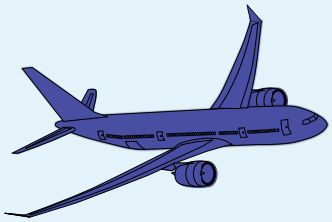
SOLUÇÃO AGNÓSTICA

Documentação para explicação e how-to da solução agnóstica apresentada.

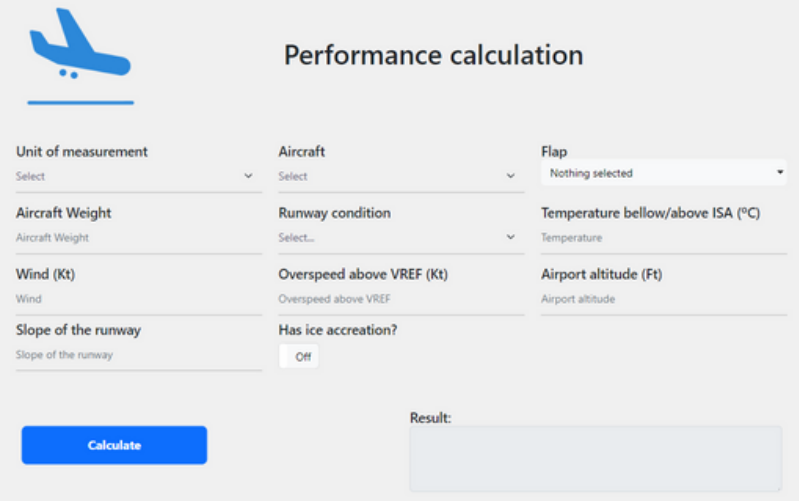


CONTEXTO:

O projeto apresenta uma interface para que os pilotos de aeronaves possam, a partir dos parâmetros considerados, calcular a distância necessária para realizar o pouso de forma segura. Para tal cálculo, uma tabela foi disponibilizada para que todos os fatores sejam levados em consideração. Tais fatores são: modelo, motor, certificação, peso e flap de pouso da aeronave, altitude do aeroporto, temperatura, vento, inclinação da pista, uso de reversor, aditivo de velocidade, acúmulo de gelo, condição de pista e nível de aplicação de frenagem.

MEMBROS DO GRUPO:

Ana Carolina
Davi Elias
Diego Batista
Gustavo Lobato
Jeniffer Cristina
Mateus Henrique
Thales Kerber



Página de cálculo de distância de pouso

Nesta página, o usuário deve informar os valores referentes as condições momentâneas/locais dos parâmetros estabelecidos. Dessa forma, para cada tipo de modelo de aeronave e flap selecionado, já terá disponibilizado todos os valores das respectivas tabelas cadastradas.

SOLUÇÃO AGNÓSTICA

Considerando-se o fundamento de que qualquer tipo de aeronave pode ser cadastrada, além dos seus respectivos parâmetros associados, criou-se uma solução agnóstica que comporta todas essas necessidades. Assim, para implementar tal solução, é necessário cadastrar a aeronave com todos os dados tabelados, como por exemplo, reference, weight, altitude, temperature, wind, slope, overspeed, reversor. Em todos os casos têm as opções de input "com e sem gelo na pista". Além disso, a solução agnóstica se aplica a casos onde não tem disponibilizado o parâmetro nas opções de cadastro e calculo, sendo necessário criá-lo no back e front do projeto.

PROCEDIMENTO PARA CRIAÇÃO DE UM NOVO PARÂMETRO:

Primeira etapa:

1. Inicialmente, deve-se abrir o projeto na pasta denominda "server", em seguida a pasta "src" e a pasta "models", por fim, é necessário abrir o arquivo "operationDistance.ts".

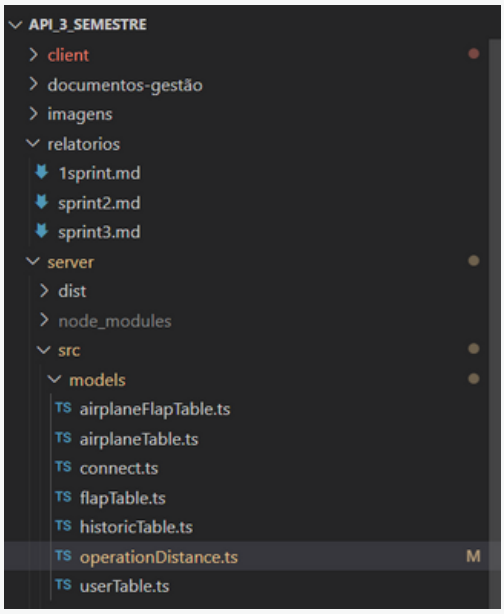
2. Ao acessar o arquivo "operationDistance.ts", basta criar um novo atributo no banco de dados dentro do objeto "const operationDistance" associado ao parâmetro requisitado .

3. O atributo deve seguir o formato:

```
nomeNovoParametro:{
  type: Sequelize.INTEGER,
  allowNull: false
}
```

Obs: se dentro do parâmetro houver outras variáveis, é necessário criar um novo atributo para cada tipo de situação. Ex: novoParametroComGelo, novoParametroSemGelo.

PROCESSOS DA ETAPA I




```
novoParametroComGelo: {
    type: Sequelize.INTEGER,
    allowNull: false
},
novoParametroSemGelo: {
    type: Sequelize.INTEGER,
    allowNull: false
}
```

```
//operationDistance.sync({ alter: true });

module.exports = operationDistance;
```

Após introduzir o novo parâmetro como atributo, deve-se descomentar o "//operationDistance.sync({ alter: true });" atualizar o projeto e comentá-lo novamente para não dar conflito no projeto.

```
export default class NovoParametro extends FatorCalculo{

    private aircraft: Aircraft;
    private table: Table;
    constructor(aircraft: Aircraft,temGelo: boolean, table: Table){
        super();
        this.temGelo = temGelo;
        this.aircraft = aircraft;
        this.table = table;
    }
    public converterSistema(unitMeasurement: UnitMeasurement): void {

    }

    public calcular(): number {
        if(this.temGelo){
            return this.table.novoParametroComGelo

        }
        return 0
    }
}
```

Segunda etapa - Parte I

1. Inicialmente, deve-se abrir o projeto na pasta denominda "client", em seguida a pasta "src" e a pasta "models", por fim, é necessário criar uma classe referente ao novo parâmetro que se deseja adicionar.

2. Após criar a classe com o método calcular, é necessário inicializá-lo na classe "Calcular" que é onde será considerado esse novo parâmetro no cálculo.

Segunda etapa - Parte 2

1. Inicialmente, deve-se abrir o projeto na pasta denominda "client", em seguida a pasta "src" e a pasta "models", por fim, é necessário abir o arquivo "table.ts".

```
export default class Table {

    public novoParametroComGelo: number
    //novo parâmetro com gelo
```

2. Dentro da classe Table, deve-se criar o atributo com o nome do novo parâmetro, indicando seu tipo.

3.Em seguida, deve-se adicionar no construtor o novo parâmetro (atributo).

```
constructor(
    novoParametrocomGelo: number,
) {
    this.novoParametroComGelo = novoParametrocomGelo
```

```
}
set setNovoParametroComGelo(value:number){
    this.novoParametroComGelo = value
}
```

4. Por fim, é necessário passar o novo parâmetro nos métodos set e get.

```
get getNovoParametroComGelo(): number {return this.novoParametroComGelo}
```

Terceira Etapa

1. Inicialmente, deve-se abrir o projeto na pasta denominda "client", em seguida a pasta "src" e a pasta "views", por fim, é necessário abir o arquivo "CadastroFlap.tsx.".

```
type state = {
    novoParametroComGeloError: string,
```

2. Ao acessar o arquivo "CadastroFlap.tsx.", deve-se adicionar dentro do "type state ={" o novo parâmetro com a adição da palavra Error no nome e definir o tipo como string.

3. Em sequência, deve-se adicionar o parâmetro dentro da estrutura principal, abaixo do "this.state = {" o novo parâmetro com " " e, em seguida, adicionar dentro do parenteses do new Table : 0 se for do tipo número, ou aspas se for do tipo string.

```
private table: Table = new Table(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '');
constructor(props: any) {
    super(props);
    this.state = {
        novoParametroComGeloError: '',
        altitudeReferenceError: '',
        altitudeWithIceError: '',
```

4. O próximo passo se resumo em adicionar o parâmetro com a palavra Change a sua frente e igualar o novo parâmetro com change ao método bind.

```
}
this.novoParametroComGeloChange = this.novoParametroComGeloChange.bind(this);
```

5. Nesse passo, torna-se necessário criar um método com o parâmetro event, seguindo toda estrutura já estabelecida.

```
novoParametroComGeloChange(event) {
    let novoParametroComGeloError;
    const target = event.target;
    this.table.novoParametroComGelo = target.value;
    if (!this.table.novoParametroComGelo) {
        novoParametroComGeloError = "O novo parâmetro com gelo é obrigatório";
    } else {
        novoParametroComGeloError = ""
    }
    this.setState({ novoParametroComGeloError: novoParametroComGeloError })
}
```

```
validate = () => {
    let novoParametroComGeloError = "";
```

```
if (!this.table.novoParametroComGelo) {
    novoParametroComGeloError = "O novo parâmetro com gelo é obrigatório";
} else {
    novoParametroComGeloError = ""
}
```

6. Nesse momento, deve-se criar uma variável do novo parâmetro dentro da função validate , sendo igual a " ". Além disso, é necessário aplicar abaixo as condições envolvendo o if e else.

7. Em seguida, no método state deve-se adicionar o nome do novo parâmetro com o Error a sua frente, e adicioná-lo na condição do if também.

```
this.setState({
    novoParametroComGeloError: novoParametroComGeloError,
})
if (novoParametroComGeloError
    || refWithIceError
```

Terceira Etapa

```
postClickButton = async (event: any) => {
    event.preventDefault();
    const isValid = this.validate();
    if (isValid) {
        let res = -1
        await axios.post("http://localhost:3001/flap/cadastrar", {

            tipoFlap: this.table.flap
            // airplaneId: res
        }).then((response) => {
            res = response.data.id
            //axios
            axios.post("http://localhost:3001/operationDistance/cadastrar", {
                id: res,
                novoParametroComGelo: this.table.novoParametroComGelo,
```

8. Nessa etapa, é necessário adicionar o novo parâmetro na rota de cadastro(post), seguindo a sequência demonstrada.

9. Por fim, é necessário criar uma coluna e linha para o novo parâmetro ser demonstrado no front, seguindo a ordem demonstrada.

```
<Row>
    <Col>
        <h5 className="card-title">Novo Parâmetro com Gelo (Unidade de Medida)</h5>
        <input type='number'
        className='input form-control form-control-lg inputGroup-sizing-sm'
        id="novoParametroComGelo"
        placeholder="Novo Parâmetro com Gelo"
        onChange={this.novoParametroComGeloChange} />
        <div style={{ fontSize: 12, color: "red" }}>
            {this.state.novoParametroComGeloError}
        </div>
    </Col>
    <Col></Col>

</Row>
```

Considerações Finais:

De acordo com toda demonstração apresentada, a solução agnóstica possibilita adicionar novos parâmetros que comporte propriedades específicas de cada aeronave. É válido ressaltar, que para tal aplicação, é necessário seguir as etapas de adição tanto no back (banco de dados do projeto) quanto no front(interface do projeto).