Documentação - Projeto Final

Version 1.0

http://deploy-backend-on11-barbara.herokuapp.com/

PANORAMA SLOW

Pesquisador de marcas e produtores que lutam por práticas socioambientais

Descrição da API

> Seguindo inspiração do movimento Fashion Revolution (movimento em prol uma moda mais justa e transparente), o intuito desse projeto é cadastrar marcas de acordo com cada região para facilitar a pesquisa de pessoas que estão em busca de consumir de forma mais responsável e consciente por micro e pequenas empresas locais e produtores artesanais que lutem por causa socioambiental. A pesquisa pode ser feita por estado, cidade e nome da marca ou produtor que ajudará na hora para uma pesquisa mais precisa. É uma forma de apoio ao comércio local e incentivar pessoas a entender os impactos de suas roupas/produtos e como elas podem influenciar a industria global da moda ao procurar por empresas que se preocupam com o impactos negativos gerados e mostrando sua transparência como empresa, como todas deveriam ser.

▼ "Inspirar pessoas a consumir menos, valorizar a qualidade e cuidar melhor de suas roupas/produtos." - Manifesto Fashion Revolution

Implementações futuras

Empresas que estejam ligadas também à agenda da ONU 2030 constando até 1 dos 17 Objetivos de Desenvolvimento Sustentável também poderá fazer o cadastro e ser divulgada. Cada marca terá uma especificação por filtro de causa de forma mais geral. Elas poderão está atualizando sua página de transparência sempre que tiverem uma atualização para fazer. Constará também um filtro apenas para marcas feitas por pessoas negras e LGBTQIA+.

- Pesquisa por marcas que impactam o bem;
- Campo para transparência de marcas/produtos contribuintes;
- É uma marca que deseja compartilhar dados? Cadastre-se aqui;
- Atualizar página de transparência;
- Filtragem de marca/produtor feita/idealizada por pessoa negra;
- Filtragem de marca/produtor feita/idealizada por LGBTQIA+;
- Filtragem de marca/produtor por cidade e estado;
- Vídeos contando a história e práticas por trás de cada;
- Finalizar autenticação de usuários e limitar controle de conteúdo;

EndPoints / Métodos:

Model:

```
{
"id": 1,
"data criação": NUMBER,
"nome marca": "STRING",
"descricao": "STRING",
"motivacao": "(?)",
"causas que cumpre atualmente": "STRING",
"estado": "STRING",
"cidade": "STRING",
"é uma marca feita por pessoa negra?": BOLLEAN,
"fundada em": NUMBER,
"classificação": "STRING",
"contatos": "STRING"
}
```

Autenticação do usuário.

```
const autentication = (req, res, next) => {
  const authorization = req.get("authorization")
  if (authorization == null) {
    res.status(401).json({ message: "Acesso negado." })
  }
  let token = authorization.split(" ")[1];
  jwt.verify(token, process.env.JWT_SECRET, (error, data) => {
    req.usuarioAutenticado = data;
    if (error) {
      return res.status(400).json({ message: "Não autorizado ou token inválido." });
    }
    return next();
  })
}
```

[GET] "/"

Retornar toda a apresentação do projeto.

```
router.get("/", function (req, res) {
    res.status(200).send({
        title: "Projeto Final Reprograma - On11 - Batizado de Panorama Slow, inspirada no movimento Fashion Revolution, com essa API s
        version: "1.0.0"
    })
})
```

[GET] "/marcas/"

Visualizador de todas as Marcas cadastradas.

```
const getAll = async (req, res) => {
    try {
        const marcasAll = await Marcas.find()
        return res.status(200).json(marcasAll)
    } catch (err) {
        return res.status(500).json({ message: err.message })
    }
}
```

[GET] "/marcas/id"

Pesquisa cada Marca por ID.

```
const getById = async (req, res) => {
    try {
        const marcasId = await Marcas.findById(req.params.id)
        if (marcasId == null) {
            return res.status(404).json({ message: 'Marca não encontrada!' })
        }
        return res.json(marcasId)
    } catch (err) {
        return res.status(500).json({ message: error.message })
    }
}
```

[POST] "/marcas/cadastrar"

Cadastrar marcas que deverá ser enviada pelo body.

```
const criaMarcas = async (req, res) => {
    const novaMarca = new Marcas({
        _id: new mongoose.Types.ObjectId(),
        nomeMarca: req.body.nomeMarca,
        descricao: req.body.descricao,
        motivacao: req.body.motivacao,
        causasQueCumpreAtualmente: req.body.causasQueCumpreAtualmente,
        cidade: req.body.cidade,
        estado: req.body.estado,
        marcaFeitaPorPessoaNegra: req.body.marcaFeitaPorPessoaNegra,
        fundadaEm: req.body.fundadaEm,
        classificacao: req.body.classificacao,
        contato: req.body.contato,
        cnpj: req.body.cnpj
    })
    const marcaJaExiste = await Marcas.findOne({ nomeMarca: req.body.nomeMarca })

    if (marcaJaExiste) {
        return res.status(400).json({ error: 'Marca já cadastrada!' })
    }
    try {
        const marcaSalva = await novaMarca.save()
        res.status(201).json(marcaSalva)
    } catch (err) {
        res.status(500).json({ message: err.message })
    }
}
```

[PATCH] "/marcas/:id"

Atualizar marca por ID.

```
const atualizaMarca = async (req, res) => {
    try {
        const marcasAtualizar = await Marcas.findById(req.params.id)
        if (marcasAtualizar == null) {
            return res.status(404).json({ message: 'Marca não encontrada!' })
        }

        if (req.body.dataCriacao != null) {
            filme.dataCriacao = req.body.dataCriacao
        }
        if (req.body.nomeMarca != null) {
            filme.nomeMarca = req.body.nomeMarca
        }
        if (req.body.descricao != null) {
            filme.descricao = req.body.descricao
        }
        if (req.body.motivacao != null) {
            filme.motivacao = req.body.motivacao
        }
        if (req.body.causasQueCumpreAtualmente != null) {
            filme.causasQueCumpreAtualmente = req.body.causasQueCumpreAtualmente
        }
        if (req.body.cidade != null) {
            filme.cidade = req.body.cidade
        }
        if (req.body.estado != null) {
            filme.estado = req.body.estado
        }
        if (req.body.fundadaEm != null) {
            filme.fundadaEm = req.body.fundadaEm
        }
        if (req.body.classificacao != null) {
            filme.classificacao = req.body.classificacao
        }
        if (req.body.contatos != null) {
            filme.contatos = req.body.contatos
        }
        if (req.body.cnpj != null) {
            filme.cnpj = req.body.cnpj
        }

        const marcasAtualizadas = await Marcas.save()
        res.json(marcasAtualizadas)

    } catch (error) {
        res.status(500).json({ message: error.message })
```

```
    }
}
```

[DELETE] "/marcas/:id"

Deletar marcas por ID.

```
const deleteMarca = async (req, res) => {
    try {
        const marcasDeletar = await Marcas.findById(req.params.id)
        if (marcasDeletar == null) {
            return res.status(404).json({ message: 'Marca não encontrada!' })
        }
        await marcasDeletar.remove()
        res.json({ message: 'Marca deletada com sucesso!' })
    } catch (error) {
        return res.status(500).json({ message: error.message })
    }
}
```

[GET] "/login/"

Retornar todos os usuários para análise do administrador.

```
const getAll = async (req, res) => {

  const todosUsuarios = await Login.find()
  return res.status(200).json(todosUsuarios)
};
```

[POST] "/login/"

Login do usuário.

```
const loginUsuario = async (req, res) => {

  const usuario = await Login.findOne({ email: req.body.email })

  if (usuario == null) {
    return res.status(400).json({ message: "Ops! Não encontramos o usuário ou senha. Verifique se está correto." })
  }
  let user
  if (bcrypt.compareSync(req.body.password, usuario.password)) {
    user = {
      name: usuario.name,
      email: usuario.email
    }
    jwt.sign(user, process.env.JWT_SECRET, { expiresIn: '7d' }, (error, token) => {
      if (error) {
        res.status(500).json({ message: error.message })
      }
      res.status(201).json({ token: token, data: user })
    })
  }
  else {
    res.status(400).json({ message: "Ops! Não encontramos o usuário ou senha. Verifique se está correto." })
  }
}
```

[POST] "/login/registrar"

Cadastrar o usuário.

```
const registroLogin = async (req, res) => {
  const novoRegistro = new Login({
    _id: new mongoose.Types.ObjectId(),
    name: req.body.name,
    email: req.body.email,
    password: req.body.password
  })
  novoRegistro.password = hashPassword(req.body.password)

  const emailJaExiste = await Login.findOne({ email: req.body.email })
```

```
  if (emailJaExiste) {
    return res.status(409).json({ error: 'Email já cadastrado!' })
  }
  else {
    const registroSalvo = await novoRegistro.save()
    res.status(201).json(registroSalvo)
  }
}
```

[PUT] "/login/substitui"

Mudar senha do usuário.

```
const substituiPassword = async (req, res) => {
  try {
    const usuario = await Login.find()
    if (usuario == null) {
      return res.status(401).json({ message: "Não foi possível encontrar usuário." })
    }
    usuario.password = hashPassword(req.body.password)
    usuario.save()
    return res.status(200).json({ message: "Senha atualizada com sucesso!" })
  } catch (error) {
    return res.status(500).json({ message: error.message })
  }
}
```

[DELETE] "/login/:id"

Deletar usuário por ID.

```
const delById = async (req, res) => {
  try {
    const usuario = await Login.findByIdAndDelete(req.params.id)
    if (usuario == null) {
      return res.status(304).json({ message: "Sem modificações!" })
    }
    return res.status(200).json({ message: "Usuário deletado com sucesso!" })
  } catch (error) {
    return res.status(500).json({ message: error.message })
  }
}
```