QR-Caching

Grupo:

- **João Moniz (Número: 20220550)**

- **Tomás Salgueiro (Número: 20220589)**

Resumo do Projeto

O projeto QR Caching é uma iniciativa que visa promover o turismo de forma intuitiva e interativa, utilizando códigos QR espalhados por pontos estratégicos de Lisboa. Além disso, tem como objetivo criar uma pseudo rede social, permitindo que os utilizadores partilhem as suas experiências e interajam entre si ao longo dos percursos. O sistema também é projetado para ser usado em eventos, como a WebSummit, oferecendo aos participantes uma forma dinâmica e inovadora de explorar o espaço através da criação de rotas de QR Codes. Através de uma aplicação móvel, os utilizadores podem dar scan aos códigos QR, enquanto o site oferece uma interface para a visualização do mapa, integração de inteligência artificial para otimização de rotas, e gestão de eventos.

Engenharia de Software

A engenharia de software desempenha um papel essencial no desenvolvimento deste projeto, assegurando que o processo seja estruturado, eficiente e fácil de manter.

Funcionalidades:

- A adoção de metodologias ágeis para um desenvolvimento flexível, com interações frequentes e foco no feedback contínuo.

- A criação de diagramas de arquitetura e fluxos de dados que descrevem como os componentes do sistema interagem, o que inclui a base de dados de QR Codes, o sistema de recomendação e o mapa interativo.

Levantamento dos Requisitos

Requisitos Funcionais:

1. Mapa interativo de Lisboa, onde os utilizadores podem visualizar os QR Codes disponíveis.
2. Criação de eventos que podem incluir QR Codes específicos e personalizados para o evento.
3. Recomendações de QR Codes e rotas otimizadas através de IA.
4. Distribuição e otimização de QR Codes em eventos de grande escala, para maximizar a interação com os participantes.

Requisitos Não Funcionais:

1. **Desempenho:** O sistema deve carregar o mapa e os QR Codes rapidamente, mesmo com um grande volume de utilizadores.
2. **Segurança:** Proteção rigorosa dos dados dos utilizadores com encriptação e medidas de prevenção contra ataques.
3. **Escalabilidade:** A arquitetura distribuída deve permitir que o sistema se expanda conforme necessário.
4. **Usabilidade:** A interface deve ser intuitiva e fácil de usar.

Guiões e Personas

Personas:

- **Diogo (Turista)**
  - Idade: 21 anos
  - Motivação: Explorar pontos turísticos culturais em Lisboa de forma eficiente e divertida.
  - Interação: Utiliza a aplicação para otimizar seu tempo e descobrir locais históricos.
- **Tiago (Organizador de Eventos)**
  - Idade: 30 anos
  - Motivação: Gerir interações de códigos QR para participantes do WebSummit.
  - Interação: Utiliza o sistema para configurar rotas de QR Codes que guiem participantes por grandes espaços.

Guiões:

- **Cenário 1:** Experiência do Turista
  - O Diogo visita Lisboa e planeja uma rota antes de sair do seu hotel. Ele segue o caminho sugerido pelo website para passar por todas as

caches em locais históricos na zona que ele planeja visitar naquele dia.

- **Cenário 2:** Configuração de Evento
    - O Tiago configura uma série de QR Codes para o WebSummit em locais importantes, como saídas de emergência, casas de banho e hotspots.

Arquitetura Tecnológica

Componentes:

- **Frontend:** Interface interativa para visualização do mapa e gestão de eventos.
- **Backend:** Processa dados e comunica-se com a base de dados e serviços auxiliares.
- **Base de Dados (MongoDB):** Armazena todos os dados relacionados a códigos QR, eventos e interações de utilizadores.
- **Containers:** Cada componente opera num container Docker separado para modularidade e tolerância a falhas.

Configuração no Ubuntu Server:

- **Sistema Operativo:** Ubuntu Server
- **Orquestração:** Docker Compose para gerir containers do backend, MongoDB e outros serviços.
- **Comunicação:** Rede interna Docker para garantir interações seguras e eficientes entre serviços.

Plano de Testes para Integração do Sistema

Objetivos:

- Validar a comunicação entre o backend e o MongoDB.
- Garantir a consistência de dados entre os componentes.

Cenários de Teste:

- **Acesso à Base de Dados:**
  - Ação: Consultar MongoDB por uma lista de códigos QR.
  - Resultado Esperado: O backend retorna os dados corretamente em até 100ms.
- **Integração Frontend:**
  - Ação: O utilizador seleciona um QR Code no mapa.
  - Resultado Esperado: O sistema exibe os detalhes do código QR em tempo real.
- **Gestão de Eventos:**
  - Ação: Adicionar um novo evento com múltiplos códigos QR.
  - Resultado Esperado: O evento é salvo e visível no sistema.

Componentes de IA:

- **Testes Unitários:** Testar a precisão do algoritmo de pathfinding.
- **Testes de Usabilidade:** Testar se a interface permite ao usuário interagir intuitivamente com os QR Codes.

- **Testes de Performance:** Simular alta carga de códigos QR e medir desempenho.

Segurança Informática

Para a componente de Segurança informática o nosso projeto contém, para além da prevenção contra mongoDB injections, implementação de HTTPS, e o Hash das passwords com Salt random por User.

Para o Hash das passwords utilizamos a biblioteca Node JS Crypto Module, e procedemos da seguinte maneira:

SignUp

- Primeiro criamos um Salt random para o User que está a ser criado;
- Segundo Fazemos o Hash da Password com o Salt;

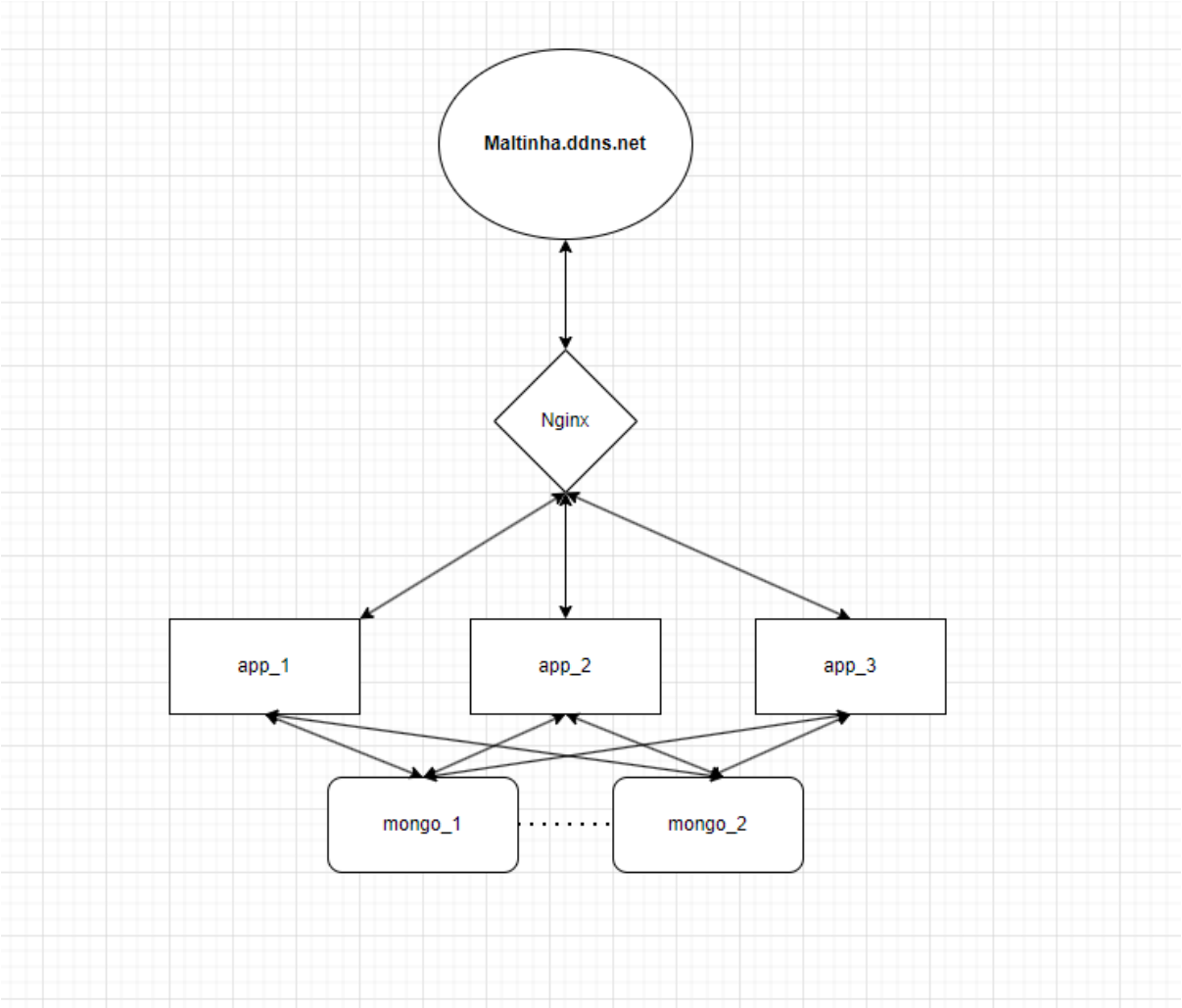
```
// Generate salt and hash the password using HMAC with SHA-256
const salt = crypto.randomBytes(16).toString('hex');
const hashedPassword = crypto.createHmac('sha256', salt).update(user_password).digest('hex');
```

Login

- Primeiro encontramos o Salt do User a dar login;
- Segundo damos Hash à Password com o respectivo Salt;
- Terceiro comparamos o Hash recebido com o Hash guardado na DB;

```
// Retrieve the salt associated with the user
const saltEntry = await Salt.findOne({ user_id: user.user_id });
if (!saltEntry) return res.status(500).json({ message: 'Error, User Not Found' });

// Hash the incoming password with the user's salt
const hashedPassword = crypto.createHmac('sha256', saltEntry.salt).update(user_password).digest('hex');

// Compare the hashed password with the stored password
if (hashedPassword !== user.user_password) {
    return res.status(401).json({ message: 'Invalid credentials' });
}
```

Sistemas Distribuídos



Inteligencia Artificial

Para a componente de Inteligência Artificial decidimos fazer um sistema de Path Finding com A*, de maneira a que num evento quando um User Escolhe um grupo de Qr Codes, é-lhe devolvido a ordem mais eficiente para os apanhar.

Para que isto aconteça temos um file em python que é chamado na página floorplan e que recebe a lista de Qr Codes selecionados e o SVG do evento

```python
app = Flask(__name__)  # Initialize the Flask app

@app.route('/find-path', methods=['POST'])
def find_path():
    data = request.get_json()
    eventSVG = data['eventSVG']
    selected_qr_codes = data['qrCodeDataString']

    # Parse the SVG file
    walls = parse_svg(eventSVG)

    # Call the Python function
    qr_sequence = find_shortest_path(selected_qr_codes, walls)

    # Return the result as JSON
    return jsonify({'qr_sequence': qr_sequence})
```

Com isto depois o programa pega no SVG e retorna em que áreas há paredes

```python
def parse_svg(svg_path):
    tree = ET.parse(svg_path)
    root = tree.getroot()
    walls = []

    for elem in root.iter():
        if elem.tag.endswith('rect'):
            style = elem.attrib.get('style', '')
            if 'fill:#ff0000' in style:  # red walls
                x = float(elem.attrib['x'])
                y = float(elem.attrib['y'])
                width = float(elem.attrib['width'])
                height = float(elem.attrib['height'])
                walls.append(((x, y), (x + width, y + height)))

    return walls
```

Assim que tem as paredes e os Qr Codes o programa corre a função para obter a melhor ordem dos Qr Codes

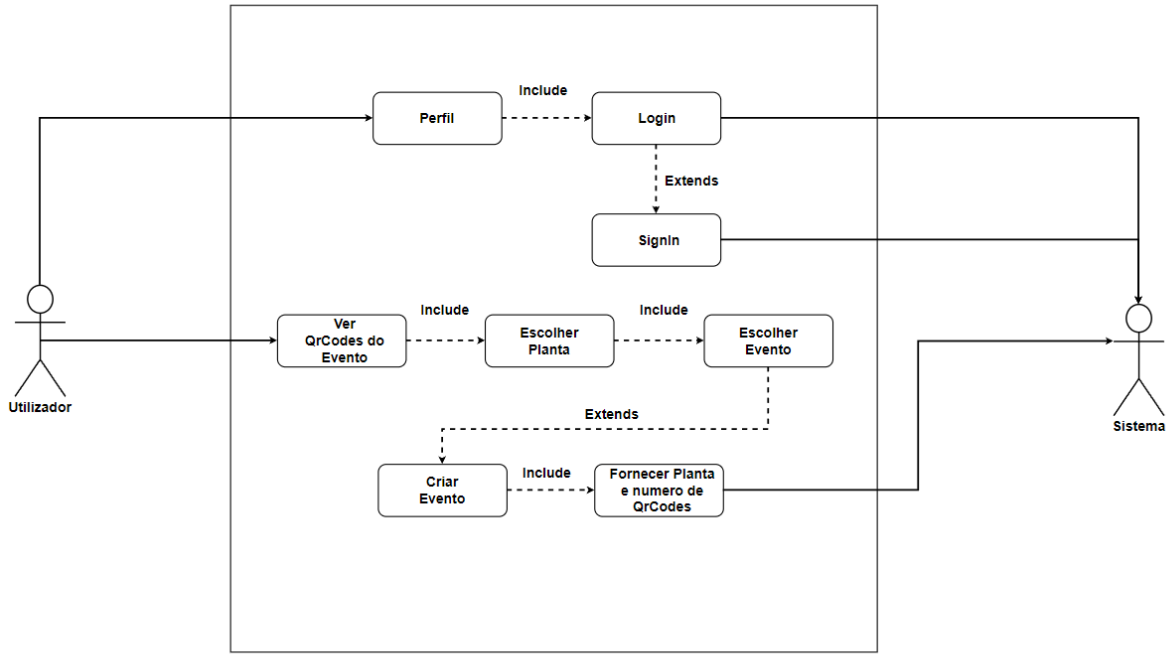
```python
def find_shortest_path(qr_codes, walls):
    def is_collision(point, walls):
        for wall in walls:
            (x1, y1), (x2, y2) = wall
            if x1 <= point[0] <= x2 and y1 <= point[1] <= y2:
                return True
        return False

    def neighbors(node):
        moves = [(0, 1), (1, 0), (0, -1), (-1, 0)]
        result = []
        for dx, dy in moves:
            neighbor = (node[0] + dx, node[1] + dy)
            if not is_collision(neighbor, walls):
                result.append(neighbor)
        return result

    def a_star(start, goal):
        open_set = PriorityQueue()
        open_set.put((0, start))
        came_from = {}
        g_score = {start: 0}
        f_score = {start: distance.euclidean(start, goal)}

        while not open_set.empty():
            _, current = open_set.get()
            if current == goal:

                path = []
                while current in came_from:
                    path.append(current)
                    current = came_from[current]
                path.reverse()
                return path

            for neighbor in neighbors(current):
                tentative_g_score = g_score[current] + distance.euclidean(current, neighbor)
                if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
                    came_from[neighbor] = current
                    g_score[neighbor] = tentative_g_score
                    f_score[neighbor] = tentative_g_score + distance.euclidean(neighbor, goal)
                    open_set.put((f_score[neighbor], neighbor))
        return []

    path = []
    qr_sequence = [qr_codes[0]]
    current = qr_codes[0]
    for next_code in qr_codes[1:]:
        path += a_star(current, next_code)
        qr_sequence.append(next_code)
        current = next_code
    print(qr_sequence)
    return qr_sequence
```

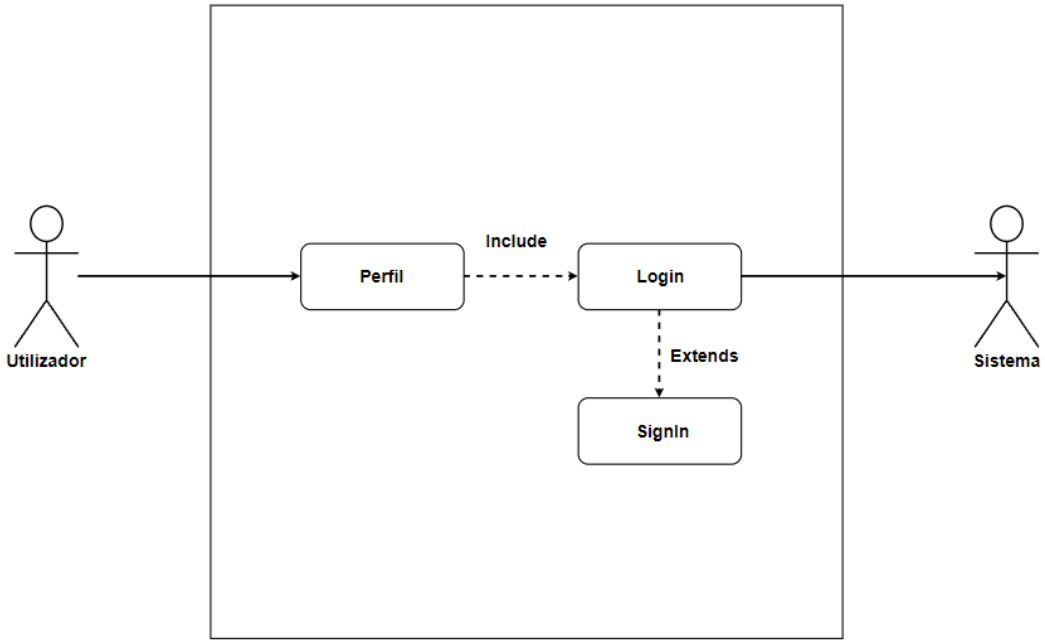
E com isto recebemos o path de Qr Codes no Javascript

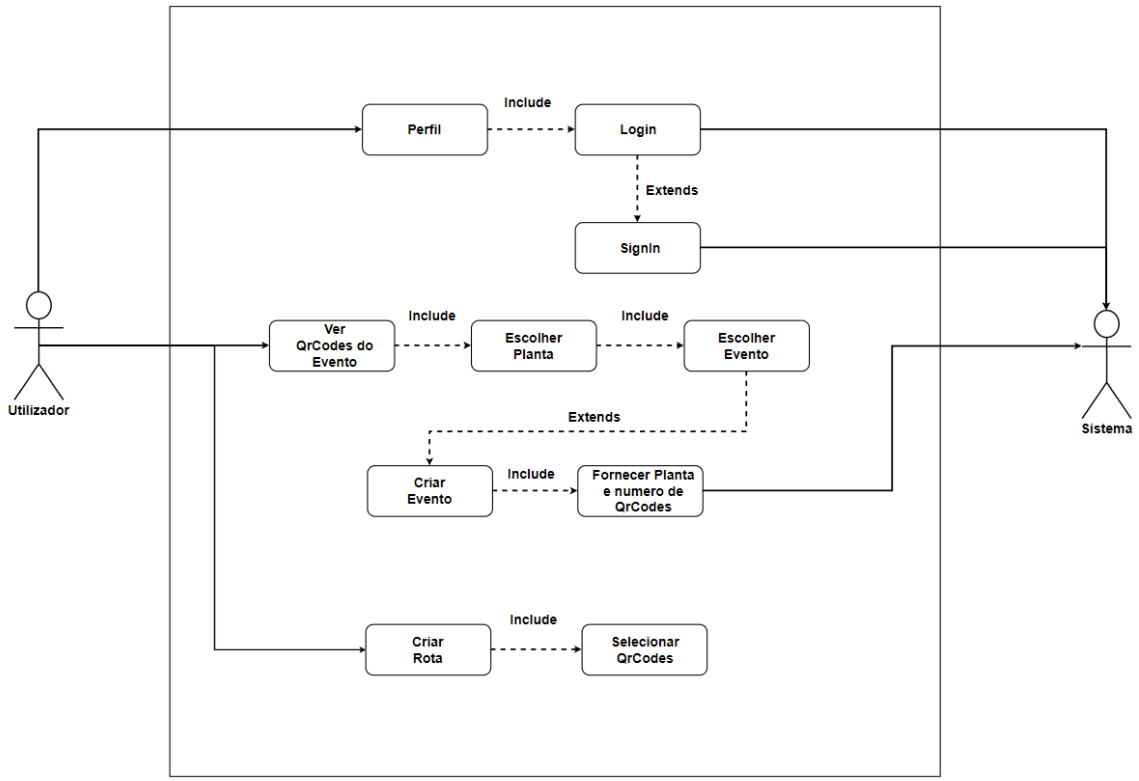
```
    const pathData = { qrCodeDataString, eventSVG };

    const response = await fetch('/find-path', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(pathData),
    });

    if (!response.ok) {
        throw new Error(`Error finding path: ${response.statusText}`);
    }

    const result = await response.json();
    console.log('Received QR sequence:', result.qr_sequence);
} catch (error) {
    console.error('Error:', error);
}

// Display the result
const pathOutput = document.getElementById('path-output');
pathOutput.textContent = `Order of QR codes to collect: ${pathOrder.join(' -> ')}`;
```

Diagramas e Tabelas

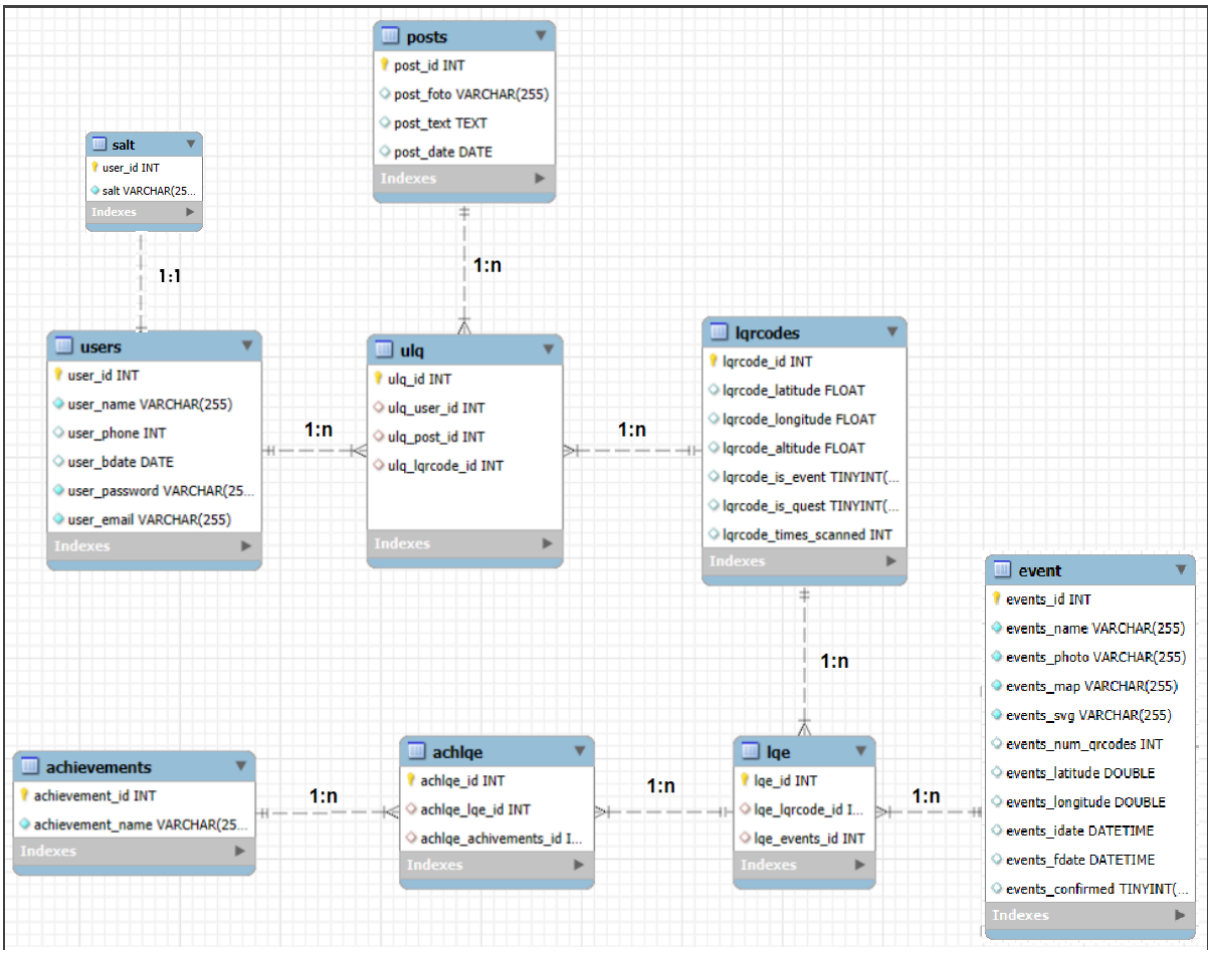
UML




Utilizador
Perfil
Include
Login
Extends
SignIn
Sistema


Utilizador
Perfil
Include
Login
Extends
SignIn
Ver QrCodes do Evento
Include
Escolher Planta
Include
Escolher Evento
Extends
Criar Evento
Include
Fornecer Planta e numero de QrCodes
Include
Criar Rota
Selecionar QrCodes
Sistema

Modelo Relacional



Modelo Scrum

Tarefa	Setembro	Outubro	Novembro	Dezembro	Janeiro
API					
Frontend					
Base de dados					
Replicação (SD)					
PathFinding (AI)					
Segurança					
	Por fazer				
	A Fazer				
	Feito				