

COS30043

Interface Design and Development

Lecture 7 – Application Programming Interface 1

Topics



- API and REST API
- getJSON
- Requesting Server Data
- Inserting Server Data
- Updating Server Data
- Deleting Server Data



API and REST API

- API - application programming interface
- It is an **interface** that defines interactions between multiple applications
- It allows applications to **access external data and interact with external software components**
- REST - representational state transfer
- A RESTful API (REST API) is an architectural style for API that uses **HTTP requests** to access and use data.
- Data can be used by **GET, PUT, POST and DELETE** method, which refers to the reading, updating, creating and deleting
- Data formats include **json, xml** and etc.



Contents

- Method we are going to use in this class:
 - `$.getJSON()`: a jQuery method to get JSON data
 - `fetch()`: a javascript method of the fetch API. The fetch API is JavaScript's built-in way to make API requests.
 - a. Requesting server data - GET
 - b. Inserting server data - POST
 - c. Updating server data - PUT
 - d. Deleting server data - DELETE



Example APIs

Weather forecasts

<http://www.7timer.info/bin/api.pl?lon=113.17&lat=23.09&product=astro&output=json>

Cocktail recipes

<https://www.thecocktaildb.com/api/json/v1/1/search.php?s=margarita>

GitHub Jobs

<https://jobs.github.com/positions.json?description=api>

Json Placeholder - Free fake API for testing and prototyping

<https://jsonplaceholder.typicode.com/>



Topics

- API and REST API
- getJSON
- Requesting Server Data
- Inserting Server Data
- Updating Server Data
- Deleting Server Data



View – HTML

```
<div id= "app" >
```

```
... code to prepare  
input data
```

```
call the method in the component  
output the results
```

```
</div>
```



Model – Data in JSON Format

- For example

```
[ { "msg" : "data 1" } ,  
  { "msg" : "data 2" } ,  
  { "msg" : "data 3" } ]
```

- Data can be stored in a text file for get method, or in the database table
- Data updated through a representational state transfer application programming interfaces (RESTful API)



getJSON example

HTML:

```
<div id="app">
  <app-readjson></app-readjson>
</div>
```

JavaScript:


```
app.component('app-readjson', {
  data: function() {
    return {msg: [ ]}
  },
  template: `
    <ul>
      <li v-for="m in msg"> {{ m }} </li>
    </ul>
  `,
  mounted() {
    var self = this;
    $.getJSON('https://jsonplaceholder.typicode.com/posts',
function(data) {
    self.msg = data;
  })
    .fail(function() { alert('getJSON request failed! ');
    })
  }
})
```

Need jQuery

```
<script src="js/jquery.min.js"></script>
```



Topics

- API and REST API
- getJSON
-  • Requesting Server Data
- Inserting Server Data
- Updating Server Data
- Deleting Server Data



View – Requesting JSON Data

```
// HTML
...
<app-readjson></app-readjson>
...
// Vue JS
...
Vue.component('app-readjson', {
  template: `
    ...

    <ul>
      <li v-for="m in msg">{{m}}</li>
    </ul>
    <p>Error: {{err}}</p>

    ...
```

N.B. "... " indicates lines of code which are not shown here



ViewModel – Requesting JSON Data(Continued)

... .

```
mounted() { //Called after the instance has been mounted
    var self = this;
    var url = 'https://jsonplaceholder.typicode.com/posts';

    fetch(url)
    .then( response =>{
        //turning the response into the usable data
        return response.json( );
    })
    .then( data =>{
        //This is the data you wanted to get from url
        self.msg=data;
    })
    .catch(error => {
        self.err=error
    });
}
```

}

... .



Model

- model – populated from a text file
 - **persons.json** stored in data directory
- ```
[{ "name": "Alice", "age": 20 },
 { "name": "Billy", "age": 22 },
 { "name": "Chris", "age": 25 }]
```
- Data can also be retrieved from a database table using a RESTful API



# Topics

---

- API and REST API
- getJSON
- Requesting Server Data
-  • Inserting Server Data
- Updating Server Data
- Deleting Server Data



# Insert server data

---

...

```
<v-form>
 <v-text-field label="Title" v-model="title" >
 </v-text-field>

 <v-text-field label="Message" v-model="body" >
 </v-text-field>

 <v-btn v-on:click="postData(title,body)"
 color="success">
 Add
 </v-btn>
</v-form>...
```

**N.B. "..."** indicates the lines of code which are not shown here



# Insert server data (Continued)

---

```
methods: {
 postData: function(title, body){
 var self = this;
 fetch('https://jsonplaceholder.typicode.com/posts', {
 method: 'POST',
 headers: {
 'Content-Type': 'application/json'
 },
 body: JSON.stringify({
 userId:1, id:1,
 title: title, body: body
 })
 })
 .then(response => {
 return response.json()
 })
 .then(data =>
 // this is the data we get after response.json()
 this.msg=data
)
 .catch(error => {self.err=error})
 }
}
```



# Insert server data (await syntax)

---

```
methods: {
 postData: async function(title, body){
 var self = this;
 let response = await
 fetch('https://jsonplaceholder.typicode.com/posts', {
 method: 'POST',
 headers: {
 'Content-Type': 'application/json'
 },
 body: JSON.stringify({
 userId:1,
 id:1,
 title: title,
 body: body
 })
 });

 const data = await response.json();
 self.msg=data;
 }
}
```

# View – Status Output

---

```
// Output
```

```
...
```

```
<v-card-text>
```

```
 <p>Output Message : {{ msg }}</p>
```

```
 <p>Error: {{err}}</p>
```

```
</v-card-text>
```

```
...
```

N.B. "... " indicates the lines of code which are not shown here



# Topics

---

- API and REST API
- getJSON
- Requesting Server Data
- Inserting Server Data
- Updating Server Data
- Deleting Server Data



# Updating server data

---

...

```
<v-form name="myForm2" class="form-
horizontal">
 <v-text-field label="Title"
 v-model="title" /> </v-text-field>
 <v-text-field label="Message"
 v-model="body" /> </v-text-field>
 <v-btn v-on:click="updateData(title, body) "
 color="primary"> Update
</v-btn>
</v-form>
```

...

N.B. "... " indicates the lines of code which are not shown here



# Updating server data (Continued)

---

```
methods: {
 updateData: function(title, body) {
 //your data to send
 const myObject = {
 "userId": 1,
 "id": 1,
 "title": title,
 "body": body
 };
 fetch('https://jsonplaceholder.typicode.com/posts/1', {
 method: 'PUT',
 headers: {
 'Content-Type': 'application/json'
 },
 body: JSON.stringify(myObject)
 })
 .then(response => {return response.json() })
 .then(data => { // this is the data we get after response.json
 console.log(data)
 })
 .catch(error => console.log('The error is: ', error))
 }
}
```

# View – Status Output

---

```
// Output
```

```
...
```

```
<v-card-text>
```

```
 <p>Output Message : { {msg} }</p>
```

```
 <p>Errors: { {err} }</p>
```

```
</v-card-text>
```

```
...
```

N.B. "... " indicates the lines of code which are not shown here



# Model

---

- The example will only work if the PUT method api exists
- Need to check the parameters required by the API



# Topics

---

- API and REST API
- getJSON
- Requesting Server Data
- Inserting Server Data
- Updating Server Data
- Deleting Server Data





# Deleting server data

---

```
...
<v-form>
 <v-text-field label="Name" v-model="id" /> </v-
text-field>

 <v-btn depressed v-on:click="deleteData(id) "
 color="primary"> Delete
</v-btn>

</v-form>
```

...  
**N.B. "..."** indicates lines of code which are not shown here



# Deleting server data (Continued)

---

```
methods: {
 deleteData: function(id) {
 fetch('https://jsonplaceholder.typicode.com/posts/'+id, {
 method: 'DELETE',
 headers: { 'Content-Type': 'application/json' },
 body: null
 })
 .then(
 response => { return response.json() }
)
 .then(// this is the data we get after response.json()
 data => console.log(data)
)
 .catch(error => console.log('error is', error))
 }
}
```

# View – Status Output

---

```
// Output
```

```
...
```

```
<v-card-text>
```

```
 <p>Output Message : {{msg}}</p>
```

```
 <p>Error: {{err}}</p>
```

```
</v-card-text>
```

```
...
```

N.B. "...” indicates the lines of code which are not shown here



# Model

---

- The above example will only work if the DELETE method API exists
- Need to check the parameters required by the API, for this example the table name, field key e.g. name, and key value are required
- It will delete the record(s) where it matches the name



# WHAT'S NEXT?

## - API 2