GAT 108 – Automação Avançada

Marcus Vinícius Oliveira Pacheco

AV2
Relatório de aplicativo
Cross Docking

LAVRAS - MG
JULHO DE 2023

Introdução

Nesse relatório, será abordado sobre como foi feito o desenvolvimento do aplicativo *Cross Docking*. O princípio é que o aplicativo recomende a velocidade ao usuário para que seja possível chegar no tempo estipulado. Caso outro dispositivo estabeleça comunicação com o primeiro, ambos sincronizam o destino para chegar ao mesmo tempo, alterando a velocidade recomendada conforme necessário.

ThreadLocation

Essa é a classe que implementa o uso de *Thread* e é responsável por obter os dados do GPS do celular. Nela, são estabelecidos uma distância a ser percorrida, o tempo estipulado e quantos fluxos serão utilizados para a reconciliação. Quando iniciada, a *thread* usa da biblioteca *Location* para adquirir as informações de posição do dispositivo e calcular a distância percorrida e a velocidade atual. Posteriormente, define os vetores iniciais a serem utilizados para fazer a reconciliação de dados, com os tamanhos correspondentes ao número de nós. O vetor Y é preenchido com o valor de tempo que o veículo deverá gastar para chegar na distância do nó, e o vetor V recebe um valor de incerteza do GPS estipulado. Depois, é utilizado o código fornecido no ambiente da disciplina para fazer a reconciliação, baseando na matriz A feita a partir da informação que F1 = -F2 – F3 ... e assim por diante, sendo então uma matriz de diagonal igual a 1 e o restante igual a -1. Quando se percorre a distância equivalente ao próximo nó, o valor de tempo gasto é colocado na matriz Y junto com o atraso ou adiantamento. O método *reconciledFlow* do código *Reconciliation* fornece então o valor utilizado para calcular a velocidade necessária para chegar no destino a tempo. Na Thread também consta um método a ser utilizado no serviço de *Cross Docking*, que analisa se o tempo estipulado pelo veículo é maior que o seu e, caso seja, atualiza o tempo estipulado para que ambos consigam chegar juntos.

ServicoTransporte

Essa é a classe responsável por organizar os atributos de número de identificação do serviço, data/hora de início e fim, lista de Passageiros, lista de Carga e lista de Motoristas.

JSONManager e NetworkManager

São as classes responsáveis por tratar o envio e recebimento de mensagens JSON. A *JSONManager* coloca as informações no formato necessário, enquanto a *NetworkManager* comunica com um servidor internet. Este servidor fica localizado na porta 10.0.2.2:8080, no qual são feitas as requisições de mensagens.

JsonWebTokenUtility

É a classe responsável por criptografar as mensagens. É feito utilizando o padrão aberto JWT, numa chave mestra definida aleatoriamente como 4317985356778213, utilizada nos métodos de codificação e decodificação das informações.

Interface

É uma interface simples, que mostra as informações de distância total, distância percorrida, tempo total, tempo percorrido e a velocidade recomendada. Possui dois botões, um para começar e outro para parar o serviço.

MainActivity

É classe principal da aplicação. Depois de declaradas todas as variáveis a serem usadas, ela inicia associando as informações do layout à suas respectivas variáveis. Depois, é feita a checagem da permissão de uso das informações de localização do dispositivo, e feito pedido de uso caso a permissão ainda não tenha sido concedida.

O primeiro método é a sobrescrição do método onRequestPermissionsResult que, a partir do clique no botão, instância uma thread e a inicia. O método onLocationUpdate atualiza as informações na tela e chama os métodos para comunicação via JSON.

O método sendEncryptedJson é responsável por enviar a mensagem JSON criptografada. Ele possui um semáforo para não permitir que mais de um processo tente enviar mensagem ao servidor ao mesmo tempo através da biblioteca Semaphore. Com uma estrutura Try/Catch, a permissão ao acesso é verificada e, se concedida, cria uma mensagem JSON com o método createJson , encriptograda utilizando a classe JsonWebTokenUtility e envia a mensagem pelo NetworkManager, liberando o semáforo após a conclusão.

No método fetchFromServer, é utilizado um outro semáforo para que não seja possível mais de um processo acessar o servidor ao mesmo tempo. Se for permitido o acesso, são usados os métodos para descriptografar da classe JsonWebTokenUtility e são checadas se as mensagens não vêm do próprio dispositivo e se não são mensagens já lidas, posteriormente as informações do outro dispositivo são armazenadas, é chamado o método de Cross Docking da Thread para atualizar o valor de tempo estipulado para a chegada e assim calcular a velocidade correta e, por fim, o semáforo é liberado.