Automação de Registro de Frequência Discente no Emile com Bluetooth Low Energy

Trabalho de Conclusão de Curso

Chrystyan Byrnes Santos de Olinda

Sandro Santos Andrade
Orientador

Instituto Federal da Bahia - IFBA
Curso de Análise e Desenvolvimento de Sistemas
Campus Salvador

Salvador, Bahia, Brasil
11 de novembro de 2025

Sumário

1 Visão Geral

No cotidiano de professores e estudantes, o celular tornou-se uma das principais portas de acesso à informação e aos sistemas acadêmicos. Pensando nisso, o Emile [1] foi desenvolvido como um aplicativo multiplataforma, disponível para *Android* e *iOS*, que aproxima o ambiente institucional da rotina real dos usuários. Ele funciona como uma interface móvel mais simples e intuitiva para o Sistema Unificado de Administração Pública (*SUAP*) [2], permitindo que diversas funcionalidades utilizadas no dia a dia acadêmico sejam realizadas de forma mais prática e natural, diretamente pelo *smartphone*.

A comunicação entre o aplicativo e o *SUAP* é realizada por meio de *WebScraping*. Em termos simples, essa técnica funciona como um assistente virtual que acessa o sistema original, lê as informações da tela e as organiza automaticamente dentro do aplicativo, assim possibilitando ao Emile extrair e organizar informações relevantes ao usuário autenticado. Para perfis de Aluno, o sistema apresenta recursos como visualização da matriz curricular, acompanhamento das disciplinas matriculadas, controle de faltas por componente curricular e participação em *quizzes* baseados nas aulas ministradas. Para perfis de Professor, o aplicativo oferece ferramentas para registrar aulas com todos os dados necessários, realizar a chamada da turma e criar *quizzes* de forma rápida a partir das aulas cadastradas. Dessa forma, o Emile busca tornar o acesso às atividades acadêmicas mais eficiente, integrado e alinhado à realidade de uso dos dispositivos móveis.

1.1 Declaração do problema

Atualmente, o professor dispõe de duas formas para realizar o registro de presença dos alunos da instituição. Ele pode utilizar o *SUAP*, onde os estudantes são exibidos em uma lista e devem ser chamados individualmente, ou recorrer ao Emile, que adota um formato *swipe*, mais adequado a dispositivos móveis e mais simples de utilizar do que uma lista tradicional.

Em ambos os casos, contudo, o processo ainda é predominantemente manual, exigindo que cada aluno seja chamado para que sua presença ou falta seja registrada. Diante desse cenário, este trabalho tem como objetivo implementar uma solução baseada em *Bluetooth Low Energy* (*BLE*) para automatizar esse fluxo e tornar mais eficiente o processo de registro de presença dos alunos dentro do próprio aplicativo Emile.

1.2 Proposta de solução de software

O *BLE* é projetado para consumir pouquíssima energia, permitindo que dispositivos pequenos, como sensores, *wearables*, *beacons*, fechaduras inteligentes e equipamentos médicos, comuniquem-se de forma eficiente e contínua [3]. Ele opera por meio de conexões rápidas e pacotes curtos, podendo funcionar tanto no modo conectado quanto no modo de anúncio (*advertising*), onde transmite pequenos blocos de dados sem precisar parear. Graças ao seu baixo consumo, custo reduzido e boa cobertura, o *BLE* é amplamente usado em aplicações de *Internet of Things* (*IoT*), automação, rastreamento e integração com dispositivos móveis.

Nosso foco aqui é a integração entre dispositivos móveis, onde o dispositivo do aluno

atuará como um dispositivo de *advertising* no contexto *BLE*. Ele será responsável por transmitir periodicamente pacotes de dados, contendo informações essenciais para o registro da presença, para o dispositivo do professor, que funcionará como a Central. A Central, nesse caso, monitorará constantemente o ambiente para identificar os dispositivos de *advertising* e, ao receber os dados transmitidos, processará as informações enviadas pelo dispositivo do aluno.

Esse tipo de comunicação *BLE* permite uma troca de dados eficiente e de baixo consumo energético, ideal para contextos em que a comunicação entre os dispositivos precisa ser constante e sem interrupções, como no caso de registro de presença em ambientes educacionais. O processo garante que as informações sejam transmitidas de forma rápida e precisa, sem a necessidade de interação manual, otimizando a experiência tanto para o professor quanto para o aluno.

Figura 1: Topologias de comunicação BLE: Point-to-Point (via GATT) e Broadcasting (via GAP).

1.2.1 Conceitos fundamentais do *BLE*

Para a compreensão da solução proposta, é necessário detalhar os dois protocolos fundamentais que compõem a pilha de comunicação do *Bluetooth Low Energy*: o *GAP* e o *GATT*. Eles operam de maneira complementar: enquanto um gerencia a visibilidade e o acesso ao meio (incluindo a difusão de dados), o outro estrutura a troca de informações.

***GAP (Generic Access Profile)* [4]** O *GAP* atua como a camada de controle da rede. Ele é responsável por tornar o dispositivo visível ao mundo externo e determinar como dois dispositivos podem (ou não) interagir. É dentro das especificações do *GAP* que se define o mecanismo de *Broadcasting/Advertising*.

Neste modo de difusão, um dispositivo transmite pacotes de dados periodicamente para

o ambiente sem a necessidade de estabelecer uma conexão ponto-a-ponto. O *GAP* também define os papéis operacionais (*roles*) que os dispositivos podem assumir:

- ***Peripheral* (Periférico):** Dispositivo, geralmente de menor potência, que anuncia sua presença (*advertiser*) e aguarda conexões;
- **Central:** Dispositivo com maior capacidade de processamento que escaneia o ambiente (*scanner*) em busca de anúncios e inicia conexões.

***GATT (Generic Attribute Profile)* [5]** Enquanto o *GAP* gerencia a descoberta e a difusão, o *GATT* define a estrutura hierárquica dos dados. Ele utiliza o protocolo de transporte *ATT (Attribute Protocol)* [6] para organizar a informação em um modelo Cliente/Servidor.

Esta organização é essencial para a interoperabilidade entre dispositivos de fabricantes diferentes, estruturando-se em dois níveis principais:

- ***Services* (Serviços):** é um contêiner lógico utilizado para agrupar funcionalidades ou dados relacionados. Cada serviço é identificado univocamente por um *UUID* (*Universally Unique Identifier*) e é composto por uma ou mais *Characteristics*, podendo também incluir referências a outros serviços secundários. Sua função principal é organizar a estrutura de dados do dispositivo, permitindo que a Central descubra e compreenda quais funcionalidades o Periférico oferece (por exemplo: um serviço de monitoramento cardíaco, um serviço de bateria ou, no caso de aplicações proprietárias, um serviço de automação específico);
- ***Characteristics* (Características):** constituem a unidade fundamental de troca de dados. Elas representam o valor real da informação (como o nível de bateria, uma temperatura ou uma *string* de texto) e possuem permissões associadas que definem como a Central pode interagir com elas:
  - ***Read*:** Permite que o dispositivo cliente leia o valor atual armazenado na característica;
  - ***Write*:** Permite que o cliente envie dados para modificar o valor da característica (podendo exigir ou não uma confirmação de resposta);
  - ***Notify* e *Indicate*:** Mecanismos que permitem ao servidor (Periférico) enviar atualizações de valor automaticamente para o cliente (Central), sem a necessidade de solicitações repetidas de leitura (*polling*).

Além do valor principal e das propriedades, uma característica pode conter *Descriptors*, que fornecem informações adicionais sobre o dado, como seu formato, unidade de medida ou configurações de controle para notificações.

Em suma, qualquer aplicação *BLE* depende da interação entre estes dois perfis: o *GAP* para encontrar e conectar (ou apenas difundir dados via *broadcast*) e o *GATT* para entender e transacionar a informação útil.

1.2.2 Mecanismos de Comunicação Padrão do *BLE*

Uma vez compreendidos os papéis e protocolos, é necessário distinguir as duas formas primárias de troca de dados permitidas pela especificação *BLE*: a comunicação orientada a conexão e a comunicação via difusão. A escolha entre elas impacta diretamente o consumo de energia e a latência da aplicação.

Difusão (*Broadcasting*): Neste modelo, regido pelo perfil *GAP*, o dispositivo Periférico atua como um transmissor de rádio, enviando dados unidirecionais para qualquer dispositivo Central que esteja escutando, sem estabelecer um vínculo persistente. É o mecanismo mais eficiente para situações de ”um para muitos”, permitindo que um único dispositivo seja detectado por múltiplos *scanners* simultaneamente sem degradação de performance.

Conexão (*Connections*): Para interações que exigem segurança, bidirecionalidade ou garantia de entrega, o *BLE* utiliza o modelo conectado via *GATT*. Uma vez estabelecida a conexão, o canal de comunicação torna-se exclusivo entre os dois dispositivos. Dentro deste canal, destacam-se três operações de transporte de dados:

- ***Write* (Escrita):** Operação onde a Central envia dados para o Periférico. Pode ser configurada como ”Sem Resposta”(para velocidade) ou ”Com Resposta”(para garantir que o Periférico processou o dado);
- ***Notify* (Notificação):** O Periférico envia dados para a Central sem exigir confirmação de recebimento (menor latência, sem garantia de entrega);
- ***Indicate* (Indicação):** O Periférico envia dados e aguarda obrigatoriamente um pacote de confirmação (*ACK*) da Central antes de prosseguir (maior latência, garantia total de entrega).

1.2.3 Estrutura de Dados de Anúncio (*Advertising Data*)

Embora o mecanismo de *advertising* sirva primariamente para descoberta, o protocolo *BLE* permite a inclusão de pequenos blocos de informação nestes pacotes. O padrão define uma capacidade máxima de carga útil (*payload*) de 31 *bytes* por pacote no *BLE* legado (versões 4.x e 5.x sem extensões) [7].

Dentro desse espaço limitado, os dados são organizados em estruturas chamadas *AD Structures*, compostas por tamanho, tipo e valor. Um dos tipos mais relevantes para aplicações personalizadas é o *Manufacturer Specific Data* (Tipo 0xFF). Este campo permite que desenvolvedores insiram dados proprietários no pacote de anúncio, possibilitando a transmissão de informações (como identificadores de usuário) sem a necessidade de estabelecer uma conexão, técnica fundamental para cenários de alta densidade de dispositivos.

1.2.4 Intervalos de Publicidade (*Advertising Interval*)

A frequência com que um dispositivo periférico transmite seus pacotes de anúncio é determinada pelo *Advertising Interval*. Este parâmetro representa um compromisso crítico (trade-

off) em projetos *BLE* [7]:

- **Intervalos Curtos (menor que 100ms):** Facilitam a descoberta rápida e conexão imediata, mas aumentam drasticamente o consumo de bateria e a probabilidade de colisão de pacotes em ambientes com muitos dispositivos;
- **Intervalos Longos (maior que 1000ms):** Economizam energia, mas tornam a detecção lenta e a experiência do usuário ”travada”.

Para ambientes como salas de aula, onde dezenas de dispositivos competem pelo mesmo espectro de rádio, a escolha correta deste intervalo é essencial para evitar o congestionamento da banda de 2.4GHz.

1.3 Tecnologias adotadas

O desenvolvimento desse projeto demandou a adição de algumas tecnologias específicas ao Emile com base nas necessidades específicas da aplicação. Como o projeto envolve a detecção de dispositivos por meio do *BLE*, se fez necessária uma busca extensiva para identificar qual seria a melhor abordagem com ferramentas que fossem eficientes e bem documentadas.

- ***C++* [8]** é uma linguagem moderna, multiparadigma e de alto desempenho, com controle avançado de memória e uma biblioteca padrão (*STL*) rica em containers e algoritmos, adequada desde sistemas embarcados a aplicações desktop de missão crítica. Sua padronização contínua e diretrizes modernas fortalecem a robustez e a portabilidade do código.
- ***Qt* [9]** fornece um conjunto completo e multiplataforma de bibliotecas em C++, *User Interface* (*UI*) tradicional (*Qt Widgets*) e moderna (*Qt Quick/QML*), além de rede, banco de dados, multimídia e *bluetooth*, aliado a um modelo produtivo de programação com *signals/slots* e ferramentas oficiais como *Qt Creator*. Essa combinação permite iniciar projetos rapidamente, com pouca configuração manual, entregando interfaces nativas em *desktop*, *mobile* e embarcados.
  - ***Qt Creator* [10]** é uma *Integrated Development Environment* (*IDE*) multiplataforma que acelera o ciclo escrever-compilar-depurar para apps Qt em desktop, mobile e embarcados. Oferece editor avançado com modelo de código baseado em *Clang*[11], Kits para padronizar ambiente (*toolchain*, versão do *Qt*, *debugger*) e integração nativa com *CMake*[12]. Inclui assistentes para projetos *Qt Quick/QML* e *Qt Quick Designer*, além de *deploy/debug* para *Android* via *Android Debug Bridge* (*ADB*)[13] e empacotamento com *androiddeployqt*[14]. Esse conjunto reduz configuração manual e acelera da prova de conceito ao release.
- ***CMake* [12]** automatiza a configuração de sistemas de compilação (*build*). Ele controla o processo de compilação de *software* usando arquivos de configuração simples, chamados CMakeLists.txt. O *CMake* gera configurações e espaços de trabalho nativos de build que você pode usar no ambiente de compilador da sua preferência.

- ***BLE (Bluetooth Low Energy)*[3]** habilita a comunicação sem fio de baixo consumo na banda ISM de 2,4 GHz, com publicidade (*advertising*) e conexões baseadas em *Generic ATTribute Profile* (*GATT*) (serviços/características), ideais para sensores, *beacons* e *wearables*. O modelo Central/Periférico e o ecossistema de perfis padronizados aceleram a integração em *Android*, *iOS* e embarcados, mantendo troca de dados eficiente em pequenos pacotes e longa autonomia de bateria.

- ***WebScraping*** é uma técnica utilizada para extrair dados automaticamente de páginas da web, simulando a navegação humana, mas de forma programática e muito mais rápida. Por meio de scripts ou ferramentas especializadas, o scraper acessa um site, interpreta seu conteúdo, geralmente em HTML, e coleta informações específicas, como textos, preços, tabelas ou imagens, estruturando-as em formatos úteis para análise ou integração com outros sistemas.

1.4 Trabalhos relacionados

A automação do registro de presença utilizando tecnologias sem fio não é um tema inédito na literatura e no mercado. Diversas soluções buscaram mitigar o problema da chamada manual, cada uma adotando estratégias distintas de arquitetura e usabilidade. Nesta seção, analisamos três trabalhos que serviram de referência para o desenvolvimento do Emile, destacando suas contribuições e as lacunas que nossa proposta visa preencher.

1.4.1 *Presence: Bluetooth Attendance*

O aplicativo *Presence* propõe a criação de ”salas virtuais”onde o professor atua como hospedeiro e os alunos realizam um *check-in* digital ao entrarem no alcance do rádio *Bluetooth*. Embora a solução elimine a chamada oral, ela depende de uma interação ativa do usuário (abrir o app e solicitar entrada) e opera sob um modelo de conexão tradicional.

Diferencial do Emile: Ao contrário do *Presence*, o Emile adota uma abordagem de descoberta passiva baseada em *advertising*. Isso remove a necessidade de ”*check-in*”manual: basta que o aplicativo do aluno esteja transmitindo para que a presença seja detectada, validada e confirmada automaticamente, tornando o processo transparente.

1.4.2 Sistema de Frequência da Unioeste

No âmbito acadêmico nacional, destaca-se o projeto desenvolvido na Universidade Estadual do Oeste do Paraná (Unioeste). A solução utiliza uma arquitetura cliente-servidor móvel para realizar a chamada via *Bluetooth*, focando na integração com o sistema acadêmico local. O trabalho demonstrou a viabilidade do uso de dispositivos móveis para este fim, mas enfrentou desafios comuns de interoperabilidade entre diferentes sistemas operacionais móveis.

Diferencial do Emile: O Emile evolui essa proposta ao implementar uma camada de controle em **C++**/**Qt** com tratamento específico para as limitações de *payload* do *iOS* (31 bytes). Enquanto soluções anteriores frequentemente falham em detectar *iPhones* em modo

de suspensão, nossa arquitetura inverte a lógica de empacotamento de dados para garantir interoperabilidade total entre *Android* e *iOS*.

1.4.3 *Attendance Taker*

O *Attendance Taker* é uma solução de mercado que permite ao professor escanear dispositivos próximos e exportar a lista de presença. No entanto, sua arquitetura de dados é desconectada: o resultado final é geralmente um arquivo CSV ou texto que o professor deve processar manualmente ou enviar por e-mail para consolidar os dados.

Diferencial do Emile: A principal limitação do *Attendance Taker* é a falta de integração sistêmica. O Emile resolve isso eliminando o passo intermediário de exportação de arquivos. Através da comunicação direta com o *SUAP* via *WebScraping*, a presença detectada pelo *Bluetooth* é convertida instantaneamente em um registro oficial no diário de classe, fechando o ciclo de automação.

Abaixo segue uma pequena tabela comparando o trabalho proposto nesse projeto com os trabalhos apresentados por outras instituições:

Tabela 1: Comparação entre soluções de registro de presença

Critério	**NFC (2020)**	**Reconhecimento Facial (2024)**	**BLE (Proposto)**
Interação do Aluno	**Ativa:** Exige deslocamento físico e aproximação do dispositivo (1:1).	**Passiva:** Exige posicionamento frontal à câmera e boa iluminação.	**Mínima:** Aluno ativa o modo de anúncio em seu lugar; detecção automática.
Simultaneidade	**Baixa:** Processamento serial (um aluno por vez).	**Média:** Limitada pelo ângulo da câmera e capacidade de processamento.	**Alta:** Leitura paralela de múltiplos alunos via *Advertising*.
Infraestrutura	*Tags NFC* ou leitores específicos.	Câmeras de alta resolução e servidor de *IA*.	Apenas os *smartphones* já existentes (BYOD).
Custo	Médio (aquisição de *tags*).	Alto (hardware e licenças de software).	Zero (baseado em *software*).
Privacidade	Alta (dados no cartão).	Baixa (dados biométricos sensíveis).	Média (apenas matrícula e código da turma).
Dependência de Rede	Independente (local).	Alta (geralmente processado em nuvem).	Independente (comunicação direta *device-to-device*).
Compatibilidade	Limitada (nem todos celulares possuem *NFC*).	Alta (qualquer câmera), mas depende de luz.	Universal (*Bluetooth* 4.0+ é padrão de mercado).

2 Requisitos

2.1 Requisitos funcionais

RF1 - O sistema deve permitir ao professor optar pelo registro de presença via método tradicional (manual) ou via *Bluetooth Low Energy* (*BLE*)

RF2 - Caso o professor opte pelo método tradicional, o sistema deve redirecionar para a interface de chamada padrão já existente na aplicação.

RF3 - Se o professor escolher realizar o registro de presença pelo *BLE*, o sistema deve redirecionar a visualização para a nova tela onde o fluxo por *Bluetooth* inicia.

RF4 - Ao abrir a tela pelo fluxo do *BLE*, o sistema deverá gerar e disponibilizar para o

professor um código de sessão único utilizando-se de algumas informações como parâmetros para gerar essa informação:

- ID da aula ministrada
- Data/Hora

RF5 - Ao abrir a tela pelo fluxo do *BLE*, o sistema deve recuperar do *SUAP* a lista de alunos matriculados na disciplina e deve exibir essa informação em formato de lista com o nome do aluno, a quantidade de ausências para aquela aula e se ele está presente ou não

RF6 - Ao abrir a tela pelo fluxo do *BLE*, o dispositivo do professor deve atuar como Central, realizando o escaneamento passivo (*Scanning*) do ambiente em busca de pacotes de anúncio que contenham dados de identificação acadêmica no campo *Manufacturer Data*.

RF7 - O aplicativo do aluno deve apresentar uma funcionalidade dedicada para ”Registrar Presença via Bluetooth”, permitindo a inserção manual do Código da Turma fornecido pelo professor.

RF8 - O dispositivo do aluno deve atuar como *Peripheral* em modo *Advertising* (anunciante), transmitindo periodicamente um pacote contendo sua matrícula e o código da turma inserido. O modo deve permitir conexões efêmeras para recebimento de confirmação.

RF9 - O sistema do professor deve descartar pacotes cujo o código de sessão único enviado pelo aluno não corresponda ao código gerado para aquela aula.

RF10 - O sistema deve implementar lógica de deduplicação, garantindo que a presença de um aluno seja registrada apenas uma vez por sessão, independentemente de quantas vezes o dispositivo for detectado pelo *scanner*.

RF11 - O sistema do professor deve enviar uma confirmação (*ACK*) ou invalidação (*INVALID*) ao aluno via escrita em característica *BLE* (*Write With Response*), atualizando visualmente o status no dispositivo do aluno.

RF12 - O sistema deve permitir a correção de estado em tempo real: caso um aluno inicialmente detectado com código incorreto corrija a informação, o sistema do professor deve reconhecer a alteração, remover a restrição e validar a presença.

2.2 Requisitos não-funcionais

RNF1 - O sistema deve ser desenvolvido utilizando o *framework Qt* e a linguagem *C++*, garantindo portabilidade de código..

RNF2 - A implementação do protocolo *BLE* deve garantir interoperabilidade entre as plataformas *Android* e *iOS*, tratando especificamente as limitações de tamanho de pacote de anúncio (31 bytes) do sistema *iOS*.

RNF3 - O sistema deve suportar o processamento de turmas com densidade média a alta, utilizando filas de processamento para evitar congestionamento do adaptador *Bluetooth*.

RNF4 - A interface do usuário (UI) não deve sofrer travamentos durante o processo de escaneamento e conexão, devendo as operações de rádio ocorrerem de forma assíncrona.

RNF5 - O sistema deve possuir mecanismos de robustez (*Watchdog*) para recuperar-se automaticamente de falhas de conexão transientes (como erros *GATT*) sem intervenção do usuário.

RNF6 - O intervalo de *Advertising* do aluno deve ser configurado (entre 160ms e 240ms) para balancear a descoberta rápida e o consumo de bateria, evitando a saturação do espectro de rádio em salas cheias.

RNF7 - O processo de registro não deve exigir pareamento (*bonding*) prévio entre os dispositivos do professor e dos alunos, funcionando através de descoberta e conexão pontual.

3 Design

3.1 Projeto C4

Esta seção apresenta as decisões de design arquitetural adotadas no desenvolvimento do Emile, com foco na organização estrutural do sistema e na representação de seus principais elementos. Para esse fim, foi adotado o *C4 Model*, um modelo de documentação arquitetural baseado em múltiplos níveis de abstração, que permite descrever o sistema de forma progressiva, desde sua interação com usuários e sistemas externos até a decomposição interna de seus módulos [15].

A escolha do C4 justifica-se por sua clareza, aderência a sistemas distribuídos e capacidade de comunicar decisões arquiteturais de maneira objetiva e compreensível, características essenciais para o contexto multiplataforma e orientado a serviços do Emile.

- **Diagrama de Contexto do Sistema**
  O diagrama de contexto apresenta uma visão de alto nível do Emile, evidenciando seus principais usuários e os sistemas externos com os quais interage. Seu objetivo é delimitar claramente as fronteiras do sistema, destacando o papel do Emile no ecossistema institucional e suas dependências externas.

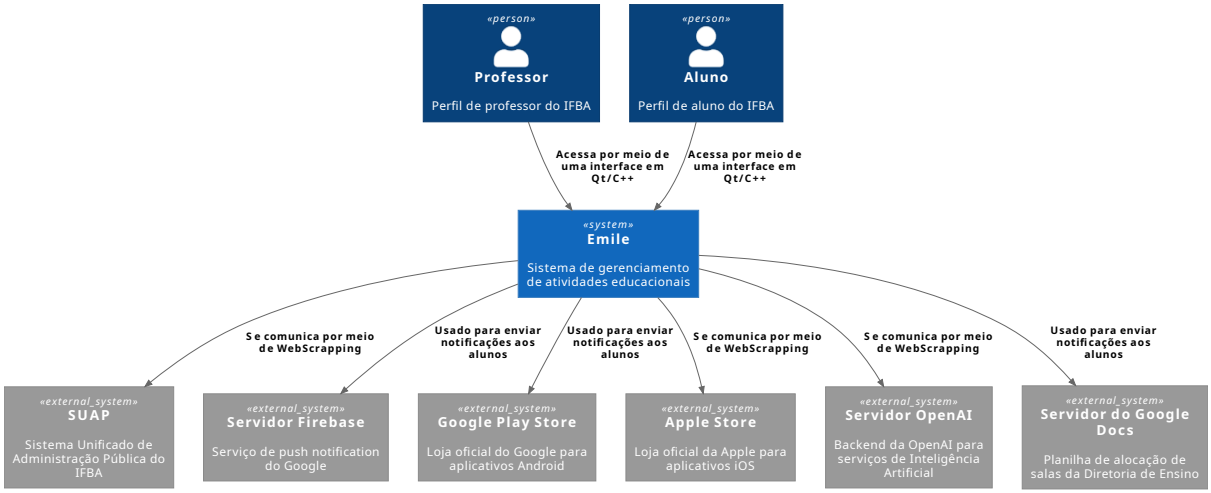


Figura 2: Diagrama de Contexto do Sistema do Emile

- **Diagrama de Containers**
  O diagrama de containers detalha a decomposição do Emile em suas principais aplicações e serviços, evidenciando responsabilidades, tecnologias utilizadas e formas de comunicação entre os containers. Essa visão permite compreender como o sistema é estruturado do ponto de vista lógico e tecnológico.

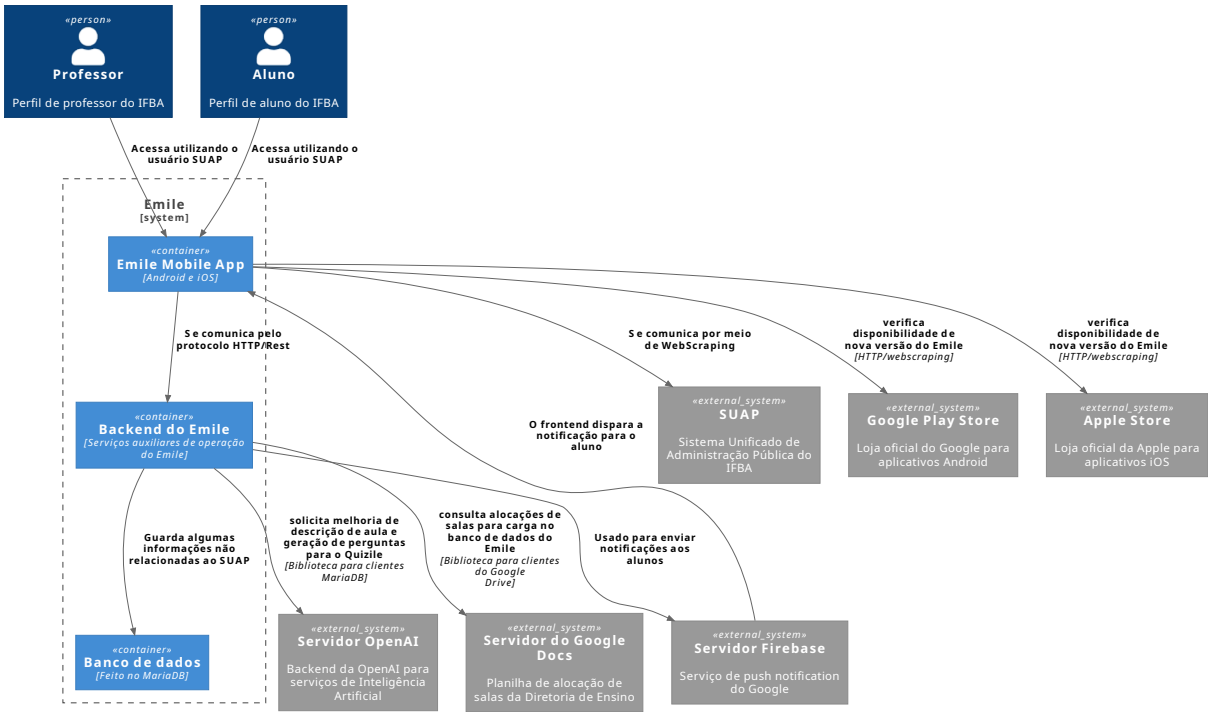


Figura 3: Diagrama de Containers do Emile

- **Diagrama de Componentes**
  O diagrama de componentes aprofunda a visão interna de um contêiner específico, destacando os principais módulos responsáveis pelo registro de presença via *Bluetooth Low Energy*. Esse diagrama explicita as responsabilidades de cada componente e suas

interações, auxiliando na compreensão das decisões arquiteturais relacionadas ao protocolo *BLE*.

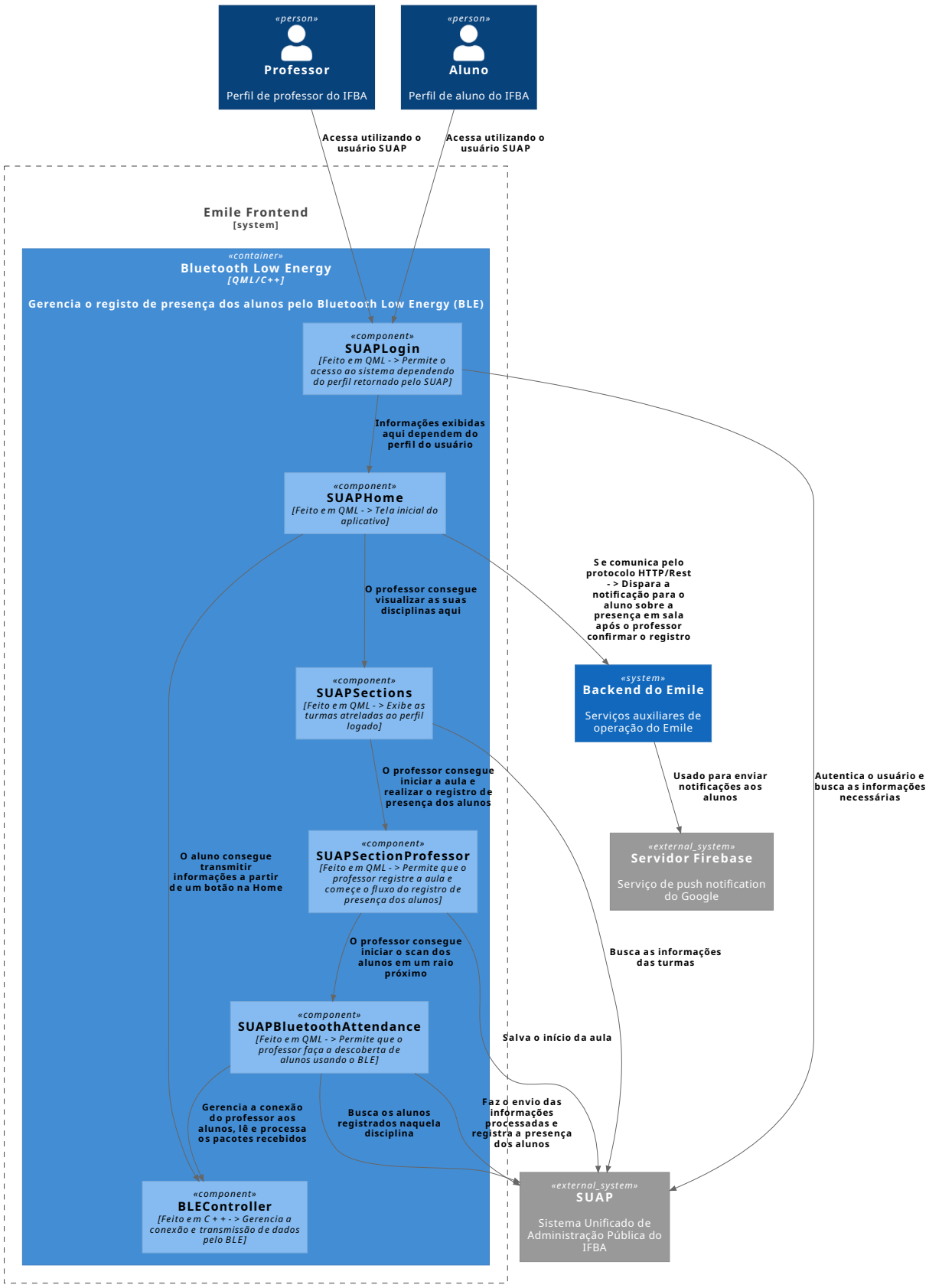


Figura 4: Diagrama de Componentes do Emile no contexto do *Bluetooth Low Energy*

3.2 Visão arquitetural

A arquitetura proposta para esse projeto foi pensada para operar de forma distribuída entre dois dispositivos móveis, o *smartphone* do aluno e o do professor, utilizando o modelo *Central/Periférico* definido pela especificação *BLE*. Essa organização permite que cada dispositivo assuma responsabilidades bem delimitadas, mantendo baixo consumo energético, alta responsividade e compatibilidade com *Android* e *iOS*, conforme estabelecido nos requisitos não funcionais.

Do ponto de vista arquitetural, a solução é composta por duas camadas principais:

3.2.1 Camada de interface (QML/Qt Quick)

Responsável pela interação com o usuário e pela orquestração visual das ações do *BLEController*. Esta camada engloba os componentes de interface (*SUAPHome, SUAPSectionProfessor, SUAPBluetoothAttendance*), que se comunicam com o módulo de controle *BLE* através do sistema de *Signals/Slots* nativo do *framework Qt*, garantindo que a interface permaneça responsiva mesmo durante operações intensivas de rádio.

3.2.2 Camada de Controle *BLE* (*BLEController* em C++)

Esta camada gerencia a máquina de estados do protocolo e a lógica de negócios da comunicação, implementando estratégias específicas de tolerância a falhas para garantir a interoperabilidade entre plataformas:

Modo Aluno (*Peripheral - Advertiser*) O dispositivo do aluno opera transmitindo pacotes de anúncio (*Advertising Packets*) em intervalos não determinísticos ajustados (160ms a 240ms) para minimizar colisões de rádio em ambientes densos. Para contornar as restrições de tamanho de pacote, a arquitetura inverte a lógica padrão de descoberta:

- **Priorização de *Payload*:** O campo *Manufacturer Data*, contendo a matrícula e o código da turma, é inserido no pacote principal de *Advertising*. Isso assegura que dispositivos Central consigam ler os dados imediatamente, sem depender da solicitação de *Scan Response*;
- **Segurança de Sessão:** Implementa-se um *timeout* automático (configurado para 5 minutos) que interrompe a transmissão automaticamente, economizando bateria e prevenindo tentativas de registro fora da janela de tempo da chamada.

Modo Professor (Central - *Scanner* e *Arbiter*) O dispositivo do professor atua como orquestrador, executando um ciclo contínuo de três estágios:

- **Filtragem e Fila:** O *Scanner* intercepta pacotes de anúncio em tempo real. Se o código da turma contido no pacote for válido, a matrícula é inserida em uma fila de processamento *FIFO* (*First-In, First-Out*), evitando o descarte de alunos detectados simultaneamente;

- **Conexão Blindada (*Watchdog*):** Ao processar a fila, o sistema estabelece uma conexão *GATT* exclusiva com o aluno. Um mecanismo de *Watchdog* (temporizador de segurança de 5 segundos) monitora a transação; caso ocorra instabilidade ou falhas, o *Watchdog* aborta a conexão, limpa os recursos e reinicia o ciclo, impedindo o travamento da aplicação;

- **Redenção de Estado:** O sistema monitora alterações nos pacotes de alunos já processados. Caso um aluno marcado como ”Inválido”corrija sua matrícula e reinicie a transmissão, o sistema detecta a mudança no *payload* do anúncio, remove a restrição da lista negra e processa a presença correta automaticamente.

3.3 Estratégia de Protocolo e Confirmação

Para atender aos requisitos de desempenho em salas de aula densas e garantir a interoperabilidade entre *Android* e *iOS*, o Emile adota uma arquitetura híbrida de comunicação:

- **Leitura via Difusão (*Advertising*):** Diferente da abordagem tradicional onde a Central precisa conectar-se para ler características, o Emile extrai a matrícula e a turma diretamente dos pacotes de *Advertising* (campo *Manufacturer Data*). Isso transforma a descoberta em um processo passivo e paralelo, permitindo que o Professor filtre dezenas de alunos simultaneamente sem o *overhead* temporal de estabelecer conexões *GATT* completas;

- **Confirmação via Escrita com Resposta (*Write With Response*):** Embora a literatura clássica sugira o uso de *Indicate* para dados críticos, esta operação é unidirecional (Periférico $\rightarrow$ Central). Como a autoridade de confirmação de presença reside no Professor (Central), a arquitetura utiliza o método *Write With Response*. Neste fluxo, o Professor conecta-se brevemente ao aluno apenas para ”carimbar”a presença. O protocolo garante que a operação só seja concluída se o dispositivo do aluno enviar o *hardware ACK*, assegurando que o *feedback* visual (mensagem de confirmação tela do aluno) realmente ocorreu.

Esta separação de responsabilidades (Dados no *Advertising* para leitura massiva e Conexão Pontual para confirmação individual) elimina a necessidade de manter conexões abertas, resolvendo o gargalo de limite de dispositivos simultâneos e garantindo escalabilidade.

3.4 Descrição do processo de registro de frequência com *BLE*

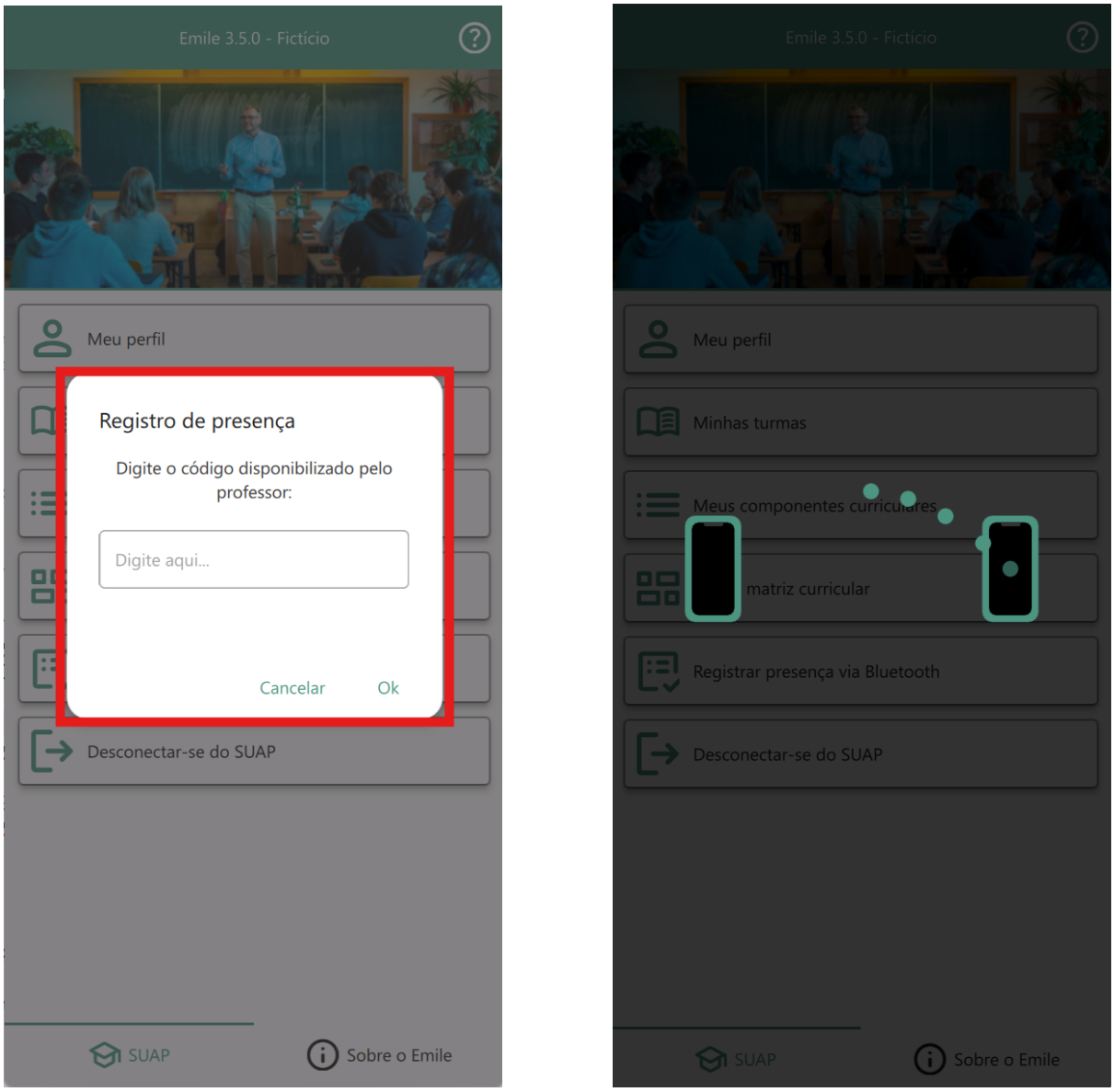
3.4.1 Fluxo do aluno

Como foco na facilidade de uso, o aluno só precisa seguir alguns passos para conseguir transmitir sua presença ao professor:

- Com um usuário do tipo aluno logado, clique no botão ”Registrar presença via *Bluetooth*”;

Figura 5: Botão ’Registrar presença via Bluetooth’

- Digite o código único da sessão disponibilizado pelo professor e confirme.



(a) Modal com o campo input

(b) Envio de dados ao professor

Figura 6: Fluxo de envio de dados pelo aluno

Com isso, o dispositivo do aluno inicia o modo de anúncio (*Advertising*), transmitindo pacotes de presença continuamente em intervalos não determinísticos (entre 160ms e 240ms). A transmissão persiste de forma contínua até que uma das condições de encerramento seja atendida: o recebimento da confirmação de presença (*ACK*) enviada pelo professor via escrita *GATT* ou o fim do tempo limite da sessão (configurado para 5 minutos). Para melhor detalhar esta lógica de estados, segue abaixo o diagrama *BPMN* com o fluxo do aluno:

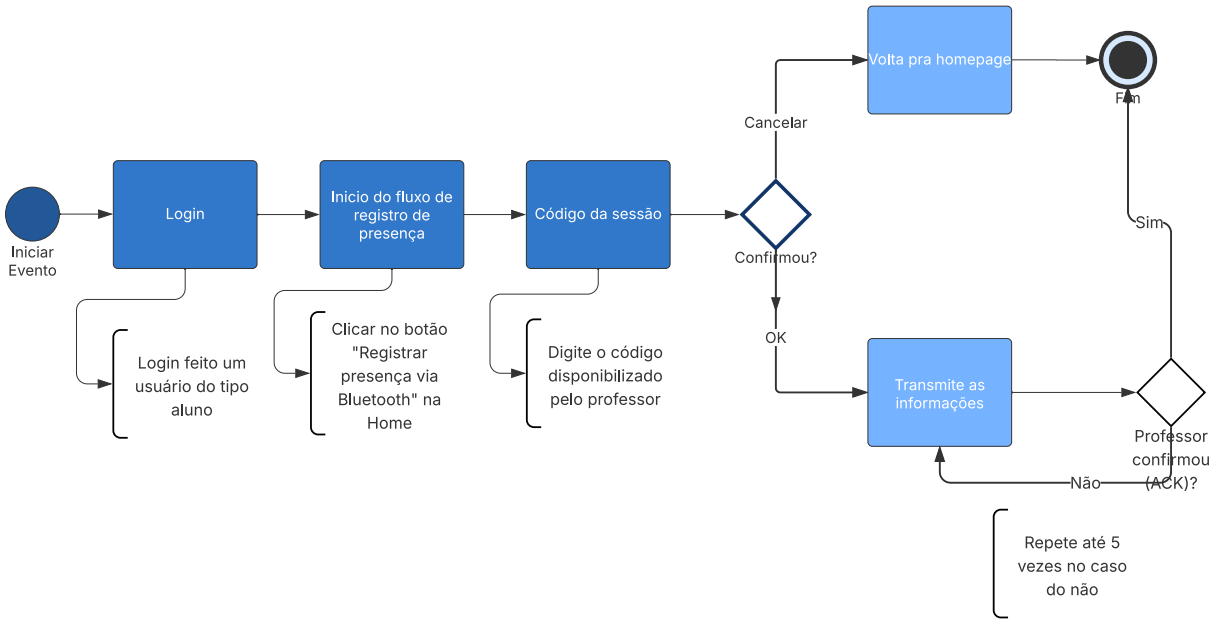
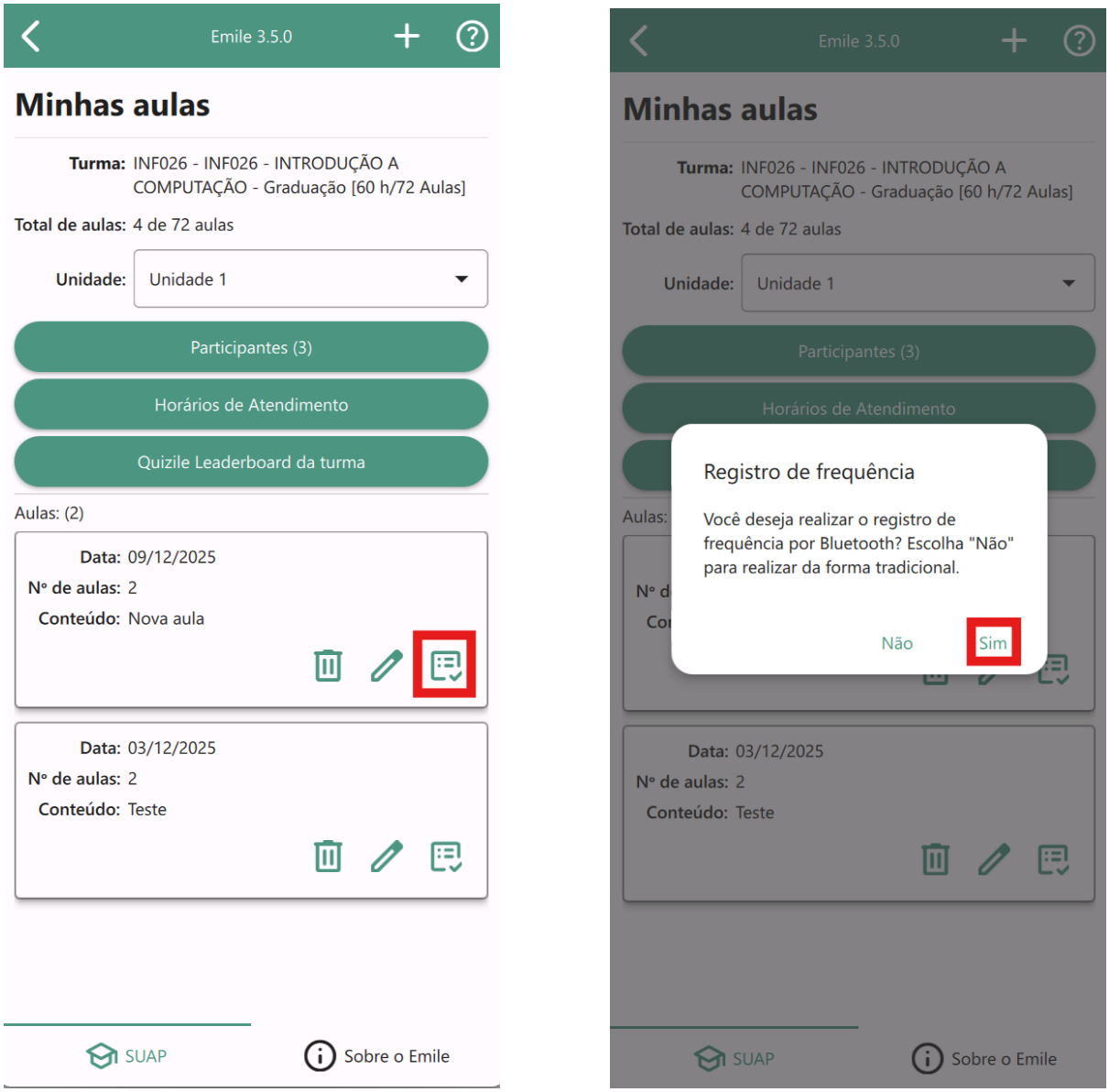


Figura 7: Diagrama BPMN do fluxo do aluno

3.4.2 Fluxo do professor

Embora o perfil docente envolva etapas adicionais para o controle da chamada, como demonstrado nas figuras abaixo, a interface foi projetada para manter a linearidade e a fluidez do processo.

- Com um usuário do tipo professor logado e após iniciar a aula, comece o fluxo do registro de presença e escolha via *Bluetooth*;

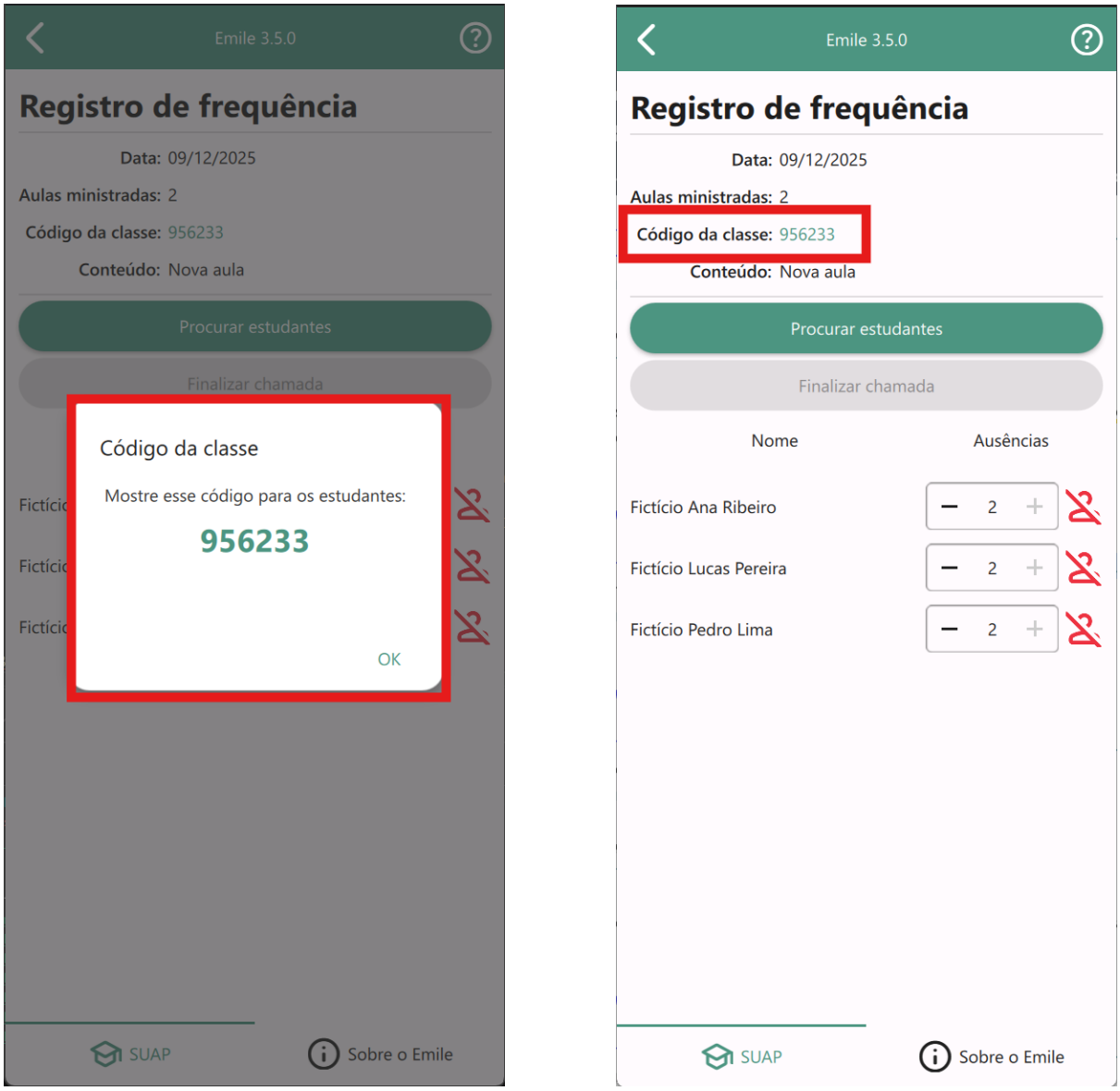


(a) Iniciar registro de presença

(b) Iniciar via *Bluetooth*

Figura 8: Fluxo de início do registro de presença pelo professor

- Passe o código único para os alunos. Ele aparece em uma modal assim que abrir a tela, mas também pode ser visualizado na parte superior da tela junto as outras informações da aula;



(a) Código único ao iniciar

(b) Código único no cabeçalho

Figura 9: Localização dos códigos que devem ser passados para os alunos

- Etapa de busca após apertar no botão ”Procurar estudantes”;

(a) Iniciar busca por alunos

(b) Radar exibido durante a busca

Figura 10: Busca por alunos no entorno do professor

- Finalize o processo registrando a presença de quem foi encontrado

Figura 11: Registrar presença

O professor pode mandar escanear quantas vezes achar necessário, isso é útil em casos ao não identificar alunos com um sinal muito baixo. Para melhor exemplificar segue abaixo um diagrama *BPMN* com o fluxo do professor:

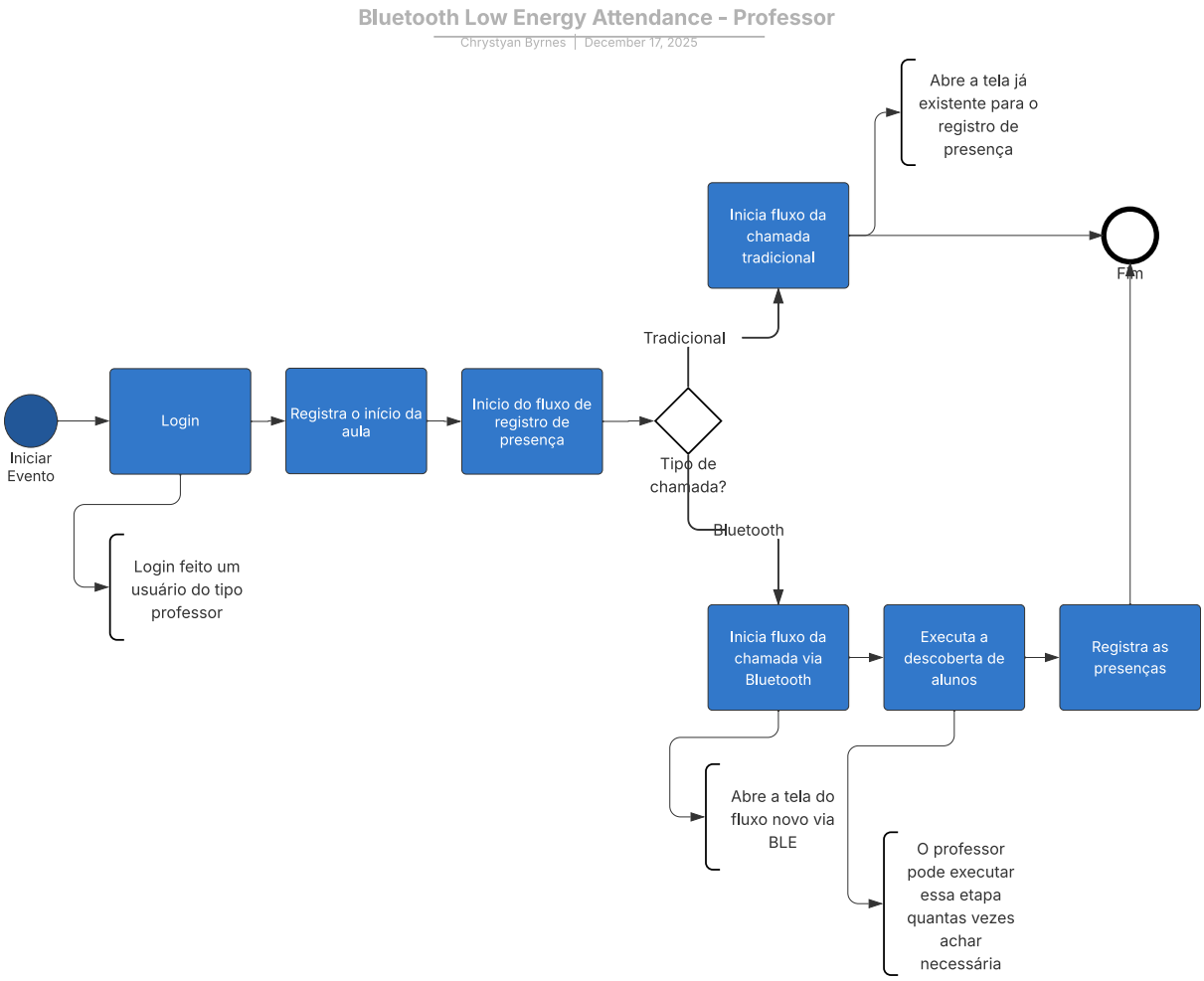


Figura 12: Diagrama BPMN do fluxo do professor

4 Implantação

O processo de disponibilização do aplicativo Emile nas lojas oficiais de aplicativos envolve etapas específicas para cada plataforma, seguindo as diretrizes da *Google Play Store*, para dispositivos *Android*, e da *Apple App Store*, para dispositivos *iOS*. Em ambos os casos, é necessário realizar o cadastro do desenvolvedor, configurar os metadados do aplicativo, como nome, descrição, ícones, capturas de tela e classificação indicativa, e gerar versões assinadas para publicação. No ambiente *Android*, o aplicativo é distribuído no formato *Android App Bundle* (*AAB*), enquanto no *iOS* o processo exige a criação de certificados digitais, perfis de provisionamento e o envio do aplicativo por meio do *Xcode* ou do *Transporter*, integrados ao ecossistema da *Apple Developer Program*.

Considerando que o Emile utiliza *BLE* como parte essencial de sua arquitetura para o registro automatizado de presença, ambas as plataformas impõem requisitos específicos quanto ao uso e à justificativa das permissões solicitadas. Na *Google Play Store*, o aplicativo deve declarar permissões relacionadas ao *Bluetooth*, além de permissões de localização quando exigidas pelo sistema operacional, uma vez que o escaneamento *BLE* pode permitir inferências de proximidade física. Essas permissões devem ser claramente justificadas na política de pri-

vacidade, demonstrando que o uso do *Bluetooth* está diretamente associado à funcionalidade principal do sistema.

De forma semelhante, a *Apple App Store* adota um processo rigoroso de validação baseado nos princípios de transparência e necessidade funcional. O aplicativo deve informar explicitamente, no arquivo de configuração *Info.plist*, a finalidade do uso do *Bluetooth*, apresentando ao usuário uma explicação clara no momento da solicitação da permissão. Além disso, é fundamental assegurar que os dados coletados sejam limitados ao necessário para o funcionamento do aplicativo, sem fins de rastreamento indevido.

Por fim, o processo de validação em ambas as lojas envolve análises automáticas e manuais que avaliam a estabilidade do aplicativo, a coerência entre as permissões solicitadas e as funcionalidades oferecidas, bem como a conformidade com as políticas de privacidade. Dessa forma, um uso planejado e bem documentado do *Bluetooth* é determinante para a aprovação do Emile e sua posterior disponibilização aos usuários finais.

5 Referências

1 ANDRADE, S. S. **Emile**: aplicativo móvel. 2025. Disponível para iOS (App Store) e Android (Google Play). Disponível em: ⟨https://play.google.com/store/apps/details?id=br.edu.ifba.gsort_emile&hl=pt_BR⟩. Acesso em: 17 fev. 2026.

2 INSTITUTO FEDERAL DO RIO GRANDE DO NORTE. **SUAP: Sistema Unificado de Administração Pública**. Natal, RN: [s.n.], 2026. Disponível em: ⟨https://suap.ifrn.edu.br/⟩. Acesso em: 17 fev. 2026.

3 , G. . **Bluetooth Low Energy — Connectivity**. 2025. Disponível em: ⟨https://developer.android.com/develop/connectivity/bluetooth/ble/ble-overview⟩.

4 SIG, B. **Bluetooth Core Specification: Volume 3, Part C – Generic Access Profile (GAP)**. 2025. Disponível em: ⟨https://www.bluetooth.com/specifications/bluetooth-core-specification/⟩.

5 ____. **Bluetooth Core Specification: Volume 3, Part G – Generic Attribute Profile (GATT)**. 2025. Disponível em: ⟨https://www.bluetooth.com/specifications/bluetooth-core-specification/⟩.

6 Bluetooth SIG. **Bluetooth Core Specification: Attribute Protocol (ATT)**. 2024. Vol. 3, Part F (Attribute Protocol). Acesso em: 21 jan. 2026. Disponível em: ⟨https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Core-54/out/en/host/attribute-protocol--att-.html⟩.

7 TOWNSEND, K. et al. **Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking**. Sebastopol, CA: O'Reilly Media, 2014. ISBN 978-1491949511.

8 , C. . **cppreference.com**. 2025. Disponível em: ⟨https://cppreference.com⟩.

9 , T. Q. C. . **Qt Overviews**. 2025. Disponível em: ⟨https://doc.qt.io/qt-6/overviews-main.html⟩.

10 ____. **IDE Overview — Qt Creator Manual**. 2025. Disponível em: ⟨https://doc.qt.io/qtcreator/creator-overview.html⟩.

11 ____. **Clang Code Model — Qt Creator Documentation**. 2025. Disponível em: ⟨https://doc.qt.io/qtcreator/creator-clang-codemodel.html⟩.

12 ____. **CMake — Qt Creator Documentation**. 2025. Disponível em: ⟨https://doc.qt.io/qtcreator/creator-project-cmake.html⟩.

13 ____. **Android Deploy Configuration — Qt Creator Documentation**. 2025. Disponível em: ⟨https://doc.qt.io/qtcreator/creator-deploying-android.html⟩.

14 ____. **Developing for Android — Qt Creator Documentation**. 2025. Disponível em: ⟨https://doc.qt.io/qtcreator/creator-developing-android.html⟩.

15 BROWN, S. **The C4 model for visualising software architecture**. 2026. Disponível em: ⟨https://c4model.com/⟩. Acesso em: 17 fev. 2026.