ЛАБОРАТОРНО УПРАЖНЕНИЕ №5

Достъп до нерелационни бази данни

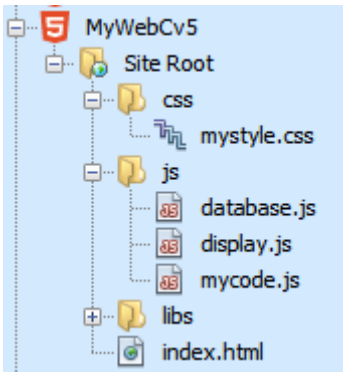
Цел: Програмна реализация на асинхронни заявки към облачно базирани нерелационни (NoSQL) бази данни.

Задача 1: Създайте Web приложение, което ще бъде ваше лично CV. Разширете възможностите на приложението от Лабораторно упражнение №4 със следната функционалност:

- Информацията за студента от секция с идентификатор „info-box" да се получава от база данни IBM Cloudant. За целта създайте база данни с име webservice и документ с име, което съвпада с вашия факултетен номер. Документът е в JSON формат и трябва да съдържа информация за вас (текст и снимка). За повече информация виж Семинарно упражнение №3.
- Програмният код за комуникация с IBM Cloudant да бъде във файл с име database.js, папка js. Да се използва библиотека jQuery с цел създаване на функция за *асинхронна* комуникация с базата данни чрез AJAX заявки.
- Да се използват възможностите на jQuery за прехващане на събития при стартиране и завършване на AJAX заявки. При стартиране на заявка да се показва информация за това (текст). При завършване на заявка – съобщението да се скрива. За целта създайте нова секция с идентификатор „loading-box" в тялото на етикет main.
- Създайте нова секция в тялото на етикет main с идентификатор „error-box" в която ще се показва съобщение при възникване на грешка при комуникация с базата данни. Напишете метод showError() - файл display.js.

Достъпът до IBM Cloudant се реализира чрез протокол HTTPS. Заявките изискват авторизация на достъпа чрез ключ и неговата стойност. Тази двойка се получават при администриране на базата данни (виж Семинарно упражнение №3). Услугата очаква да използвате *базова авторизация* на достъпа. За да може да зададете timeout за заявките към базата данни се налага да използвате *асинхронни* AJAX заявки. За целта трябва да използвате възможностите на библиотека jQuery и да използвате *JavaScript промиси*, за да синхронизирате вашият код с асинхронните заявки.

Структурата на обновения проект ще бъде следната:



Как се генерират AJAX заявки чрез библиотека jQuery”?

Библиотека jQuery позволява генериране на *синхронни* и *асинхронни* AJAX заявки към Web или облачно-базирани услуги. Функцията, която ще използваме, е с име ajax(). Тя изисква като аргумент един JSON обект чрез който се описва какво точно представлява заявката. При синхронните заявки функцията връща след като се получи отговор от сървъра. При асинхронните заявки, функцията връща веднага, а комуникацията се извършва на фонов режим спрямо вашия код. Следва пример за използване на ajax():

```
$.ajax({
    type: 'GET',
    url: url,
    async: true,
    cache: false,
    contentType: 'application/json',
    timeout: TIMEOUT,
    beforeSend: function (req) {
        req.setRequestHeader('Accept', 'application/json');
        req.setRequestHeader('Authorization', 'Basic ' + hash);
    },
    success: function (data) {
        // код за успешно изпълнение
    },
    error: function (data) {
        // код за неуспешно изпълнение
    }
});
```

При конкретния пример JSON обекта, който се предава като аргумент, има следните полета:

- type – тип на заявката (GET, POST, …).
- url – URI към ресурса до който се праща заявката.
- async – асинхронна или синхронна заявка (true, false).
- cache – проверка или не на кеша при формиране на заявката. Ако искате всеки път заявката реално да достигне до ресурса задайте false.
- contentType – в какъв MIME формат се очаква да бъде отговора – в случая очакваме JSON обект.
- timeout – може да се задава само при *асинхронни* заявки – колко време в ms да се чака отговор от сървъра.
- succsess – функция, която се вика при успешно изпълнение на заявката.
- error – функция, която се вика при неуспешно изпълнение на заявката.
- beforeSend – функция, която се вика преди заявката да се изпрати. Тук целта е да конфигурираме някои полета от *заглавния блок* на заявката, а не да останат техните стойности по подразбиране. В конкретният случай се задават стойностите на две полета: **Accept** и **Authorization**. Чрез първото указваме очаквания формат на отговора, а чрез второто задаваме *базова авторизация* на клиента. След „Basic “ трябва да следва името и паролата на клиента, но кодирани чрез алгоритъм Base64. Това се изисква при базова авторизация на достъпа. За целта може да използвате JavaScript функция btoa(), например:

```
var hash = btoa(username + ':' + password);
```

Как да синхронизираме JavaScript код?

Програмният език JavaScript използва *една нишка* при изпълнение на програмен код. Това означава, че системният и потребителският JavaScript код споделят една програмна нишка. За да не се блокират системните функции на браузъра (например обработка на събития от потребителския интерфейс), Web APIs използват множество нишки, когато е необходимо.

Програмният език JavaScript предлага начин за създаване на асинхронно работещи модули, както и възможност за *синхронизиране* на съвместната работа на множество такива модули. За целта се използва интерфейс $Promise$. Чрез обект, създаден чрез този интерфейс, е възможно да дефинираме асинхронно работещ код, както и да получим информация в момента, когато кодът завърши без или с грешка:

```
1  var promise = new Promise(function(resolve, reject) {
2      // асинхронен код, изпълнението на който
3      // завършва с уставяване на флаг статус
4      if (статус === "ок") {
5          resolve(обект);
6      }
7      else {
8          reject(обект);
9      }
10 });
```

Конструкторът (ред 1) изисква един аргумент – анонимна функция с два аргумента - $resolve$ и $reject$. Това са функции, които се викат при изпълнение на асинхронния код без грешка ($resolve$) или с грешка ($reject$). Тези функции изпълняват ролята на $callback$ функции. Обектите, които те предават, са достъпни чрез функция $then$. Тя изисква два атрибута. Първият атрибут е функция, която се вика при изпълнение на кода без грешка. Вторият атрибут е функция, която се вика при изпълнение на кода с грешка:

```
promise.then(function(обект) {
  // успешно изпълнен код (resolve)
}, function(обект) {
  // неуспешно изпълнен код (reject)
});
```

Обработката при грешки може да се реализира и чрез функция $catch$. Логиката е същата, както при предходния пример, но този синтаксис е по-разбираем:

```
promise.then(function(обект) {
  // успешно изпълнен код (resolve)
}).catch(function(обект) {
  // неуспешно изпълнен код
})
```

Когато трябва да се синхронизира работата на няколко блока от асинхронен код, за всеки от тях се създава обект $promise$. След това се използва функция all. Към така създадения чрез функция all promise обект може да се приложат функции $then$ и/или $catch$:

```
Promise.all(array_of_promises).then(function() {
  // всички модули са завършили успешно
}, function() {
  // един или повече модули не са
  // завършили успешно
});
```

За да не блокира кода на функция ajax() ще използваме асинхронни заявки (async=true). Ще напишем функция getInfo() която връща обект промис когато асинхронната заявка към базата данни завърши:

```
function getInfo(hash) {
    var url = PROTOCOL + SERVER_NAME + ':' + PORT + '/' + DATABASE + '/' +
            DOCUMENT + '?attachments=true';
    var promise = new Promise(function (resolve, reject) {
        $.ajax({
            type: 'GET',
            url: url,
            async: true,
            cache: false,
            contentType: 'application/json',
            timeout: TIMEOUT,
            beforeSend: function (req) {
                req.setRequestHeader('Accept', 'application/json');
                req.setRequestHeader('Authorization', 'Basic ' + hash);
            },
            success: function (data) {
                resolve(data);
            },
            error: function (data) {
                reject(data);
            }
        });
    });
    return promise;
}
```

Идентификаторът на ресурса (документа от базата данни) се формира от множество константи:

```
var PORT = 443;
var TIMEOUT = 4500;
var PROTOCOL = 'https://';
var SERVER_NAME =
'024b178f-c9b8-4bc0-ae0e-67b8cbc7d5ec-bluemix.cloudantnosqldb.appdomain.cloud';
var DATABASE = 'webservice';
var DOCUMENT = '123456';
```

Трябва да промените стойностите на константи SERVER_NAME, DATABASE и DOCUMENT.

Как да обработим обект обещание?

Нека да използваме функция sendRequest(), за да изпратим заявка към базата данни:

```
function sendRequest() {
    var username = $('input[name=username]').val();
    var password = $('input[name=password]').val();
    var hash = btoa(username + ':' + password);
    getInfo(hash)
        .then(
            function (result) {
                // програмен код, който получава документа като
                // JSON обект с име result
        }).catch(
            function (error) {
                // програмен код, който обработва грешката,
                // обект с име error
            }
        );
    return false;
}
```

Първо, получаваме името и паролата на клиента(username и password) чрез функция val() от библиотека jQuery. След това кодираме името и паролата чрез функция btoa(), за да можем да използваме бозава авторизация на достъпа. Следва викане на функция getInfo(), която очаква получаване на промис обект. Ако се е изпълнил метод resolve() управлението се предава в тялото на then(). Обектът result е съдържанието на документа като JSON обект. Можете да извлечете от него желаната информация по следния начин:

```
var name = result.name;
var specialty = result.specialty;
var specialtyfullname = result.specialtyfullname;
var course = result.course;
var about = result.aboutme;
var imageObject = result._attachments['picture.png'];
```

Информацията за студента е два вида – текст и картинка. Текстът трябва да се запише в тялото на секция с идентификатор „info-text”, а снимката – в секция с идентификатор „info-picture”. Тъй като това съдържание се налага да се обнови програмно се налагат промени във index.html и функция showInfo():

```
<section id="loading-box">
    Loading ...
</section>
<section id="info-box">
    <div id="info-picture">
        <img src="" id="picture" alt="Снимка"/>
    </div>
    <div id="info-text">
    </div>
</section>
<section id="error-box">
    Error
</section>
```

```
function showInfo(picture, info) {
    $('#picture').attr('src', picture);
    var infoBox = $('#info-text');
    infoBox.html(info);
    view('info-box');
}
```

Задаването на стойност на атрибут „src” от етикет img се реализира чрез jQuery функция attr(). В случая стойността е обекта picture. Ето как може да получите програмно този обект:

```
var image = imageObject.data;
var picture = '' + image;
```

Трябва да укажете, че снимката е кодиран чрез алгоритъм Base64 PNG файл (това кодиране се реализира от IBM Cloudant, когато качите снимка като attachment).

Когато се получи грешка се вика кода от тялото на catch(). Обектът, който описва грешката, е error. Чрез поле status може да получим информация за грешката с цел да я обработим, например:

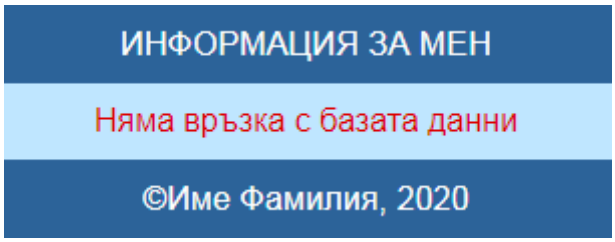
```
var status = error.status;
if (status === 0) {
    showError('Няма връзка с базата данни', 3);
}
else if (status === 401) {
    $('input[name=username]').val('');
    $('input[name=password]').val('');
    showError('Грешка при авторизация', 3);
}
```

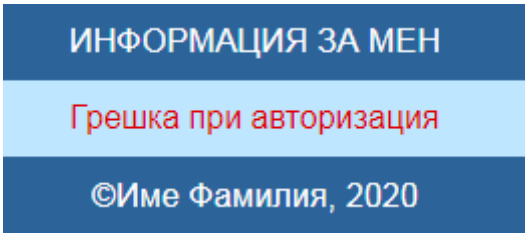
Показването на информация за грешката се реализира чрез функция showError(). Тя трябва да бъде във файл display.js. Тази функция очаква два аргумента: 1) Текстово описание на грешката и 2) Колко време в секунди да се вижда тази информация. Следва примерно съдържание на тази функция:

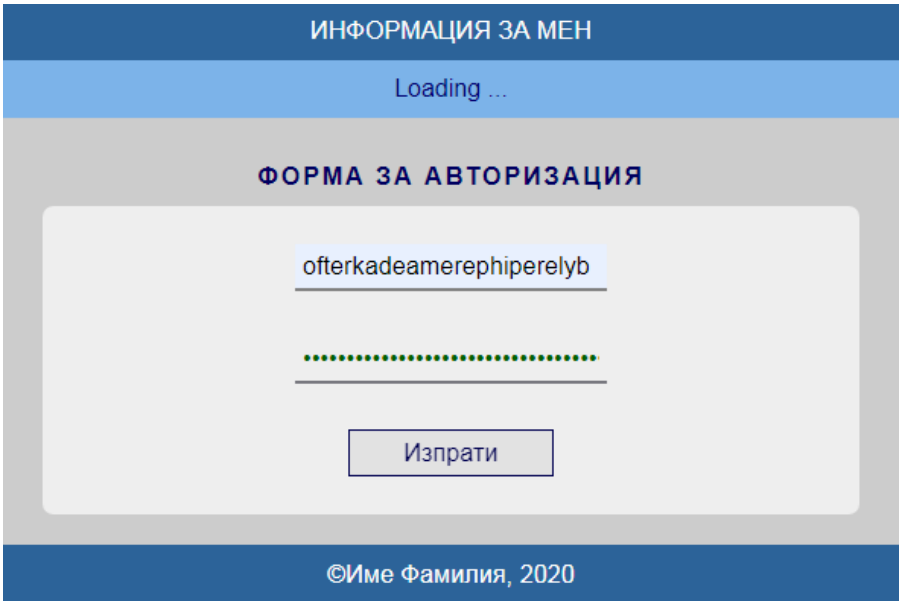
```
function showError(errorMsg, time) {
    var errorBox = $('#error-box');
    errorBox.text(errorMsg);
    view('error-box');
    setTimeout(function () {
        $('#error-box').fadeOut();
        showLoginForm();
    }, time*1000);
}
```

Чрез *програмен таймер* се задава кога информацията трябва да се скрие. Таймерът се стартира чрез функция setTimeout(). Когато времето (time*1000), зададено в ms изтече се скрива секцията със съобщението за грешка – fadeout(). Следва показване на формата за авторизация – showLoginForm().

Следва информация как работи приложението при възникване на грешки и при липса на грешки:







Всеки, който завърши проекта си го изпраща на адрес rosen@tugab.bg за проверка.