The Methods and Implementation of UUI Frontend Architecture

Shen Tao
Xu Ran

15th Jan, 2020
Overview

• The frontend architecture methods applied in UUI
• The frontend architecture implementation of UUI
• The presentation of the frontend structure of UUI and the conclusion and suggestion of frontend architecture best practice
The Frontend Architecture Methods Applied in UUI
Reasonable Project Structure

• A reasonable project structure can
  - Avoid the messy structure when there is lots of business files and make it easier to find the particular parts of codes and improve the development speed
  - Segregate the business and functional codes such as data model and http request methods to make the program ‘safer’

• A reasonable project structure should:
  - Take care of the project technology stack
  - Take care of the features of specific business
Component-based Architecture

- Common sense of modern framework
- Framework like Angular is already component-based
  - For a mature program, we need a more careful architecture. We have to extract the public components and make them easier to reuse.
Frontend and Backend Separation

• This is the essential part of modern frontend development which includes advantages such as faster development and modularity.
• Thanks to Angular, the separate frontend and backend is already in embryo.
• For a modern frontend project, we have to support more capabilities to adapt the requirement of swift and separate development. Such as: running individually, mocking interface and deploying independently.
The Frontend Architecture Implementation of UUI
We refer to many articles and high-star Angular projects in Github. According to those best practices and the actual condition of our business, we build our project as the image:

The core parts of our architecture are the core, shared and views folders
- The core folder is the container of the core functional codes for the whole project such data models and http request methods.
- The shared folder is the container for all the public components.
- The views folder is the container of the business components which is the main part of development.
Component-based Architecture - overview

• To apply this method, we extract the components from the original codes and make the business codes more 'pure'.
  - We extract the *dumb* components which are just responsible for presenting something to the DOM
  - We don't extract those components that are responsible for keeping track of state and caring about how the program works which are called *smart* components
  - Things will be different if other framework such as *react* is used, and we just take advantage of Angular.
Component-based Architecture - Pie Component

```html
<app-pie [initData]="alarmChartInit" [chartData]="alarmChartData"></app-pie>

this.alarmChartData = {
  series: [{
  ]
};
```

```javascript
ngOnInit() {
  this.initOpts = {
    renderer: 'canvas',
    height: this.initData.height,
    width: this.initData.width,
  };
  this.barOption = {
    tooltip: this.initData.option.tooltip,
    grid: this.initData.option.grid,
    xAxis: this.initData.option.xAxis,
    yAxis: this.initData.option.yAxis,
    series: this.initData.option.series
  }
}
```
Frontend and Backend Separation – mock data

• The goal of mock data scheme:
  - Build the ‘one command start’ system to start the project in mock environment
  - Separate the business codes and the mock data codes to make the mock data module an independent one
  - Swift development

• Difficulties:
  - Some original API paths consist of variable which makes it impossible to be mocked by the local data files
  - The RESTful standard allows one single API path to hold several different request methods: POST, GET, DELETE, PUT

• Solutions:
  - Use the ‘rewrite’ middleware to rewrite specific API paths
  - Create the ‘routes.js’ file to list the API paths which need to be rewrote
  - Use the interface interception system to transfer all kinds of request to GET method
Frontend and Backend Separation – mock data

Goals

- One Command Startup
- Separation of Mock and Business Codes
- Agile Development

Configuration

- Configure package.json
  - Setup json-server
  - Set mock command
  - Set proxy configuration
- Configure routes.js
- Create server.js
  - Import local json data
  - Rewrite routes
  - Transfer interface
- Create faker.js
  - Interception of all kinds of requests
    - Intercept the POST, PUT, DELETE methods
    - Rewrite the request path
    - Change request method

Solutions

- Customize Request Routes
- Interface Interception
Frontend and Backend Separation – mock data

routes.js

package.json

server.js – data interception

faker.js

"/uui-lcm/customers/:customer/service-subscriptions": "/serviceType",
"/uui-lcm/serviceNumByServiceType/:customer": "/CustomersColumn",

"mockproxy": "ng serve --proxy-config localproxy.conf.json",
"mockconfig": "node ./src/app/mock/server.js --port 3002",
"mock": "npm run mockconfig | npm run mockproxy",

server.post(`/${baseUrl}/*`, (req, res, next) => {
    const prefix = req.url.replace(baseUrl, "")
    req.url = `/${prefix}`;
    req.method = 'GET';
    next();
});

server.put(`/${baseUrl}/*`, (req, res, next) => {
    const prefix = req.url.replace(baseUrl, "")
    req.url = `/${prefix}`;
    req.method = 'GET';
    next();
});

ErrorResponse

home: _.times(10, function (n) {
    return {
        id: n,
        name: faker.name.findName(),
        phone: faker.phone.phoneNumber(),
        address: faker.address.streetAddress(),
        avatar: faker.internet.avatar()
    }));

language: {
    language: faker.random.locale()
}
We have recorded the work about how we build up the mock scheme in detail. Click this [wiki](#) if you are interested.
The Presentation of the Frontend Structure of UUI and the Conclusion and Suggestion of Frontend Architecture
The Presentation of The Frontend Architecture of UUI - Project Structure

- src
  - app
    - core
    - models
    - services
  - mock
    - fake # container of all mock data generated by faker.js
    - json # container of all local mock data files
      - routes.js
        - config file of proxy routes
      - server.js
        - mock data server
    - shared
      - components
      - utils
    - views
      - alarm
      - ......
The Presentation of The Frontend Architecture of UUI – use case in Frankfurt Release

• To support the use cases of 5G Slicing business, we develop CSMF and NSMF portal to provide customers and operators with functions such as creating service, activing/de-activing/terminating slicing and monitoring service.
  - CSMF is a portal that provides service for customers. So we have developed a new system to support functions such as creating orders, managing slicing order, managing slicing business and monitoring slicing business
  - NSMF is a portal that provides service for operators. Since the original UUI project aims to provide service to the network operators, we merge NSMF portal into the original system.
Create Slicing Business Order

- Slicing Business Name: 

- Max Number of UEs: 

- Data Rate Downlink (Mbps): 

- Latency: 

- Data Rate Uplink (Mbps): 

- Resource Sharing Level: 
  - Shared 
  - Non-shared 

- Mobility: 
  - Stationary 

- Use Interval (Month): 

- Area: 
  - Province 
  - City 
  - District 
  - Street
The Conclusion and Suggestion of Frontend Architecture

• reasonable project structure
  - Build reasonable project structure to make the project safer, clear and easy to develop

• component-based architecture
  - Angular is component-based and most all the components can be created by Angular-cli
  - Dumb component can be used in Angular framework to avoid useless work and make the structure more clear

• frontend and backend separation
  - Mock data scheme which we have built has made the frontend project totally divided from the backend and once both sides have decided the data structure, the frontend can develop without the data support of backend which is the true meaning of frontend and backend separation
Thanks