DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

isep Instituto Superior de Engenharia do Porto

Relatório ESINF SPRINT2

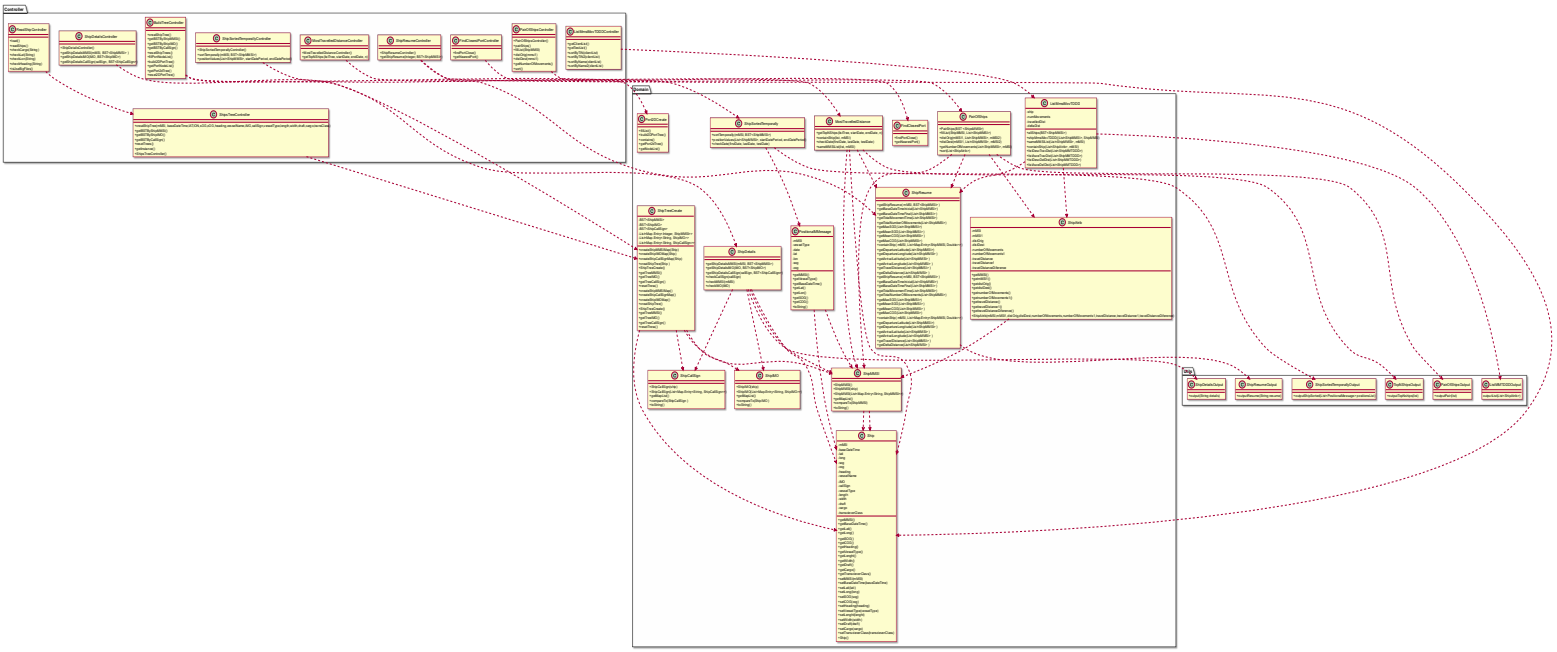
Turma 2DK _ Grupo 101

1190963_ Pedro Preto
1200963_Rui Dias
1201615_Maria Marques
1201845_Marta Ferreira

Professora
Adelaide Sampaio (AIS)
Unidade Curricular
ESINF

02/12/2021

Class Diagram



Observando o ‘class diagram’ apresentado acima é possível concluir que as novas classes criadas, com base nos requisitos das US do sprint 2, pertencentes tanto ao package do ‘Controller’ como do ‘Domain’ têm como principal função:

- Port2DTreeCreate2 que tem como principal objetivo a criação de uma kd tree balanceada, em que vai inserindo ou atualizando valores. O controller criado para esta classe foi BuildTreeController que chama o método que cria a árvore e que a preenche com os portos.

- FindClosestPort que tem como objetivo procurar o porto mais próximo através da pesquisa do vizinho mais próximo através de um dado CallSign e numa certa data. O controller criado de apoio a esta classe foi o FindClosestPortController que irá invocar os métodos necessários a concretização da US.

Análise Complexidade das Funcionalidades:

- **[US201]:** As a Port manager, I which to import ports from a text file and create a 2D-tree with port locations.
    - Acceptance criteria [ESINF]:
        - 2D-tree balanced.

Class BuildTreeController

Class Port2DTreeCreate

```
public class Port2DTreeCreate {

    private KdTree<Port> port2dTree;
    public List<Node<Port>> nodeList;

    public Port2DTreeCreate() {
        nodeList = new LinkedList<>();
        port2dTree = new KdTree<>();
    }

    public void fillList(Port port) {
        Node<Port> node = new Node<>(port, port.getLat(), port.getLon());
        nodeList.add(node);
    }
```

Método fillList (Port port) – O(n), adiciona um elemento numa lista.

```
public void build2DPortTree() {
    port2dTree.buildTree(getNodeList());
    if (port2dTree.getAll().size() != getNodeList().size()) {
        List<Node<Port>> list = new LinkedList<>();
        for (Node<Port> n : getNodeList()) {
            if (!contains(n.getObject().getCode(),port2dTree.getAll())){
                list.add(n);
            }
        }
        for (Node<Port> n :list) {
            port2dTree.insertOrUpdate(n.getObject(),n.getObject().getLat(),n.getObject().getLon());
        }
    }
}
```

Método build2DPortTree ( ) – O(n logn), visto verificar se contém já o elemento a adicionar na árvore e, caso ainda não contenha, faz a inserção do novo elemento numa árvore k-d balanceada .

```
public boolean contains(int code, List<Port> list) {
    for (Port p : list) {
        if (p.getCode() == code) {
            return true;
        }
    }
    return false;
}
```

Método contains (int code, List<Port> list) – O(n), visto ser um método que verifica se a lista já contém o elemento.

```
    public KdTree<Port> getPort2dTree() { return port2dTree; }

    public void reset2DPortTree() {
        port2dTree = new KdTree<>();
        nodeList = new LinkedList<>();
    }

    public List<Node<Port>> getNodeList() { return nodeList; }
}
```

Método getPort2dTree( ) – O(1) por ser um get.

Método reset2DPortTree( ) – $O(n\log^2 n)$ no pior dos casos

Método getNodeList( ) – O(n)

- **[US202]:** As a Traffic manager, I which to find the closest port of a ship given its CallSign, on a certain DateTime.
    - Acceptance criteria [ESINF]:
        - using 2D-tree to find closest port.

Class FindClosestPortController

Class FindClosestPort

```
public Port getNearestPort(KdTree<Port> ports, List<Map.Entry<String, ShipCallSign>> list, String shipCallSign, Date baseDateTime) {
    for (Map.Entry<String, ShipCallSign> l : list) {
        if (l.getKey().equals(shipCallSign)) {
            if (l.getValue().getBaseDateTime().equals(baseDateTime)) {
                return ports.findNearestNeighbour(l.getValue().getlAT(), l.getValue().getlON());
            }
        }
    }
    return null;
}
```

Método getNearestPort(KdTree<Port> ports, List<Map.Entry<String, ShipCallSign>>list, String shipCallSign, Date baseDateTime)- O(logn), no melhor caso e O(n), no pior caso, visto chamar o método de pesquisa do vizinho mais próximo.

```
public Port findPortClose(KdTree<Port> ports, String shipCallSign, BST<ShipCallSign> treeCallSign, Date baseDateTime) {
    return getNearestPort(ports, treeCallSign.find(new ShipCallSign(shipCallSign)).getMapList(), shipCallSign, baseDateTime);
}
```

Método finndPortClose(KdTree<Port> ports, String shipCallSign, BST <ShipCallSign> treeCallSign, Date baseDateTime)- O(logn), no melhor caso e O(n), no pior caso.