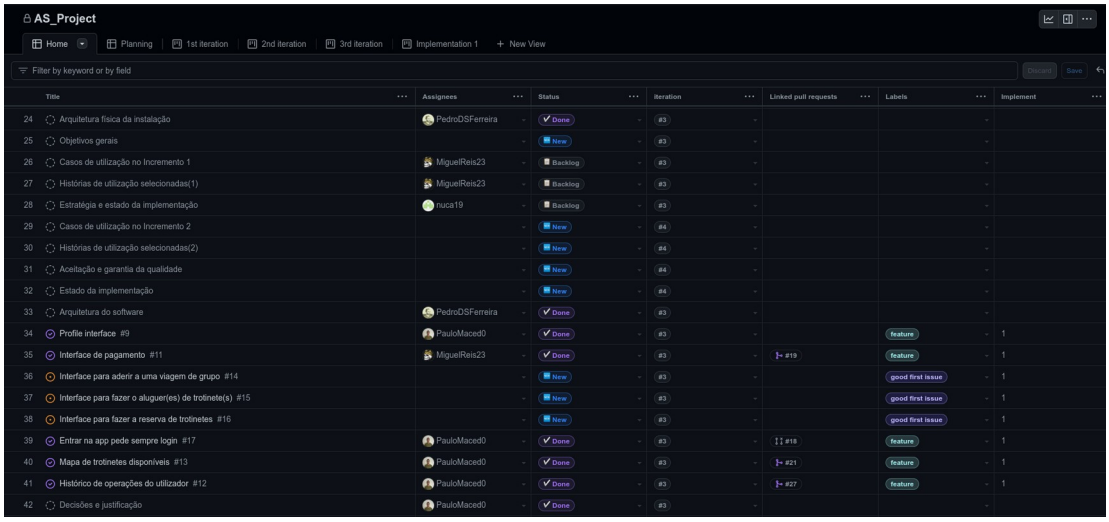
Francisco Fernandes (98178), Miguel Reis (108545), Nuno Pinho (108648), Paulo Macedo (102620), Pedro Ferreira (98620)
Versão deste relatório: **2022-06-05**, v1.0

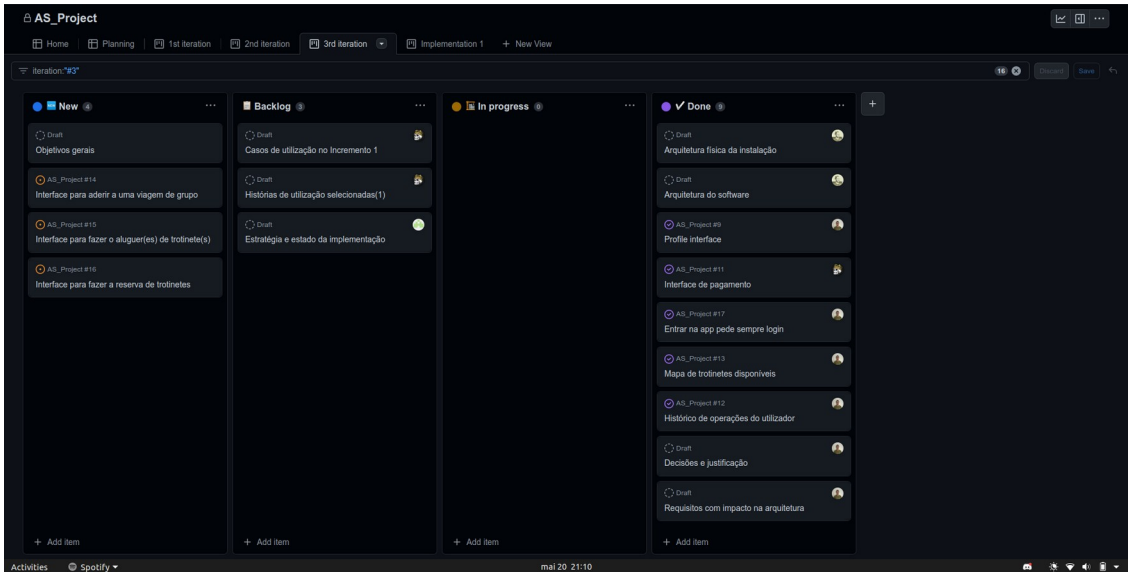
RELATÓRIO – E1

1.1 Gestão das tarefas

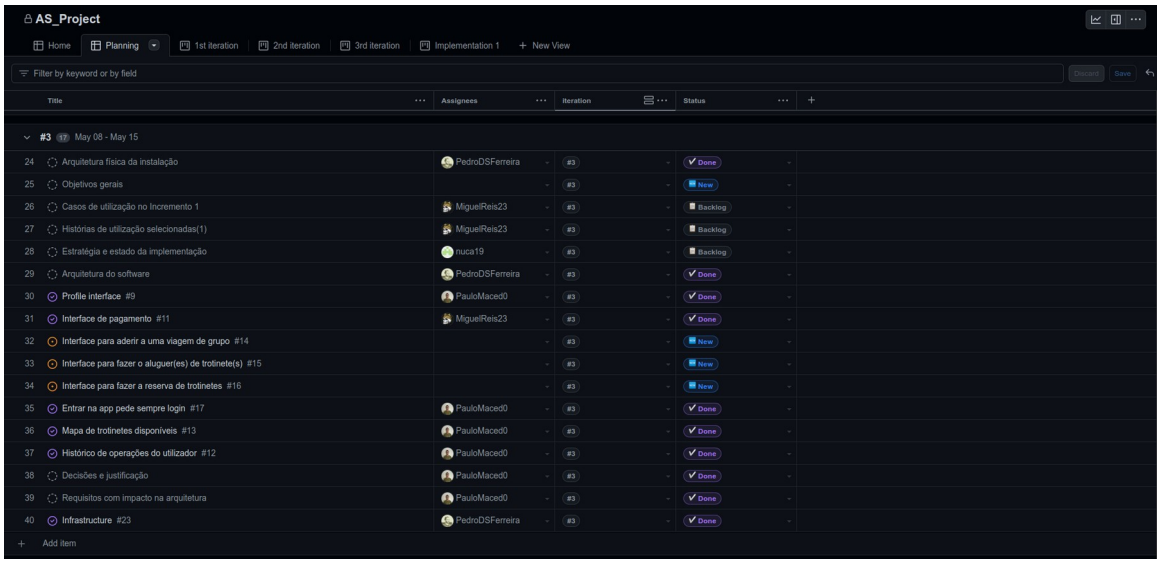
No seguinte quadro, é feita a adição de novas tarefas e posteriormente a sua atribuição a um ou mais membros da equipa. Também é possível criar "*issues*", sendo esta a forma de criar os casos de utilização a implementar na aplicação.



De forma a organizar todas as tarefas previstas para cada iteração, recorreu-se a um quadro do estilo *Kanban,* uma vez que permite visualizar as tarefas que já foram terminadas, as que estão em execução e as que ainda não foram iniciadas, havendo sempre a noção do trabalho que está a ser desenvolvido.



De forma a mostrar a vista geral do planeamento feito para cada iteração, utilizou-se uma tabela que apresenta todas as tarefas propostas para a iteração, sendo esta uma forma de complementar a gestão do projeto já feito noutras ferramentas do gestor de *backlog*.



fi

1.2 Gestão de código

Para a gestão do código na parte da implementação, escolheu-se o fluxo de trabalho do **Git**, tendo cada colaborador um *branch* e a necessidade de fazer *pull request* sempre que seja necessário submeter código.

Esta forma de gestão de código tem várias vantagens:

1. Isolamento de mudanças: Ao criar um *branch* separado para cada contribuidor, cada pessoa pode trabalhar nas suas alterações sem interferir no trabalho dos outros. Isso permite que os desenvolvedores experimentem livremente, façam correções e adicionem recursos sem afetar a base de código principal.

2. Colaboração simplificada: Os *branches* individuais facilitam a colaboração entre os membros da equipa. Cada contribuidor pode trabalhar em seu próprio branch e, quando estiverem prontos para incorporar as suas alterações, podem enviar um *pull request*.

3. Revisão de código: O uso de *pull requests* facilita a revisão de código por outros membros da equipa. Quando um contribuidor cria um *pull request*, os restantes membros podem analisar as alterações, fazer comentários, sugerir melhorias e identificar possíveis problemas antes de submeter as alterações no *branch* principal. Isso promove a qualidade do código e ajuda a evitar erros ou problemas de lógica.

4. Registro de alterações: O **Git** mantém um histórico completo de todas as alterações e discussões realizadas em cada *pull request*. Isso permite que os desenvolvedores visitem alterações anteriores, entendam as decisões tomadas e analisem o progresso do trabalho ao longo do tempo.

5. Gerir conflitos: Ao trabalhar em *branches* separados, é menos provável que ocorram conflitos diretos entre as alterações dos contribuidores. No entanto, se houver conflitos, o **Git** fornece ferramentas para resolvê-los de forma eficiente durante o *merge* dos *branches*.

Em suma, o uso do fluxo de trabalho do **Git** com um *branch* para cada contribuidor e o uso de *pull requests* promovem a colaboração, a revisão de código e a qualidade do código num projeto. Isso ajuda a evitar conflitos e melhora a gestão de alterações no código.