Desarrollo Backend con NodeJS

Services y métodos de Mongoose

course[It]

Mongoose

Integrando mongo con node

- Mongoose
  - https://mongoosejs.com/
  - Mongoose es la librería que vamos a utilizar para conectar Mongo con nuestra aplicación en Node
  - npm install —save mongoose

Conectando node a la base de datos con mongoose

```
const mongoose = require("mongoose");


mongoose.connect('mongodb://localhost:27017/basedegatos', {
    useNewUrlParser: true
 });
```

Models

Models

Un modelo de datos es donde vamos a escribir el plan de nuestra collection. Esto quiere decir que vamos a decir que propiedades van a existir y que tipo de dato van a contener. En mongoose se escriben por medio de un **schema**

Utilizando schemas en node:

```
const mongoose = require("mongoose");
const testSchema = mongoose.Schema(
  {
    testProperty: {
      type: String,
    },
    anotherProperty: {
      type: Number,
      required: true
    },
    yetAnotherProperty: []
  }
);
module.exports = mongoose.model("Test", testSchema);
```

El nombre del modelo:

- Singular
- Capitalizado
- de tipo string

```
module.exports = mongoose.model("Test", testSchema);
```

Propiedades del modelo

- Required
  - En los modelos podemos indicarle la propiedad “required: true” a un campo para que a la hora de insertar un nuevo registro, mongoose lo considere obligatorio

```
anotherProperty: {
    type: Number,
    required: true
  },
```

Propiedades del modelo

- Default
  - En los modelos podemos indicarle la propiedad “default: XXX” a un campo para que a la hora de insertar un nuevo registro, mongoose le ponga por defecto este valor en caso de no tener esa información

```
anotherProperty: {
    type: Boolean,
    default: false
  },
```

Propiedades del modelo

- Enum
  - En los modelos podemos indicarle la propiedad “enum: [‘a’, ‘b’, ‘c’]” a un campo para que a la hora de insertar un nuevo registro, mongoose valide que el valor ingresado es alguno de los pertenecientes al array enum

```
anotherProperty: {
    type: Number,
    enum: ["user", "admin", "teacher"],
    default: "user"
  },
```

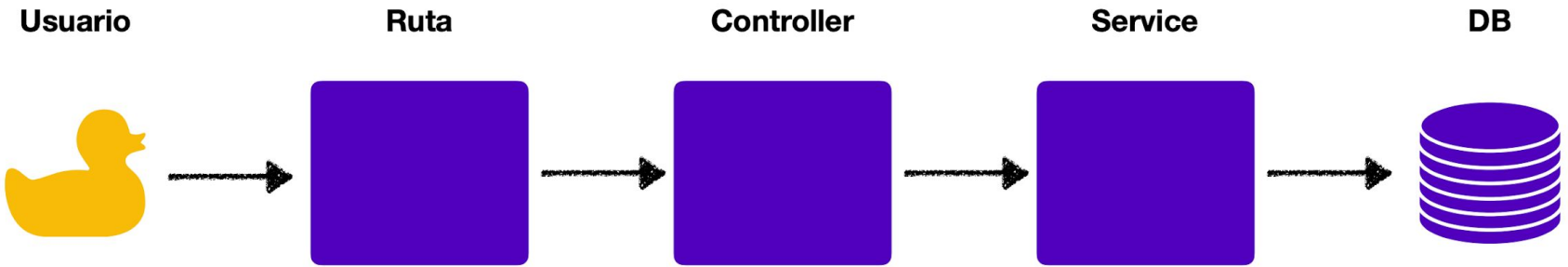
Services

Services

Los services son el espacio de código donde viven todas las funciones que interactúan directamente con nuestra base de datos

“Detrás de todo gran controller siempre hay un gran service”

-- Bel Rey, filósofa contemporánea y desarrolladora fullstack



Services

Los services son el espacio de código donde viven todas las funciones que interactúan directamente con nuestra base de datos

“Detrás de todo gran controller siempre hay un gran service”

-- Bel Rey, filósofa contemporánea y desarrolladora fullstack

Services

```
class UserService {
  getUsers() {
    const query = User.find();
    return query;
  }
}

module.exports = UserService;
```

En el controller: Utilizamos el service como variable de clase

```
class UserController {
  constructor(userService) {
    this.userService = userService
  }

  async getUsers(req, res) {
    const users = await this.userService.getUsers()
    return res.json(users)
  }
}

module.exports = UserController;
```

En las rutas: Instanciamos el service

```
const UserController = require("./../controllers/userController");
const UserService = require("./../services/userService");
const UserInstance = new UserController(new UserService());
```

Métodos de Mongoose

Métodos de Mongoose

- Model.find()
  - Nos va a devolver un array con todos los resultados de ese modelo

```
getUsers() {
  const query = User.find().exec();
  return query;
}
```

Métodos de Mongoose

- Model.findOne(filter)
  - Nos va a devolver un único objeto en base al resultado de nuestro filtro

```
getUserById(id) {
    const query = User.findOne({ _id: id }).exec();
    return query;
 }
```

Métodos de Mongoose

- Model.findOneAndUpdate(FILTER, DATA)
  - Va a modificar un único objeto en base al resultado del filtro con el valor de “data”. Los campos que no se encuentran en data, no se modifican

```
updateUser(id, data) {
      const user = User.findOneAndUpdate({ _id: id}, data).exec();
      return user
  }
```

Métodos de Mongoose

- Model.deleteOne(FILTER)
  - Borra una única entrada que cumpla con la condición pactada en el filtro

```
deleteUser() {
    const query = User.deleteOne({ name: 'Bel,' }, function (err) {
        if(err) console.log(err);
        console.log("Successful deletion");
      });
    return query
  }
```

Métodos de Mongoose

- Model.save(DATA)
  - Crea una entrada en nuestra base de datos con la data provista

```
addUser(data) {
  const newUser = new User(data);
  return newUser.save();
}
```