2021/04/26 - AULA 05.2

Laboratório de Sistemas Computacionais Complexos

https://uclab.xyz/sistemas-complexos-2021-aula05-2

Renato Cordeiro Ferreira
renatocf@ime.usp.br

João Francisco Daniel
joaofran@ime.usp.br

Alfredo Goldman
gold@ime.usp.br

Thatiane de Oliveira Rosa
thatiane@ime.usp.br

Em caso de dúvidas

Acessem www.slido.com com #complexos

ou

Agenda

Tema da aula:
Node.js & Express

1. REST e APIs *RESTful*
2. *Hello, Express!*
3. Encadeamentos e *middlewares*
4. Roteadores

REST e APIs *RESTful*

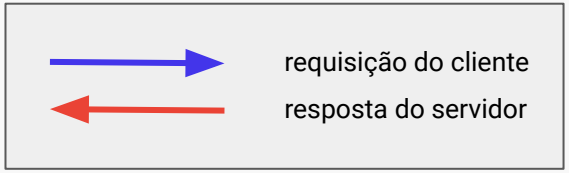
REST e APIs *RESTful*

O que é REST:
Representational State Transfer (REST) é uma arquitetura proposta por Thomas Fielding em sua tese de doutorado, nos anos 2000.

1. construída em cima do protocolo HTTP

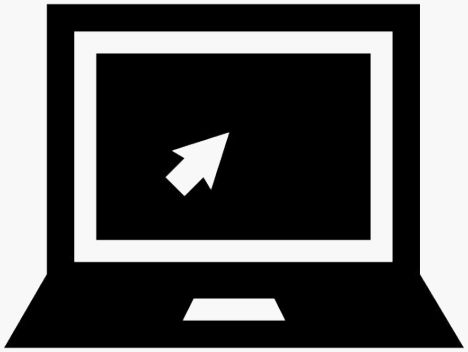
2. princípios
    a. comunicação cliente-servidor sem estado
    b. conjunto de operações bem definidas sobre todos os recursos da aplicação
    c. identificação universal dos recursos por uma URI
    d. uso de *hipermidia* para ligações entre recursos e entre estados

REST e APIs *RESTful*



Na prática:

Vamos considerar o servidor de um restaurante



`GET /sucos`

a lista de todos os sucos

`GET /sucos/42`

todas as informações sobre o suco com *id*=42
(tamanhos, preços, adicionais, etc)

Hello, Express!

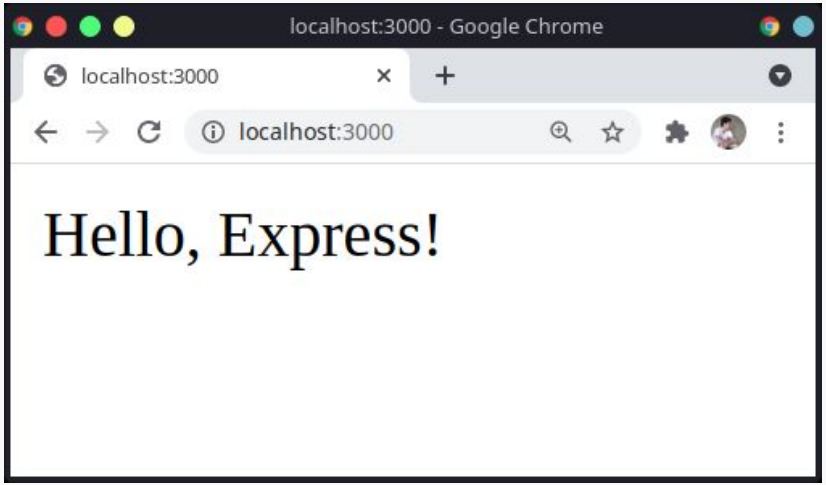
Hello, Express!

Express:
Framework Node.js para construção de aplicações Web que sejam *RESTful*

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello, Express!')
})

app.listen(port, () => {
  console.log(`Running at :${port}`)
})
```



Hello, Express!

Node.js:
Alguns aspectos que dão suporte ao código

1. `const port = 3000`
2. `const express = require('express')`
3. `( … ) => { … }`

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello, Express!')
})

app.listen(port, () => {
  console.log(`Running at :${port}`)
})
```

Hello, Express!

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello, Express!')
})

app.listen(port, () => {
  console.log(`Running at :${port}`)
})
```

Express:

1. `const app = express()`
2. `app.get(ROUTE, HANDLER)`
   a. outros métodos irmãos
   b. geral: `app.METHOD(ROUTE, HANDLER)`
3. `app.listen(PORT, CALLBACK)`

Hello, Express!

```
app.get('/sucos', (req, res) => {
  // ...
  res.json({ sucos: todosSucos })
})

app.get('/sucos/:id', (req, res) => {
  const sucoID = req.params.id
  // ...
  res.json({ suco: umSuco })
})
```

Express:

1. `res.json( … )`
2. parametrização
   a. `'/sucos/:id'`
   b. `req.params.id`
3. query string
   a. `GET /sucos?offset=4&limit=4`
   b. `req.query //{ offset: 4, limit: 4 }`
4. `req.body`

Encadeamento e *middlewares*

Encadeamento e *middlewares*

Node.js:
Alguns aspectos que dão suporte ao código

1. `module.exports = …`
2. `const logger = require('./logger')`

```
const logger = require('./logger');

app.get('/', logger, (req, res) => {
  // ...
})
```

```
function logger(req, _, next) {
  const method = req.method
  const fullpath = req.baseUrl + req.path

  const now = new Date()
  console.log(`
    [${now}]
    --> ${method} ${fullpath}

  `)
  next()
}

module.exports = logger
```

Encadeamento e *middlewares*

```
function logger(req, _, next) {
  const method = req.method
  const fullpath = req.baseUrl + req.path

  const now = new Date()
  console.log(`
    [${now}]
    --> ${method} ${fullpath}

  `)
  next()
}

module.exports = logger
```

Express:

1. `app.get(ROUTE, HANDLER1, HANDLER2)`
2. `(req, res, next) => { … }`

```
const logger = require('./logger');

app.get('/', logger, (req, res) => {
  // ...
})
```

Encadeamento e *middlewares*

***Middleware* e aspectos:**
Algumas funcionalidades implementadas podem ser aspectos transversais à lógica de negócio ou podem ser adotadas repetidamente

1. autorização de requisições
2. validação de dados (exemplo: Criação e Atualização de um tipo de dado)
3. logging
4. ...

Roteadores

Roteadores

```
const express = require('express')
const app = express()
const juicesRouter = express.Router()
const dealsRouter = express.Router()

juicesRouter.get('/', ( ... ) => {
  // ...
  res.json({ sucos: todosSucos })
})

dealsRouter.get('/', ( ... ) => {
  // ...
  res.json({ combos: todosCombos })
})

app.use('/sucos', juicesRouter)
app.use('/combos', dealsRouter)

app.listen( ... )
```

Express:

1. `const juicesRouter = express.Router()`
2. `app.use('/sucos', juicesRouter)`

Roteadores também são *middlewares*

Roteadores também são *middlewares*

```
const express = require('express')
const logger = require('./logger')
const app = express()
const juicesRouter = express.Router()

joicesRouter.use(logger)

juicesRouter.get('/', ( ... ) => {
  // ...
  res.json({ sucos: todosSucos })
})

app.use('/sucos', juicesRouter)
```

Express:

1. `app.use(logger) // sintaxe alternativa`
2. `app.use('/sucos', juicesRouter)`
3. recomendação **forte:**

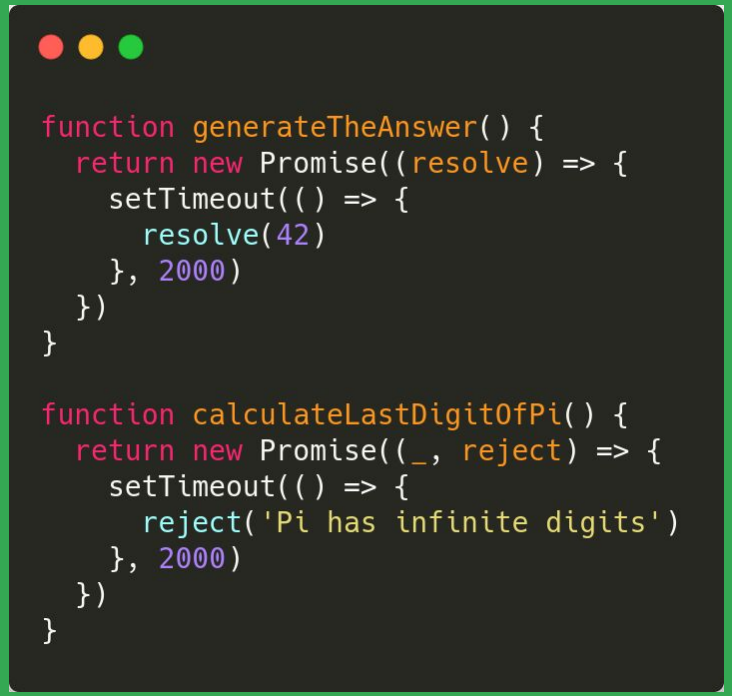
   ler mais sobre o `app.use( … )`

Mais de Node.js

Mais de Node.js

Node.js:

1. Promises
2. async/await



```
function generateTheAnswer() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(42)
    }, 2000)
  })
}

function calculateLastDigitOfPi() {
  return new Promise((_, reject) => {
    setTimeout(() => {
      reject('Pi has infinite digits')
    }, 2000)
  })
}
```

Mais de Node.js

Node.js:

1. Promises
2. async/await

```
function generateTheAnswer() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(42)
    }, 2000)
  })
}

function calculateLastDigitOfPi() {
  return new Promise((_, reject) => {
    setTimeout(() => {
      reject('Pi has infinite digits')
    }, 2000)
  })
}
```

```
function main() {
  const theAnswer = generateTheAnswer()
  console.log(theAnswer)

  const lastDigitOfPi = calculateLastDigitOfPi()
  console.log(lastDigitOfPi)
}

main()
```

Mais de Node.js

Node.js:

1. Promises
2. async/await

```
function generateTheAnswer() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(42)
    }, 2000)
  })
}

function calculateLastDigitOfPi() {
  return new Promise((_, reject) => {
    setTimeout(() => {
      reject('Pi has infinite digits')
    }, 2000)
  })
}
```

```
function main() {
  generateTheAnswer()
    .then((theAnswer) => {
      console.log('The answer: ', theAnswer)
    })

  calculateLastDigitOfPi()
    .then((lastDigitOfPi) => {
      console.log('A math miracle happened!')
      console.log('The last digit of Pi: ', lastDigitOfPi)
    })
    .catch((error) => {
      console.log(error)
    })
}

main()
```

Mais de Node.js

Node.js:

1. Promises
2. async/await

```
function generateTheAnswer() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(42)
    }, 2000)
  })
}

function calculateLastDigitOfPi() {
  return new Promise((_, reject) => {
    setTimeout(() => {
      reject('Pi has infinite digits')
    }, 2000)
  })
}
```

```
async function main() {
  const theAnswer = await generateTheAnswer()
  console.log('The answer: ', theAnswer)

  try {
    const lastDigitOfPi = await calculateLastDigitOfPi()
    console.log('A math miracle happened!')
    console.log('The last digit of Pi: ', lastDigitOfPi)
  } catch (error) {
    console.log(error)
  }
}

main()
```

Licença

Créditos

Imagens usadas nesta apresentação são provenientes de: freepik.com

Frequência

Senha do Estudante: **585fdi**

2021/04/26 - AULA 05.2

Laboratório de Sistemas Computacionais Complexos

https://uclab.xyz/sistemas-complexos-2021-aula05-2

Renato Cordeiro Ferreira
renatocf@ime.usp.br

João Francisco Daniel
joaofran@ime.usp.br

Alfredo Goldman
gold@ime.usp.br

Thatiane de Oliveira Rosa
thatiane@ime.usp.br