

LOG CENTRAL

eduardogamebooster@gmail.com

Eduardo Bento

ericasugui@gmail.com

Erica Suguimoto

ever@azul.dev

Everton Jesus

fbzpanatto@hotmail.com

Fabrizio Panatto

Projeto

Esta aplicação tem por objetivo centralizar todos os registros de erros de quaisquer outras aplicações, oferecendo consultas que permitam monitoramento e possível tratamento de acordo, por exemplo, com volume e gravidade das ocorrências.

```
JS index.js ×
api > src > services > mailer > JS index.js > ...
1  const nodemailer = require('nodemailer')
2  const { settings } = require('.../config')
3
4  class Mailer {
5    constructor() {
6      this.transport = nodemailer.createTransport({
7        service: settings.mailer.service,
8        auth: settings.mailer.auth
9      })
10   }
11
12   send(to, subject, html) {
13     return new Promise((resolve, reject) => {
14       try {
15         this.transport.sendMail({
16           from: settings.mailer.from,
17           to,
18           subject,
19           html
20         }, error => {
21           if (error) {
22             throw error
23           }
24
25           resolve()
26         })
27       } catch (error) {
28         reject(error)
29       }
30     })
31   }
32 }
33
34 module.exports = new Mailer()
```

A construção da API entregue foi idealizada e realizada de maneira que um front-end para a aplicação deva tratar apenas da integração entre as ações do usuário e a própria API. Ou seja, os registros de erros devem ser entregues ao front-end de forma prática, eximindo da interface com o usuário a necessidade de realizar tarefas como, por exemplo, ordenação e paginação de informações.

Tecnologias



Sequelize



PM2



NGINX



Amazon
RDS



Nodemailer



TM



Jest

Express



MySQL®



amazon
web services™

EC2



JWT



Swagger™

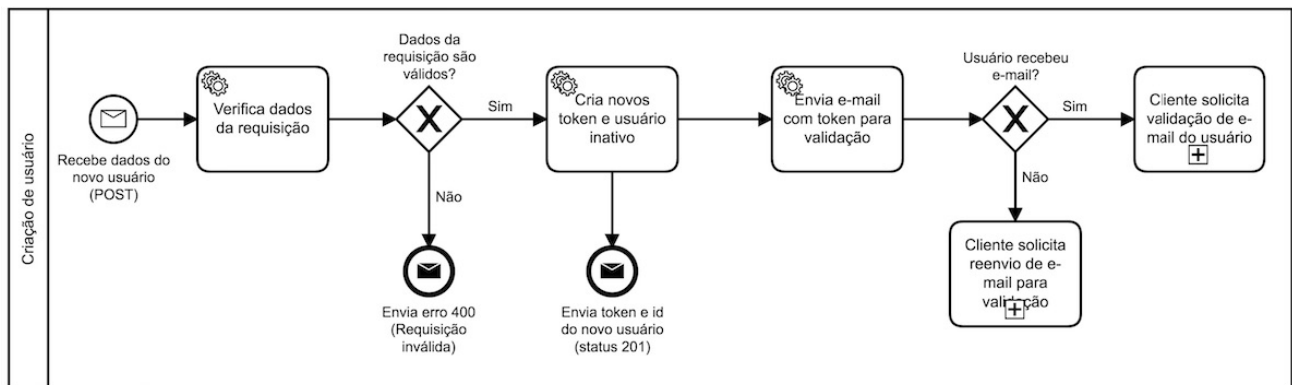
Aplicação

Autenticação

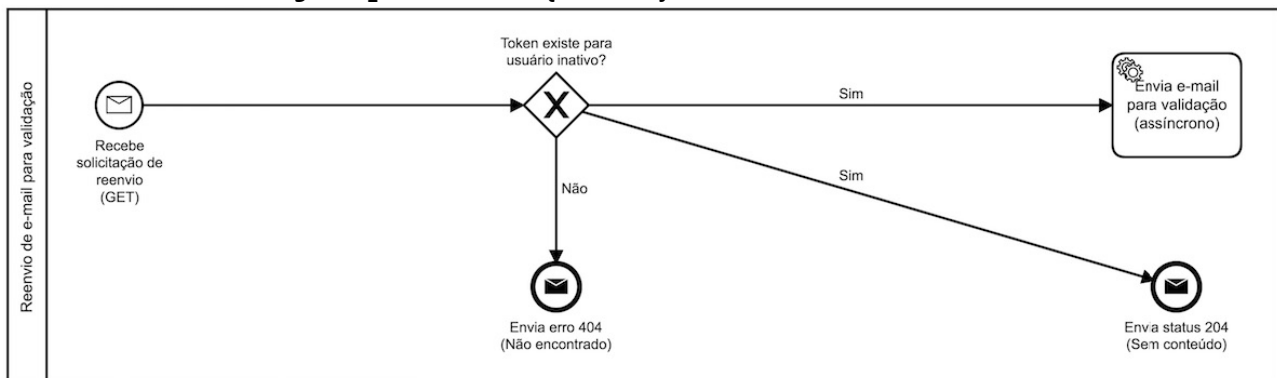
Pela natureza simples da aplicação proposta, o único conjunto de funcionalidades que contém regras de negócio mais complexas é o relacionado à autenticação, desde a criação de usuário até detalhes como validação do e-mail e recuperação de senha. Então, discorreremos sobre o processo, que é parte significativa da aplicação e faz uso de vários componentes nela implementados.

A seguir, fluxos resumidos para cada etapa, destacando o *endpoint* para acompanhamento e/ou testes via Swagger:

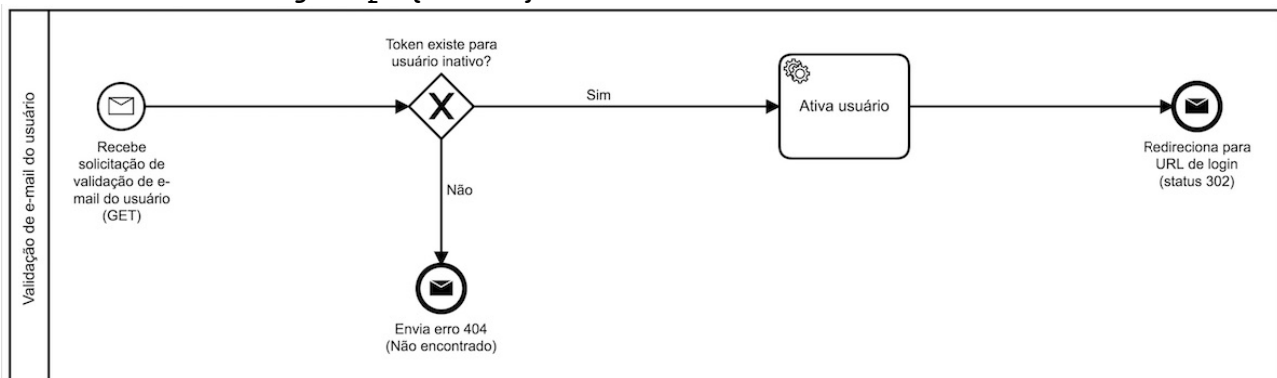
POST /User/sign-up



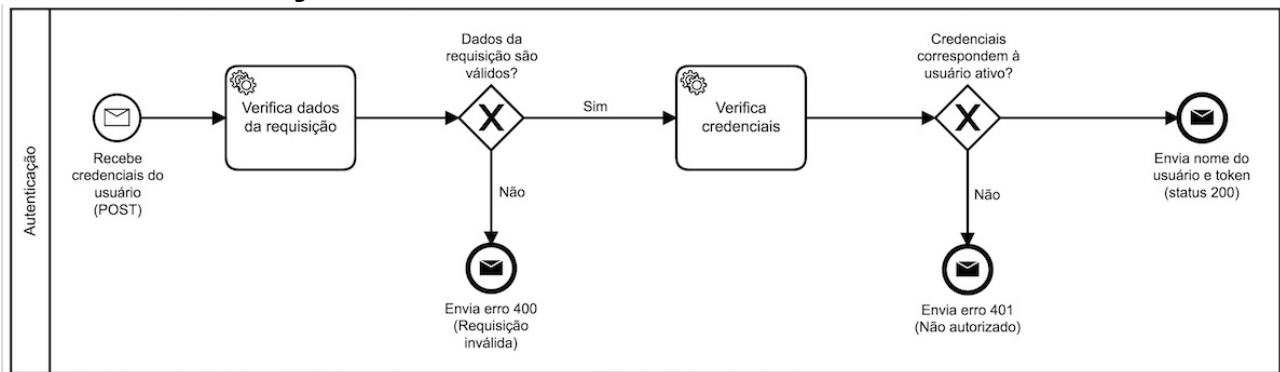
GET /User/sign-up/resend/{email}



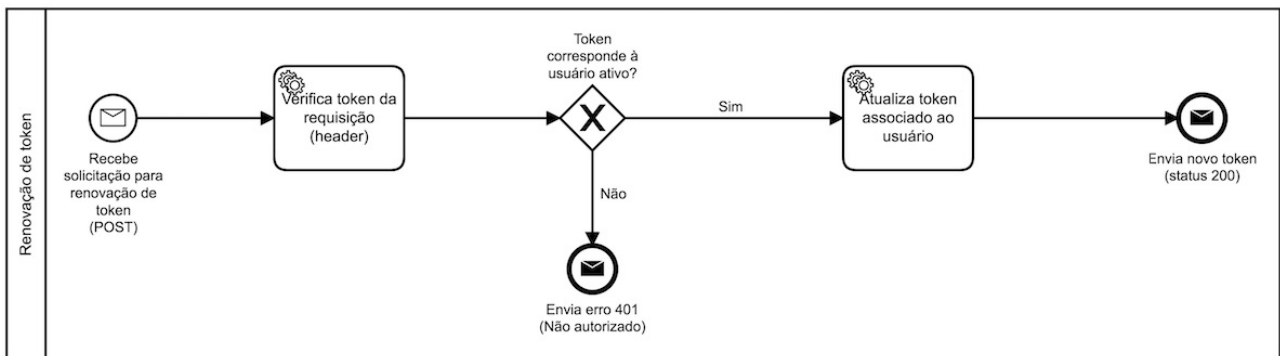
GET /User/sign-up/{token}



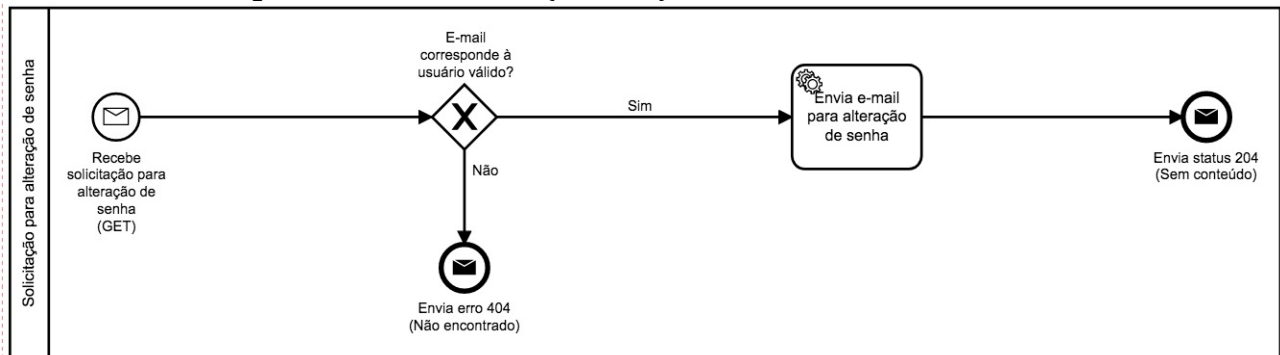
POST /User/sign-in



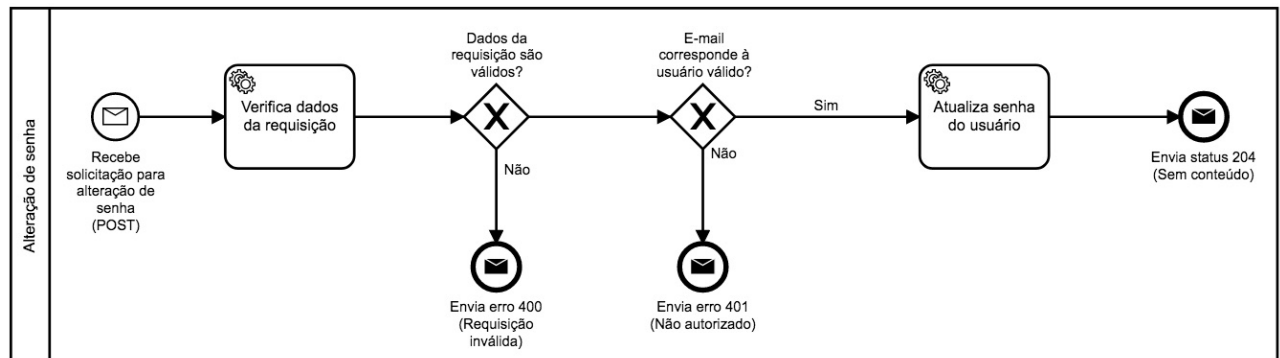
POST /User/token



GET /User/password-reset/{email}



POST /User/password



Logs

Também abordaremos a inserção e consulta dos logs, entidade central da aplicação, demonstrando os relacionamentos entre esta e as entidades de domínio criadas para assegurar a normalização da estrutura de dados.

Dentre as funcionalidades a demonstrar via Swagger, cabe destacar as funções de paginação, ordenação e filtros para a consultas destes logs, todas efetuadas a partir de *query parameters*.

```
{
  "application": {
    "id": 0
  },
  "environment": {
    "id": 0
  },
  "method": {
    "id": 0
  },
  "level": {
    "id": 0
  },
  "sourceAddress": "string",
  "datetime": 0,
  "url": "string",
  "message": "string",
  "data": "string"
}
```

Para inserção (via POST), o body enviado para o endpoint da entidade Log é composto conforme representado ao lado, indicando apenas a chave primária (campo id) de cada entidade associada.

```
Log {
  id integer
  url string
  message* string
  data json
  datetime string($date-time)
  default: CURRENT_TIMESTAMP
  sourceAddress* string
  localDatetime string($date-time)
  default: CURRENT_TIMESTAMP
  archived boolean
  default: false
  createdAt string($date-time)
  updatedAt string($date-time)
  Application {
    id integer
    name* string
    active boolean
  }
  Environment {
    id integer
    label* string
  }
  Level {
    id integer
    label* string
  }
  Method {
    id integer
    label* string
  }
  User {
    id integer
    name* string
    email* string
    active boolean
  }
}
```

Já para as consultas (via GET), além da respectiva chave primária, cada entidade associada é retornada também contendo os campos exclusivos e/ou obrigatórios.

Middleware para autenticação

Todos os *endpoints* da aplicação são submetidos ao *middleware* exclusivo para autenticação, garantindo a segurança via login/e-mail e senha.

O acesso sem autenticação é permitido apenas para *endpoints* especificados como exceções no arquivo de configurações da aplicação.



```
JS auth.js ×
api > src > middlewares > auth.js > AuthHandler > constructor > this.middleware > finally() call
1  const Config = require('../config')
2  const Log = require('../services/log')
3  const User = require('../models/user')
4  const Auth = require('../services/auth')
5
6  class AuthHandler {
7    constructor() {
8      this.middleware = (request, response, next) => {
9        try {
10         if (Config.settings.auth.exceptions.map(ex => ex.url).includes(request.url) ||
11             Config.settings.auth.exceptions.findIndex(ex => ex.children && request.url.st
12             next()
13         } else {
14           const token = Auth.getToken(request)
15
16           if (token) {
17             Auth.verifyToken(token)
18             .catch(error => Log.send(request, response, next, error, 401))
19             .finally(() => {
20               User
21                 .findOne({ where : { token, active: true }, attributes: [ 'id' ] })
22                 .then(user => {
23                   if (user) {
```

Utilizando o padrão JWT, o gerenciamento de acesso é efetuado sem registro persistente da sessão do usuário no servidor.

Configurações

Um arquivo *JSON* concentra as principais configurações da aplicação, desde informações para a conexão ao banco de dados até o intervalo para expiração dos tokens.



The image shows a code editor with a file named `settings.json`. The file contains a JSON configuration object. A yellow arrow points from the top of the file to the `db` section, and an orange arrow points from the `auth` section to the top of the file. A blue arrow points to the `exceptions` array at the bottom.

```
{
  "server": {
    "protocol": "http",
    "host": "localhost",
    "port": 3000
  },
  "gui": {
    "protocol": "http",
    "host": "localhost",
    "port": 8080
  },
  "version": "v1",
  "db": {
    "host": "localhost",
    "user": "codenation",
    "password": "codenation",
    "database": "logcentral_#database_environment#",
    "port": 3306,
    "logging": false,
    "dialect": "mysql",
    "define": {
      "underscored": true,
      "timestamps": true,
      "paranoid": false
    }
  },
  "auth": {
    "key": "x-auth-token",
    "secret": "terces-secret",
    "options": {
      "issuer": "api.logcentral",
      "expiresIn": "7d",
      "algorithm": "RS512"
    }
  },
  "exceptions": [
    { "method": "GET", "url": "/" },
    { "method": "GET", "url": "/#version#" },
    { "method": "GET", "url": "/Swagger", "children": true }
  ]
}
```

As exceções para *endpoints*, citadas anteriormente, também são definidas neste arquivo.

Model

As entidades são definidas a partir de uma classe modelo/genérica que implementa métodos comumente utilizados em operações básicas.

JS model.js ×

api > src > utils > JS model.js > ...

```
1  const Sequelize = require('sequelize')
2
3  class Model extends Sequelize.Model {
4    static findById(id, attributes = undefined, include = undefined) {
5      return this.findOne({
6        where: { id },
7        attributes,
8        include
9      })
10   }
11
12   static updateById(id, body) {
13     return this.update(body, {
14       where: { id }
15     })
16   }
17
18   static destroyById(id) {
19     return this.destroy({
20       where: { id }
21     })
22   }
23 }
24
25 module.exports = Model
```

Assim, cada entidade iniciada pode usufruir de tais métodos por herança, evitando a repetição de códigos e reduzindo o tempo despendido para execução de implementações e até mesmo de testes.

JS log.js ×

api > src > models > JS log.js > Log > init

```
1  const { DataTypes } = require('sequelize')
2  const Model = require('../utils/model')
3
4  class Log extends Model {
5    static init(sequelize) {
6      return super.init({
7        url: DataTypes.STRING,
8        message: {
9          type: DataTypes.STRING,
10         allowNull: false,
11         validate: {
12           notEmpty: true
13         }
14       },
15       data: DataTypes.JSON,
16       datetime: {
17         type: DataTypes.DATE,
18         allowNull: false,
19         defaultValue: DataTypes.NOW
20       },
21       sourceAddress: {
```

Service

Para evitar a necessidade de implementação de métodos comuns para cada uma das entidades representadas neste formato descrito no item anterior, optamos pela implementação de uma classe genérica para a camada de serviço.

Para entidades cujas regras de negócio necessitem apenas de operações *CRUD* padrão, o service será criado/estendido em tempo de execução.

```
JS service.js X
api > src > utils > JS service.js > ...
1  const Log = require('../services/log')
2  const models = require('../models')
3
4  class Service {
5    constructor(modelFile, attributes = undefined, include = undefined) {
6      this.model = require('../models/' + modelFile)
7      this.attributes = attributes
8      this.include = include
9    }
10
11    getById(request, response, next) {
12      try {
13        this.model.findById(request.params.id, this.attributes, this.include)
14          .then(data => {
15            if (data) {
16              response.json(data)
17            } else {
18              Log.send(request, response, next, null, 404)
19            }
20          })
21          .catch(error => Log.send(request, response, next, error))
22      } catch (error) {
23        Log.send(request, response, next, error)
24      }
25    }
26
27    create(request, response, next) {
28      try {
29        this.model.create(this.bodyFormat(request.body))
30          .then(newData => {
31            const attributes = !this.include ? [ ...Object.keys(newData._changed),
32
33              this.model.findById(newData.id, attributes, this.include)
34                .then(data => response.status(201).json(data))
35                .catch(error => Log.send(request, response, next, error))
36          })
37      }
38    }
39  }
```

Esta camada será responsável pelas diversas integrações entre os *controllers* (onde serão definidas as rotas) e o banco de dados – sempre através do ORM *Sequelize*.

A depender das regras de negócio, dada classe de serviço poderá herdar os métodos padrões da classe básica, conter novos métodos e/ou até mesmo subscrever métodos existentes.

Controller

Também para dispensar a necessidade de reimplementação dos mesmos métodos associados à cada entidade, um *controller* básico contém as rotas mais comuns.

```
JS log-service.js X
api > src > services > log > JS log-service.js > Config
1  const Log = require('.')
2  const Service = require('../utils/service')
3  const models = require('../models')
4  const Config = require('../config')
5
6  class LogService extends Service {
7    archive(request, response, next) {
8      this.model
9        .updateById(request.params.id, { archived: true })
10       .then(data => {
11         if (data[0] > 0) {
12           response.status(204).send()
13         } else {
14           Log.send(request, response, next, null, 404)
15         }
16       })
17       .catch(error => Log.send(request, response, next, error, 404))
18     }
19
20     create(request, response, next) {
21       models.User
22         .findOne({ where: { token: request.headers[Config.settings.auth.key] }, attributes: [ 'id' ] })
23         .then(user => {
24           if (user) {
25             request.body.user = { id: user.id }
26
27             super.create(request, response, next)
28           } else {
29             Log.send(request, response, next, null, 401)
30           }
31         })
32         .catch(error => Log.send(request, response, next, error))
33     }
34   }
35
36   module.exports = new LogService('log', undefined, include)
37
```

```
JS controller.js X
api > src > utils > JS controller.js > ...
1  const express = require('express')
2  const Service = require('./service')
3
4  class Controller {
5    constructor(serviceOrModelFile) {
6      this.router = express.Router()
7      this.service = typeof serviceOrModelFile === 'object' ? serviceOrModelFile : new Service(serviceOrModelFile)
8    }
9
10   setDefaultRoutes() {
11     this.router.get('/', (request, response, next) => this.service.getAll(request, response, next))
12
13     this.router.get('/:id', (request, response, next) => this.service.getById(request, response, next))
14
15     this.router.post('/',
16       (request, response, next) => this.service.validator(request, response, next),
17       (request, response, next) => this.service.create(request, response, next))
18
19     this.router.patch('/:id',
20       (request, response, next) => this.service.validator(request, response, next),
21       (request, response, next) => this.service.update(request, response, next))
22
23     this.router.delete('/:id', (request, response, next) => this.service.delete(request, response, next))
24   }
25 }
26
27 module.exports = Controller
--
```

Para entidades cujas regras de negócio necessitem apenas de operações CRUD padrão, o *controller* será criado/estendido em tempo de execução.

Já para entidades que necessitem de mais operações, o código implementado estende o *controller* padrão e implementa apenas as demais rotas.



```
JS log.js x
api > src > controllers > JS log.js > ...
1  const Controller = require('../utils/controller')
2  const LogService = require('../services/log/log-service')
3
4  class LogController extends Controller {
5    constructor() {
6      super(LogService)
7
8      this.router.put('/:id/archive',
9        (request, response, next) => this.service.validator(request, response, next),
10       (request, response, next) => this.service.archive(request, response, next))
11
12     this.service.update = this.service.notAllowed
13   }
14 }
15
16 module.exports = new LogController()
17
```

Conforme exemplo acima, rotas padrões indesejadas são evitadas ao atribuir um retorno padrão de status 405 (não permitido).

Dificuldades

Tempo escasso dos integrantes

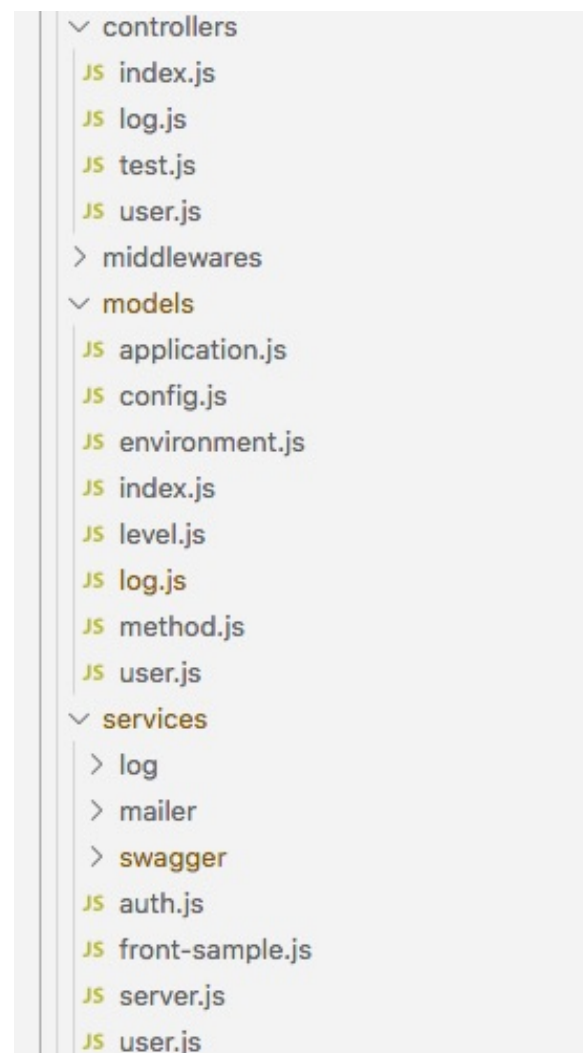
Devido à ausência de tempo, ocasionada por mudanças de rumo de carreiras e vida pessoal de alguns integrantes – incluindo a desistência de uma deles, ainda que por motivos extremamente justificáveis –, tivemos dificuldade em nos organizar para cumprir todos os objetivos.

Como a divisão das tarefas tornou-se um ponto crítico a medida que as agendas profissionais de alguns membros foi ficando reduzida, fomos obrigados a deixar de lado uma das funcionalidades que, embora não explicitada no requisito, julgávamos ser um diferencial para o projeto.

Mudança de programação

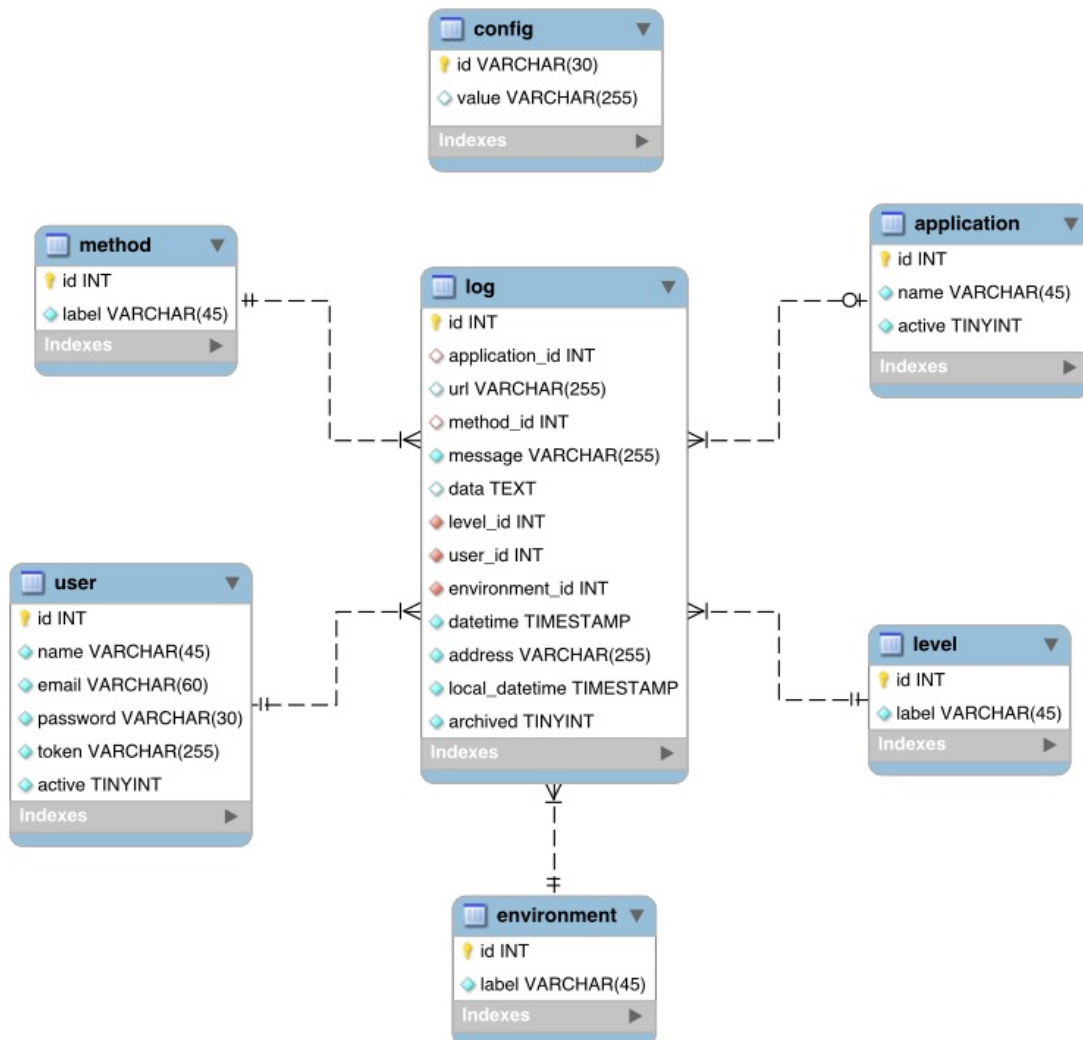
Duas aulas presenciais que seriam exclusivamente dedicadas ao projeto final foram, por motivos de força maior, convertidas para o formato online, então com conteúdo aplicado.

Ficou comprometido o principal alicerce de nosso planejamento, que seriam dois *sprints* presenciais para codificação da aplicação já idealizada.



Assertividade

Acreditamos que produzir o modelo de dados e a documentação – utilizando *Swagger* – antes de iniciar a codificação das funcionalidades tenha sido nosso maior acerto.



Tais definições foram essenciais para reduzir a incidência de dúvidas sobre o que seria necessário implementar para cada item listado no escopo – principalmente devido à necessidade de identificar requisitos pouco detalhados, a partir de *screenshots* acrescentados ao resumo do projeto.

Após identificadas as necessidades, a disposição de todos para discutir as possíveis soluções durante pequenos intervalos das aulas presenciais também foi essencial para o resultado apresentado.

`https://api.logcentral.azul.dev`