

**GRUPO 1**

1. [1,5] Dadas as assinaturas das seguintes funções que seguem o idioma comum em NodeJS, classifique numa frase a forma como cada função pode ser implementada no que diz respeito à sua natureza síncrona ou assíncrona. Nessa classificação use uma das palavras: SEMPRE, PODE, NUNCA.  
Exºs: fx NUNCA é implementada de forma síncrona; fx PODE ser implementada de forma síncrona; fx é SEMPRE implementada de forma assíncrona.
- a. [0,5] f1(p1, p2, cb) // retorna undefined
  - b. [0,5] f2(p1, p2) // retorna um valor numérico
  - c. [0,5] f3(p1, p2) // retorna uma promise
2. [5] A função `promisify(fn)` recebe como argumento `fn` uma função que recebe N argumentos, em que o último é uma função de *callback* `cb`, com o idioma convencionado em NodeJS. Retorna uma nova função que recebe os mesmos N-1 primeiros argumentos e retorna uma promise que será concluída com sucesso ou com erro, consoante a função de callback é invocada indicando sucesso ou erro, respetivamente. A listagem seguinte apresenta um exemplo de utilização desta função.

```
function f(success, a,b,c, cb) {  
  console.log(arguments)  
  setTimeout(() => {  
    if(success)  
      cb(null,a+b+c)  
    else  
      cb("err")  
  }, 2000);  
}
```

```
function showPromiseResult(p) {  
  p.then((res) => console.log(`Result is ${res}`))  
    .catch((err) => console.log(`Error is ${err}`))  
}  
let fp = promisify(f)  
showPromiseResult(fp(true, 1,2,3))  
showPromiseResult(fp(false, 1,2,3))
```

- a. [2,5] Implemente a função `promisify` de modo ter o comportamento indicado.

NOTA: A função `promisify` deve funcionar com qualquer função que respeite os requisitos enunciados e não apenas com a função `f` do exemplo.

- b. [1] Indique o output na consola da execução do código na listagem anterior.
- c. [1,5] A execução do código da listagem seguinte, apresenta na consola o resultado abaixo da listagem:

```
class MyClass {  
  constructor(success, a,b,c,) {  
    this.success = success; this.a = a; this.b = b; this.c = c  
  }  
  m(cb) {  
    f(this.success, this.a, this.b, this.c, cb)  
  }  
}  
let mc = new MyClass(true, 4,5,6)  
let fm = promisify(mc.m) // [1]  
showPromiseResult(fm())
```

Error is TypeError: Cannot read property 'success' of undefined

Alterando apenas a assinatura da função `promisify` e uma linha de código na implementação realizada anteriormente, corrija o erro de modo a que o resultado seja o esperado. Indique também a alteração necessária na chamada à função na linha [1] da listagem anterior, de modo o que o erro não ocorra.

NOTA: Indique apenas as alterações. Não reimplemente na totalidade a função `promisify`.

3. [1,5] Indique o que faz cada uma das execuções programa npm:
  - a. [0,5] npm install
  - b. [0,5] npm install express --save
  - c. [0,5] npm install webpack@4.27.1 --save-dev --save-exact
4. [2] Uma Web API tem a seguinte documentação relativos aos seus endpoints:
  - a. [1] Obtem o recurso com identificador id - GET /api/resources/{id}  
*Body do pedido:* vazio  
*Status codes:*
    - 201 Sucesso. A representação do recurso segue no body no formato application/json
    - 404 Erro no pedido. O cliente enviou um pedido inválido
    - 400 Erro no pedido. Não existe o recurso com o identificador id
  - b. [1] Cria ou atualiza um novo recurso com identificador id - PUT /api/resources/{id}  
*Body do pedido:* A representação do recurso a criar  
*Status codes:*
    - 200 Sucesso. O recurso foi criado com sucesso
    - 404 Erro no pedido. O cliente enviou um pedido inválido
    - 409 Conflito. O recurso com o identificador id já existe

Para cada endpoint indique as incorreções e omissões que deteta na documentação.

---

## GRUPO 2

5. [10] Considere o módulo gridView.js que acede aos recursos de um URL via HTTP (e.g. movies, leagues, ou outros...) manipulando-os através de uma tabela HTML.  
Este URL disponibiliza as operações CRUD e pressupõe-se que o recurso tem sempre uma propriedade id.  
A Figura 1 do Anexo, apresente dois casos de uso de gridView para dois URLs distintos: /api/laptops e /api/students.
  - Clicando sobre um célula (elemento td) é apresentado uma caixa de input que ao ser submetida actualiza visualmente o valor na **tabela** e também o **recurso** dado pelo URL.
  - Clicando no cabeçalho da coluna ordena a tabela pela respectiva coluna.

O módulo gridView.js tem a seguinte implementação:

```
const view = Handlebars.compile(require('./../views/gridView.hbs'))
module.exports = async (divMain, url) => {
  let dataSource
  try{
    const options = {method: 'GET', credentials: 'same-origin'}
    const resp = await fetch(url, options)
    const body = await resp.json()
    if(resp.status !== 200) throw body
    dataSource = body
    render()
  } catch(e) { alert(e) }
  function render() {
    const headings = Object.keys(dataSource[0])
    const items = dataSource.map(item => {return { 'row': Object.values(item)}})
    divMain.innerHTML = view({'headings': headings, 'items': items})
    divMain
      .querySelectorAll('th')
      .forEach(listenerOnHeading)
    divMain
      .querySelectorAll('tbody tr')
      .forEach((tr, idx) => {
        tr
          .querySelectorAll('td')
          .forEach((td, col) => listenerOnData(td, idx, col))
      })
  }
  function listenerOnHeading(th, colIdx) {...}
  function listenerOnData(td, rowIdx, colIdx) { /*Adds on td a click listener that calls renderInput()*/ }
  function renderInput(td, rowIdx, colIdx) {...}
}
```

a) [2] Tendo em conta o aspecto visual da aplicação, comportamento da função `render()` e respeitando o formato do objecto de contexto escreva a implementação da view `handlebars gridView.hbs`.

b) [3] Implemente a função `listenerOnHeading()` que faz com que o `click` sobre o título das colunas ordene a tabela pela respectiva coluna. Note que `dataSource` deve ficar coerente com o estado da tabela. Pode usar o método `Array.sort(cmp: (o1, o2) => Number)`.

c) [5] Complete as linhas assinaladas de 1 a 10 da função `renderInput()` que é chamada por acção do `click` sobre um elemento `td` da tabela, implementando o comportamento descrito no início desta questão.

Caso falhe a actualização do recurso no URL, deve ser reposto o valor original de acordo com **exemplo** apresentado na Figura 2 do Anexo.

```
function renderInput(td, rowIdx, colIdx) {
  td.innerHTML = _____ [1]
  td
    .querySelector(_____ [2])
    ._____ => { [3]
      ev.preventDefault()
      const val = _____ [4]
      td.innerHTML = val
      const item = _____ [5]
      const prop = _____ [6]
      const backup = _____ [7]
      item[prop] = isNaN(val) ? val : Number(val)
      listenerOnData(td, rowIdx, colIdx)
      const options = {
        _____ [8]
        _____
        _____
        _____
      }
      const path = _____ [9]
      const resp = await fetch(path, options)
      if(resp.status !== 200) {
        _____ [10]
        _____
        _____
        _____
      }
    }
}
```

Duração: 2 horas  
ISEL, 6 de Fevereiro de 2019

## Anexo

Figura 1

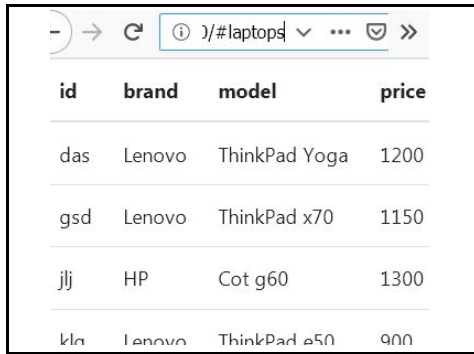
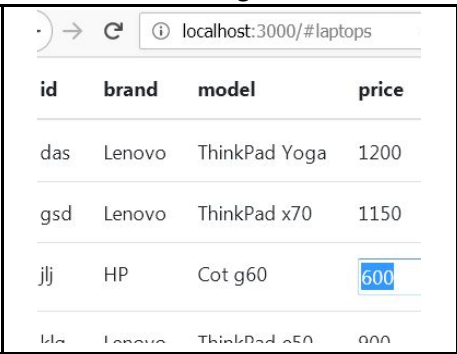
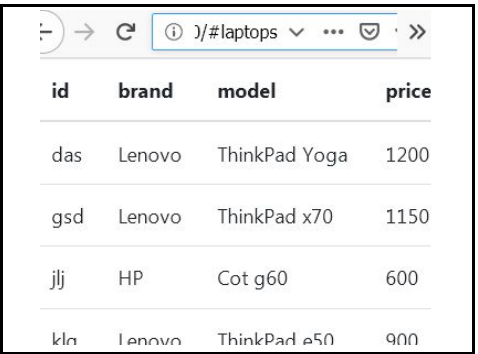
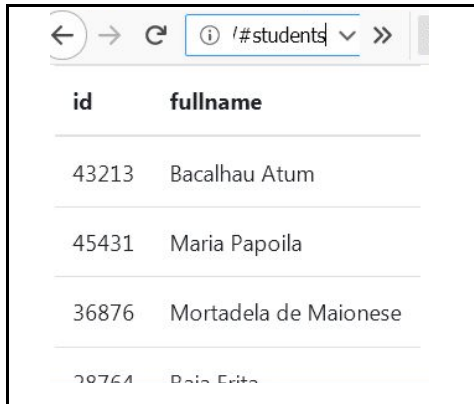
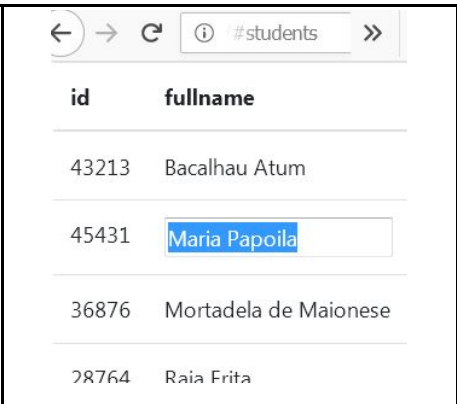
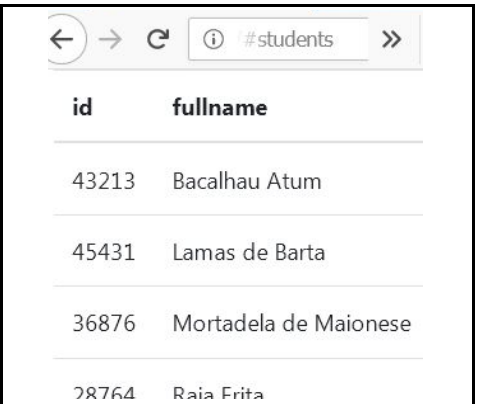
		
		

Figura 2

