块中放置 DrinkCenter、DrinkList 和 DrinkDetails 组件。添加导航：打开项目时，DrinkCenter 组件将显示在 <router-outlet> 元素的位置，DrinkList 组件将作为 <router-outlet> 元素的子元素加载。点击其中一个 DrinkList 项目时，相应的 DrinkDetails 组件将被加载。

将应用程序中的饮料列表以数组形式存储在 mock-drink-list.ts 文件中：   
Drinks: Drink [] = [ { { id: 1, name: 'Latte', price: 123 }, ... ]

1. **公园**创建一个名为 "植物 "的模块。在该模块中放置 PlantCenter、PlantList 和 PlantDetails 组件。添加导航：打开项目时，PlantCenter 组件将显示在 <router-outlet> 元素的位置，而 PlantList 组件将作为 <router-outlet> 元素的子元素加载。单击 PlantList 项目时，相应的 PlantDetails 组件将被加载。

将应用程序中的植物列表以数组形式存储在 mock-plant-list.ts 文件中：   
Plants: Plant [] = [ { { id: 1, name: 'tree', task: 'watering' }, ... ]

1. **旅行社**。创建一个名为 "旅游 "的模块。在该模块中放置组件 TourCenter、TourList 和 TourDetails。添加导航：打开项目时，TourCenter 组件将显示在 <router-outlet> 元素的位置，TourList 组件将作为该元素的子组件加载。点击其中一个 TourList 项目时，会加载相应的 TourDetails 组件。

在文件 mock-tour-list.ts 中以数组形式存储应用程序中的旅游团列表：   
Tours: Tour [] = [ { { id: 1, destination: 'Paris', type: 'excursion' }, ... ]

21.系统开发人员**团队**。创建一个名为 "开发人员 "的模块。在该模块中放置 DeveloperCenter、DeveloperList 和 DeveloperDetails 组件。添加导航：打开项目时，DeveloperCenter 组件将代替 <router-outlet> 元素显示，DeveloperList 组件将作为 <router-outlet> 元素的子元素加载。单击 DeveloperList 项目时，会加载相应的 DeveloperDetails 组件。

将应用程序中的开发人员列表以数组形式存储在文件 mock-developer-list.ts 中： Developers: Developer [] = [ { id: 1, name: 'Drozd', qualification: 'manager', salary: 123 }, ... ]。

22.**视频库**创建一个名为 "电影 "的模块。在该模块中放置 MovieCenter、MovieList 和 MovieDetails 组件。添加导航：打开项目时，MovieCenter 组件将代替 <router-outlet> 元素显示，而 MovieList 组件将作为 <router-outlet> 元素的子元素加载。点击 MovieList 项目之一时，相应的 MovieDetails 组件将被加载。

在文件 mock-movie-list.ts 中以数组形式存储应用程序中的电影列表：   
Movies: Movie [] = [ { { id: 1, name: 'Math', direcor: 'Drozd'. date: 1.02 }, ... ]

23.**时间表**。创建一个名为 Subjects 的模块。在该模块中放置组件（元件）SubjectCenter、SubjectList、SubjectDetails。添加导航：打开项目时，SubjectCenter 组件将显示在 <router-outlet> 元素的位置，而 SubjectList 组件将作为 <router-outlet> 元素的子元素加载。当点击 SubjectList 列表中的一个项目时，相应的 SubjectDetails 组件将被加载。

在文件 mock-subject-list.ts 中以数组形式存储应用程序中的科目列表：   
Subjects: Subject [] = [ { { id: 1, name: 'Math', teacher: 'Drozd', dayWeek: 'monday', class: 123 }, ... ]

24.**天气**创建名为 "区域 "的模块。在该模块中放置组件（元件）RegionCenter、RegionList 和 RegionDetails。添加导航：打开项目时，RegionCenter 组件将显示在 <router-outlet> 元素的位置，RegionList 组件将作为 <router-outlet> 元素的子元素加载。点击 RegionList 列表中的一个元素时，相应的 RegionDetails 组件将被加载。

在文件 mock-region-list.ts 中以数组形式存储应用程序中的区域列表：   
区域： 区域 [] = [ { id: 1, name: 'Gomel', language: 'russian', square: 123 }, ... ]

25.**低成本航空公司**系统。创建一个名为 Flights 的模块。在该模块中放置 FlightCenter、FlightList 和 FlightDetails 组件。添加导航：打开项目时，FlightCenter 组件将显示在 <router-outlet> 元素的位置，而 FlightList 组件将作为 <router-outlet> 元素的子元素加载。点击 FlightList 项目之一时，将加载相应的 FlightDetails 组件。

将应用程序中的航班列表以数组形式存储在 mock-flight-list.ts 文件中：   
Flights: Flight [] = [ { id: 1, from: 'Minsk', to: 'London', date: 1.03, luggage: 20, price: 80 }, ... )

26.动物园创建一个名为 Zoo 的模块。在该模块中放置组件（元件）AnimalCenter、AnimalList 和 AnimalDetails。添加导航：打开项目时，AnimalCenter 组件将显示在 <router-outlet> 元素的位置，AnimalList 组件将作为该元素的子组件加载。点击其中一个 AnimalList 项目时，会加载相应的 AnimalDetails 组件。

将应用程序中的动物列表以数组形式存储在 mock-animal-list.ts 文件中：   
动物： Animal [] = [ { { id: 1, animalkind:'Panda', nikname:'Boo', age: '3', weight: '12' }, ... ]