

Tópicos Avançados em Programação I

Prof. Me. Marcos Alves

marcos@ucdb.br

LABORATÓRIO

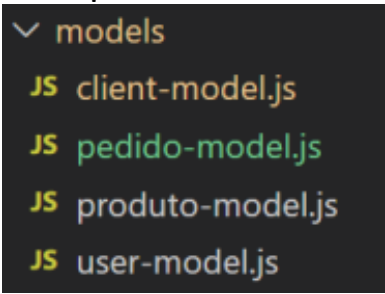
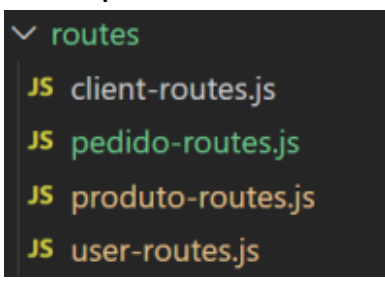
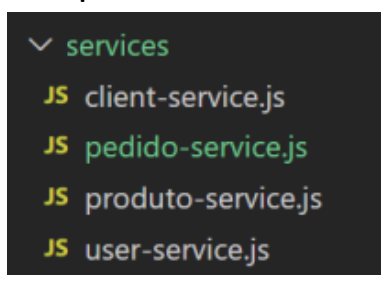
*Criando uma estrutura **Back-End**
– API com Web Services, JavaScript e
MongoDB*

O objetivo deste laboratório é colocar em prática os conhecimentos abordados durante o semestre. Este será o **ÚNICO TRABALHO** a ser utilizado para avaliação do aluno. Parte da nota será dada pelas entregas parciais durante as próximas aulas e apresentação da versão final conjunta com a disciplina de RAD1.

Para tanto, cada aluno deverá:

- 1º. Criar sua própria *branch* (colocar seu nome para facilitar a identificação) no gitHub, clonando o atual projeto ***back-end*** feito em aula;
- 2º. Vamos completar nosso projeto que hoje conta apenas com a *collection* Client, para incluirmos também as seguintes *collections*: User, Produto, Pedido (detalhadas a seguir).
- 3º. Para facilitar o trabalho, a seguir será apresentada a estrutura que vocês deverão reproduzir, bem como alguns trechos de código que irão orientá-los e ajudá-los na hora de montar os arquivos.

- **Estrutura de pastas/diretórios**

Na pasta MODELS :	Na pasta ROUTES :	Na pasta SERVICES :
		

- **Schemas para collections**

a) *Collection* User

```
(name, age, email, password (mínimo 3, máximo 8 caracteres), phone)
```

- expressão regular para email:
→ match: `/^[a-z0-9.]+@[a-z0-9]+\.[a-z]+(\.[a-z]+)?$/i`
- expressão regular para phone:
→ match: `/(?:\([0-9]{2}(?:\))\s?[0-9]{5}(?:-[0-9]{4})?$/i }`

b) *Collection* Produto

```
(desc, valor, quant)
```

c) *Collection* Pedido

Nesta *collection* entrego o *schema* completo, pois é mais complexo, e é primeira vez que faremos uma implementação de vínculo entre *collections* e também do tipo *one-to-many* (1:N).

```
produtos: [{  
  type: Schema.Types.ObjectId,  
  ref: "Produto"  
}],  
dataPedido: { type: Date, require: true, default: Date.now },  
client: {  
  type: Schema.Types.ObjectId,  
  ref: "Client"  
},  
status: { type: String, require: true, default: "pendente" }
```

- **Sobre as Rotas**

a) Rotas de User

```
const { check, body, validationResult } = require('express-validator')

router.post('/', [body('password').isLength({ min: 3, max: 8 }).withMessage('A senha deve ter entre 3 e 8 digitos')],
  async (req, res) => {
    let data = { name: req.body.name, age: req.body.age, email: req.body.email, password: req.body.password, phone: req.body.phone }
    const errors = validationResult(req)
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() })
    } else {
      let user = await userService.save(data)
      res.json(user)
    }
  })

router.get('/list', async (req, res) => {})
router.get('/maiores', async (req, res) => {})
router.get('/menores', async (req, res) => {})
router.get('/list/:search', async (req, res) => {})
router.delete('/excluir/:id', async (req, res) => {})
router.put('/alterar/:id', async (req, res) => {})
```

b) Rotas de Produto

```
const { check, body, validationResult } = require('express-validator')

router.post('/', [], async (req, res) => {
  let data = { desc: req.body.desc, valor: req.body.valor, }
  const errors = validationResult(req)
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() })
  } else {
    let prod = await produtoService.save(data)
    res.json(prod)
  }
})

router.get('/list', async (req, res) => {})
router.delete('/excluir/:id', async (req, res) => {})
router.put('/alterar/:id', async (req, res) => {})
```

c) Rotas de Pedido

Esta faremos mais pra frente.

- **Sobre os Services**

- a) *Services* para User

```
class UserService {  
  async save(data) {}  
  async list() {}  
  async listMaiores() {}  
  async listMenores() {}  
  async search(s) {} // by name  
  async delete(id) {}  
  async change(id, update) {}  
}
```

- b) *Services* para Produto

```
class ProdutoService {  
  async save(data) {}  
  
  async list() {}  
  async delete(id) {}  
  
  async change(id, update) {}  
}
```

- c) *Services* para Pedido

Este faremos mais pra frente.

- **Sobre o Server.js**

- a) Preparar os *requires* para usuários e produtos

```
const userRoutes = require('./routes/user-routes.js')  
const produtoRoutes = require('./routes/produto-routes.js')
```

- b) Inserir as rotas para usuários e produtos

```
app.use('/user', userRoutes)  
app.use('/produto', produtoRoutes)
```

- c) Para Pedidos, ainda não.