

Теоретичні відомості

MongoDB - нереляційна (**NoSQL**) об'єктно-орієнтована система управління базами даних (**СУБД**). Така база даних складається з набору колекцій, які в свою чергу містять набір об'єктів з довільним набором властивостей.

Mongoose - надбудова **ODM (Object Data Mapping)** для баз даних **MongoDB**.

Модель ODM Mongoose - об'єкт, який забезпечує роботу з колекцією бази даних (створення, пошук, додавання, знищення, оновлення даних). Модель дозволяє контролювати набір властивостей та типи даних для значень властивостей об'єктів, які може містити колекція.

Для вибірки даних на моделі (наприклад, **Model**) викликається метод **find**

```
Model.find(obj,function(err,data){  
  console.log(data)  
},
```

де **obj** - об'єкт пошуку, параметр **data** містить результат вибірки.

Для видалення даних на моделі викликається метод **remove**

```
Model.remove(obj,function(err,data){  
  console.log(data)  
},
```

де **obj** - об'єкт пошуку даних, які необхідно видалити.

Для оновлення даних на моделі викликається метод **update**

```
Model.update(obj1, obj2,function(err,data){  
  console.log(data)  
},
```

де **obj1** - об'єкт пошуку даних, **obj2** - об'єкт оновлення.

Для додавання даних в базу використовується метод **save**, який викликається на екземплярі моделі

```
var model=new Model(obj)  
model.save(function(err,data){  
  console.log(data)  
},
```

де **obj** - об'єкт з даними для запису в базу, в параметр **data** повертається записаний в базу об'єкт з унікальним ідентифікатором-властивістю **_id**

Хід роботи:

1) Реєструємось на сервісі <https://mlab.com> , який надає 500Мб безкоштовного хостінгу для нереляційних об'єктно-орієнтованих баз даних **MongoDB**.

2) Створюємо нову базу даних (**Create New -> Sandbox Free (Plan type) ->Continue ->Europe (AWS Region) -> Continue -> mydb (Database Name) -> Continue -> Submit Order**).

3) Заходимо в створену базу даних **mydb** і додаємо для неї користувача (кнопка **Add Database User** на закладці **Users**). Для прикладу, введемо наступні дані

database username: Admin

database password: Admin123

Дані створеного користувача будуть використовуватись при підключенні до бази даних з нашого проекту.

4) Інсталюємо **ODM Mongoose** для роботи з базою даних **MongoDB**

npm install mongoose --save-dev

5) В корені проекту створюємо файл-модуль **mongoose.js**, в якому будемо встановлювати з'єднання з базою даних

```
var mongoose=require('mongoose');
mongoose.Promise = global.Promise;
mongoose.connect('mongodb://Admin:Admin123@ds249992.mlab.com:49992/mydb');
console.log("mongodb connect...")
module.exports=mongoose;
```

В методі **mongoose.connect** задаємо стрічку підключення (connection string) до бази даних. Дану стрічку копіюємо з бази даних (**To connect using a driver via the standard MongoDB URI**) і змінюємо в ній параметри **<dbuser>** та **<dbpassword>** на реальні дані створеного раніше користувача.

6) Спроектуємо модель **User**, яка буде створювати в базі даних колекцію **users** і взаємодіяти з нею (вибірка, додавання, знищення та оновлення даних). В корені проекту створимо каталог **models** з файлом **user.js**

```
var mongoose=require('../mongoose');
var schemaUser=new mongoose.Schema({
  username:{
    type:String,
    unique:true,
    required:true
  },
  usage:{
    type:Number,
    required:true,
```

```

    min:18,
    max:70
  }, {versionKey:false})

var User=mongoose.model("User",schemaUser);
module.exports=User;

```

Дана модель буде здійснювати контроль типів та накладатиме обмеження на значення властивостей об'єктів з даними, які можуть міститись в колекції **users**.

7) Підключаємо модуль-модель **User** в файлі **server.js**

```

var User=require('../models/user');

```

8) Запускаємо сервер (файл **server.js**)

node server

При успішному запуску сервера (відсутні помилки) в консоль **Command Prompt** виведеться повідомлення про з'єднання з базою даних, в якій створиться колекція **users**. Назва колекції-це назва моделі в множині (**User** -> **users**).

9) В файлі **index.html** під'єднуємо **bootstrap.css** перед стильовим файлом **style.css**

```

<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous">
<link rel="stylesheet" href="style.css">

```

10) В файлі **index.html** створюємо форму для додавання користувачів в базу даних і додаємо **bootstrap**-класи для елемента **div** з **id='table'**

```

<body>
  <h1>MyApp</h1>
  <div class="row justify-content-center">
    <div class='myform col-6 text-center'>
      <input type="text" class='name form-control'
placeholder="inputName">
      <input type="text" class='age form-control'
placeholder="inputAge">
      <button class='btn btn-success add'>Add</button>
    </div>
  </div>
  <div id='table' class='row justify-content-center'></div>
</body>

```

11) Додаємо стилі в файл **style.css**

```
h1{
  background-color:brown;
  color:white;
  padding:20px;
  text-align: center;
}
.myform input{
  margin:5px;
}
table{
  margin:5px;
}
```

12) В файлі **client.js** додаємо **bootstrap**-класи **table**, **table-bordered**, **table-primary** та **col-6** в функцію створення таблиці **createTable(element,mass)**

```
function createTable(element,mass){
  $(element).empty();
  console.log(mass);
  $('<table>')
    .addClass("table table-bordered table-primary col-6")
    .appendTo(element);
  for(var i=0;i<mass.length;i++){
    $('<tr>').addClass('tr').appendTo('.table');
    for(var key in mass[i]){
      $('<td>').addClass('td')
        .appendTo('.tr:last').text(mass[i][key]);
    }
  }
}
```

13) Модифікуємо обробник **'/getusers'** на сервері. На моделі **User** викликаємо метод **find**, який вибратиме всі дані з колекції **users**. В параметр **data** отримуємо результат вибірки

```
app.get('/getusers',function(req,res){
  User.find(function(err,data){
    console.log(data);
    res.send(data);
  })
})
```

14) В файлі **client.js** створюємо функцію **addUser(name,age)**, яка приймає параметрами ім'я користувача та його вік. Дана функція відправлятиме **post**-запитом на сервер свої параметри у вигляді об'єкта

```
function addUser(name,age){
  if(!name||!age) return;
  var obj={
    username:name,
    userage:age
  }
  $.post('/adduser',obj,function(data){
```

```

        console.log(data);
        getUsers();
    })
}

```

15) Програмуємо подію **click** для кнопки **Add**, яка має клас **add**

```

$('.add').click(function(){
    var name=$('#.name').val();
    var age=$('#.age').val();
    $('#.name').val("");
    $('#.age').val("");
    addUser(name,age);
})

```

16) Інсталуємо модуль **body-parser** для **post**-запитів

npm install body-parser --save-dev

17) Підключаємо модуль **body-parser** в файлі **server.js** і прив'язуємо його до express-проекту

```

var bodyParser=require('body-parser');
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json())

```

18) В файлі **server.js** створюємо обробник клієнтського запиту **/adduser**, який буде додавати нового користувача в базу даних. Об'єкт з даними від клієнта буде доступний в **req.body**

```

app.post('/adduser',function(req,res){
    console.log(req.body);
    var user=new User(req.body);
    user.save(function(err,data){
        if(err) console.log(err.message);
        console.log(data);
        res.send('add user!');
    })
})

```

При записі в базу даних об'єкт отримує унікальну властивість-ідентифікатор (**_id**). Зверніть увагу, що метод **save** викликається на екземплярі моделі. Записаний в базу даних об'єкт повертається в параметр **data**.

19) В файлі **client.js** створюємо функцію **deleteUser(id)**, яка буде передавати на сервер унікальний **_id** користувача

```

function deleteUser(id){
    var obj={id:id};
    $.post('/deleteuser',obj,function(data){
        console.log(data);
        getUsers();
    })
}

```

20) Приховуємо програмно відображення першого стовпчика таблиці з властивістю `_id`. Також програмно додаємо в кінець таблиці стовчик, який міститиме в кожному рядку кнопку **Delete** для видалення користувача з бази даних. В функції **createTable** на ітерації зовнішнього циклу після внутрішнього циклу додаємо наступний фрагмент коду

```
$('.tr:last .td:first').hide();
$('.td').appendTo('.tr:last');
$('<button>').text('Delete').addClass('btn btn-danger')
.appendTo('td:last').click(function(){
    var id=$(this).parent().parent().find('td:first').text();
    console.log(id);
    deleteUser(id);
});
```

При кліку по кнопці **Delete** будемо зчитувати значення `_id` в прихованому стовпчику відповідного рядка і викликати написану раніше функцію **deleteUser**.

21) В файлі **server.js** створюємо обробник клієнтського запиту `/deleteuser`, який буде видаляти користувача по отриманому значенні `_id`.

```
app.post('/deleteuser',function(req,res){
    console.log(req.body);
    User.remove({_id:req.body.id},function(err,data){
        res.send('remove user');
    })
})
```

22) Реалізуйте оновлення існуючих даних в базі даних:

- В кожен рядок таблиці в функції **CreateTable** додайте програмно кнопку **Update**.
- При натисканні на цю кнопку дані користувача повинні зчитуватись з рядка таблиці в форму вводу даних, а назва кнопки **'Add'** повинна замінюватись на **'Update'**.
- При натисканні на кнопку **Update** модифіковані в полях вводу дані повинні оновлюватись в базі і браузері користувача, кнопки повертається назва **'Add'**.