Por que otimizar a validação de formulários?

1. Providencia feedback imediato.
2. Salva recursos no servidor para reduzir o número de requisições para enviar um formulário.

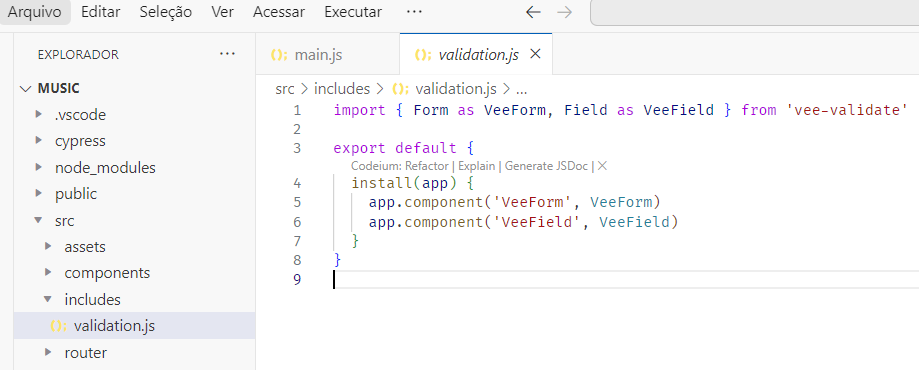
**SETTING UP FORM VALIDATION**

npm i vee-validate

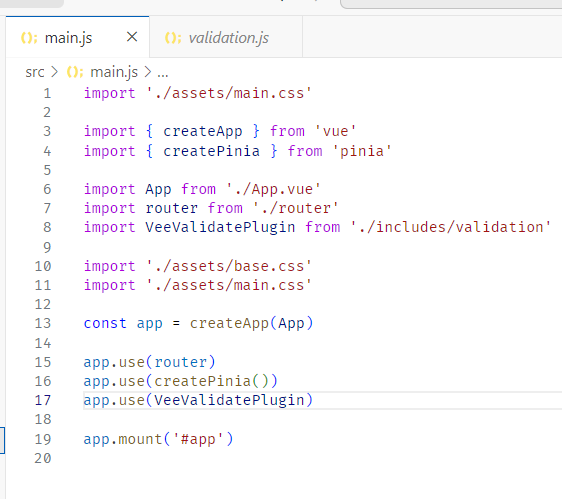
**REGISTERING A PLUGIN**

Existem duas formas de registrar um plugin: localmente ou globalmente.

Nesse caso, decidimos por registrar globalmente, eu acho rsrs:

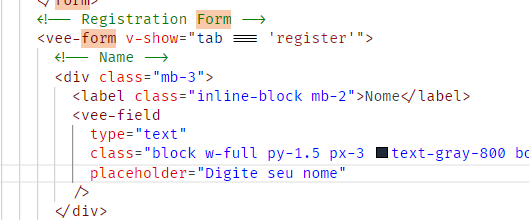


E para usá-lo:



**VALIDATION COMPONENTS**

Em seguida, precisamos ir até à form criada e substituir a tag <form></form> por <vee-form></vee-form>, e o campo que queremos validar por <vee-field>. Por exemplo, se tivermos um <input>, a tag será substituída por <vee-field>.



Por padrão, a vee-field gera um campo do tipo input. Se quiséssemos renderizar uma tag diferente, como um select, por exemplo, usaríamos a propriedade as:

<vee-field as=”select” type=”text” class…

O processo de validação é o seguinte:

1. Usar o componente <Field>, em nosso caso, vee-field.
2. Atribuir um name para o input.
3. Adicionar as regras.
4. Tratar os erros.

**DEFINING RULES**

É necessário instalar o vee-validate/rules para continuar:

npm install @vee-validate/rules

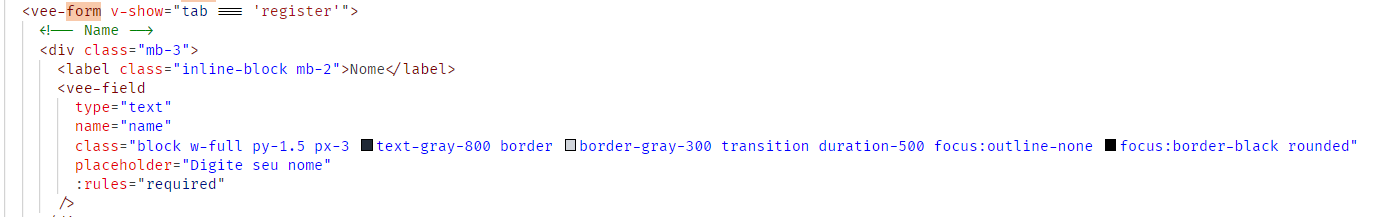
O próximo passo é registrar a rule desejada



Perceba que adicionei um novo import para required e também a defineRule.

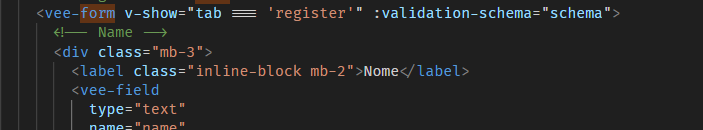
O passo seguinte é aplicar a regra, para isso, existem duas formas:

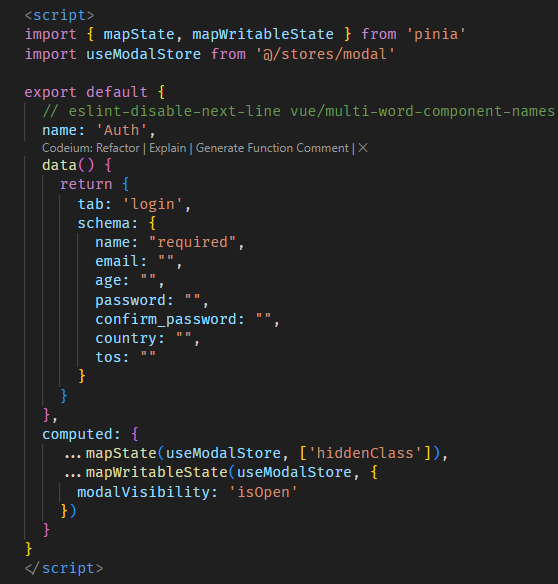
1) adicionar um bind no campo do formulário como o nome da regra:



**ADDITIONAL RULES**

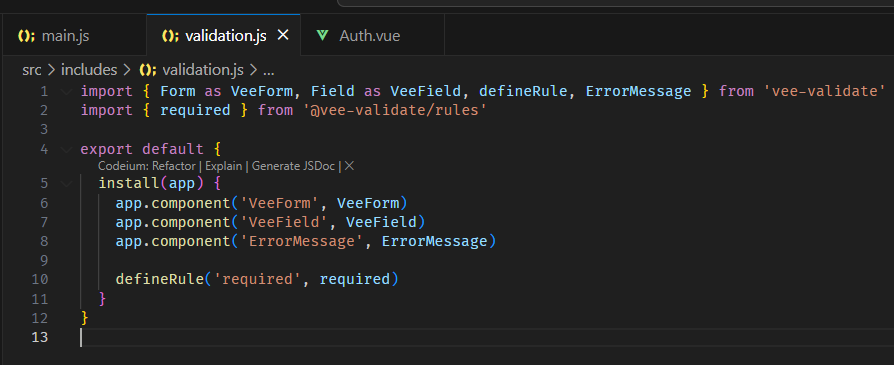
2) adicionar o bind validation-schema



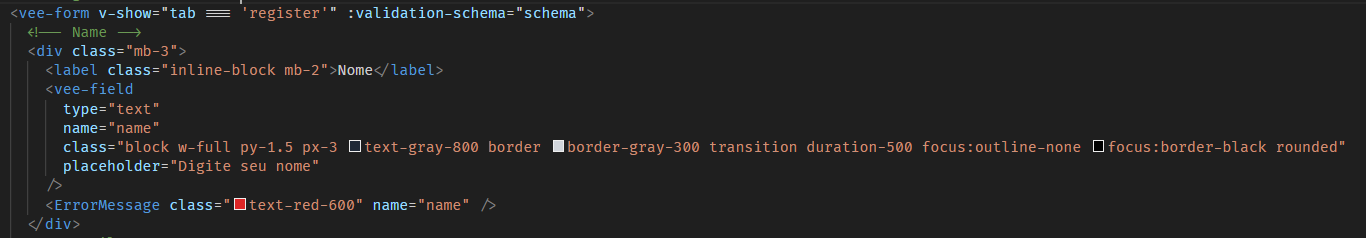


Por fim, resta fazer o tratamento de erros. Se o input é inválido, vamos querer providenciar um feedback para o usuário:

Vamos considerar um tratamento de erro para o campo de nome:



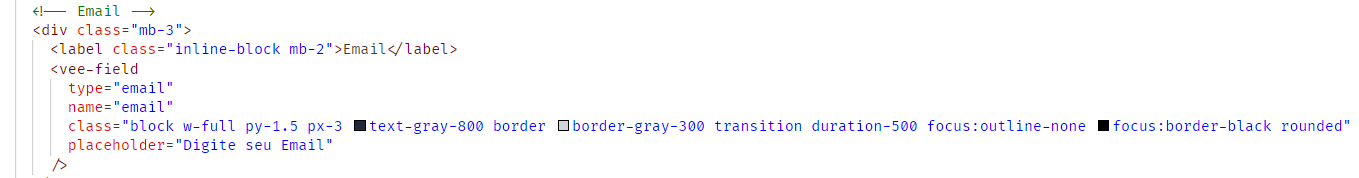
No código acima, importamos o ErrorMessage. O passo seguinte é colocar na na div onde queremos fazer o tratamento de dados:



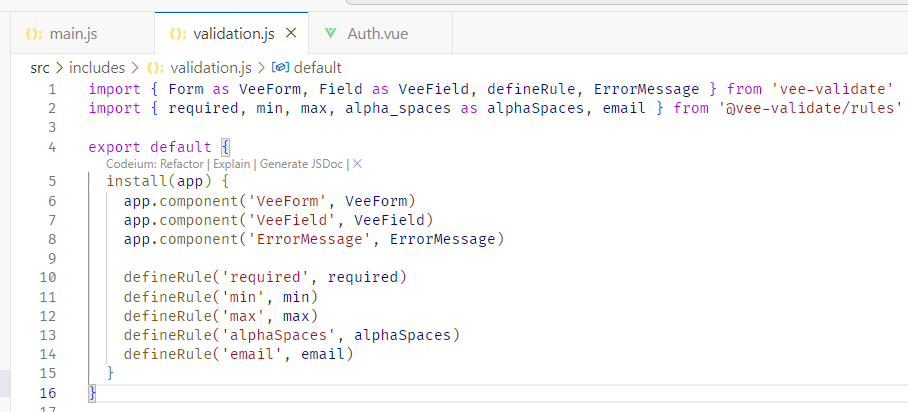
Perceba que além de personalizar o texto do erro com a class, estamos passando uma propriedade name que é correspondente ao input, veja que o name de ambos é correspondente. Há aqui dois detalhes importantes: a posição do componente ErrorMessage é onde exatamente iremos exibir a mensagem de erro caso não passar na validação, e o outro detalhe é referente ao name, que deve estar atrelado ao componente que não será validado para que a mensagem de erro seja exibida.

**VALIDATING E-MAIL**

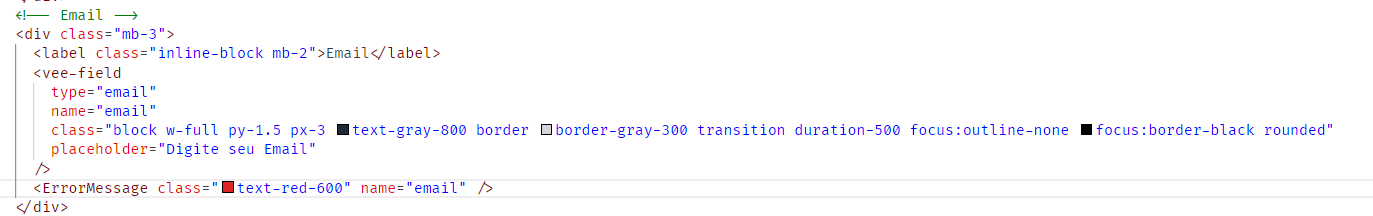
Para o e-mail remos fazer quase os mesmos passos que o campo nome. Primeiro, vamos adicionar um vee-field e um name:



Logo em seguida, vamos importar essa rule no nosso arquivo de validation.js e defini-la:

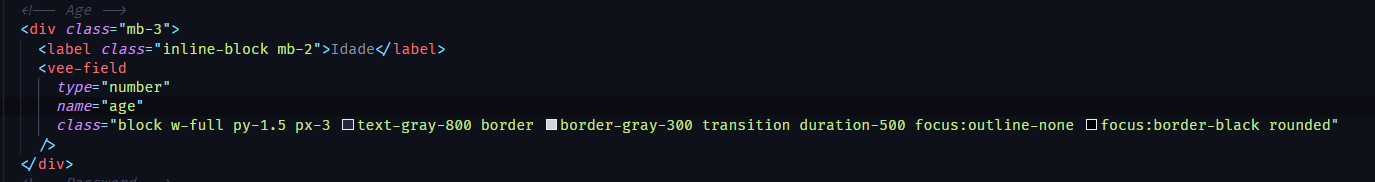


Por fim, tudo o que precisamos fazer é adicionar o tratamento de erro com a ErrorMessage component:

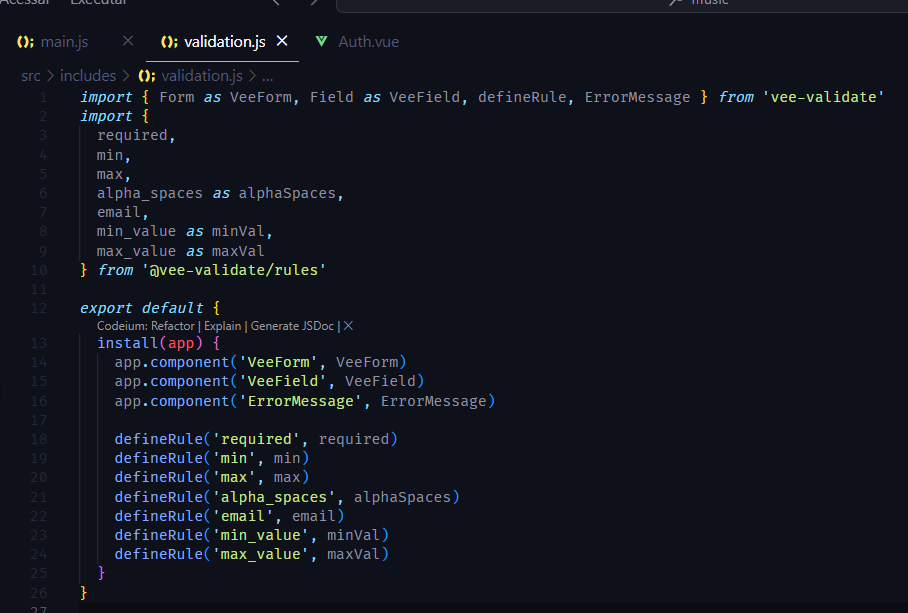


**VALIDATING NUMBERS**

O passo a passo é similar ao campo de e-mail e nome, primeiro atualizamos a tag input para vee-field e adicionamos um name.

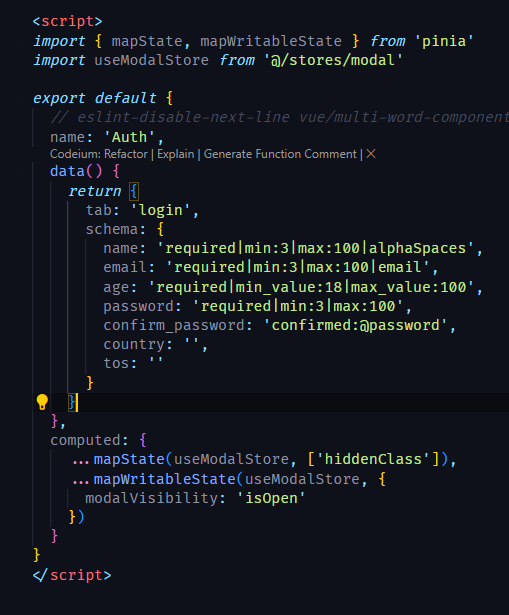


Agora, no arquivo validation.js temos uma diferença com relação às validações dos demais campos, pois aqui não irá funcionar como queremos o min e o max. Essa validação funciona para os campos de texto porque o min e o max se referem ao tamanho da string. Para o valor de um número, precisamos de uma abordagem diferente, importaremos as rules min\_value e max\_value:

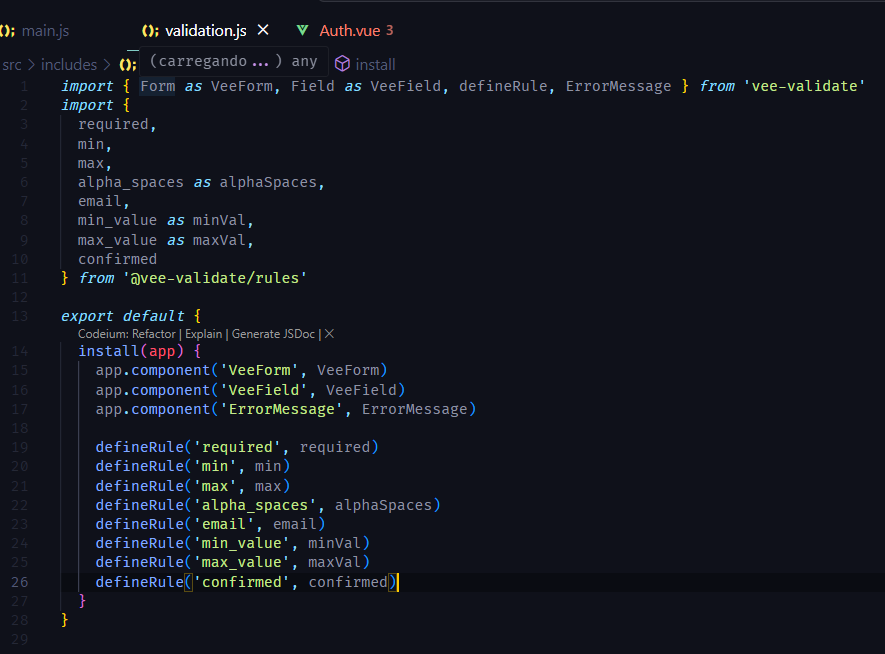


**VALIDATING PASSWORDS**

Validar as senhas segue o mesmo princípio: substituir o nome da tag input para o vee-field e adicionar um name. O passo seguinte é adicionar as rules, perceba que os campos de password e confirm\_password possuem rules diferentes, isso porque não precisamos validar os dois campos com as mesmas rules: o campo de password sim precisa daquelas rules, mas o de confirm\_password precisa somente fazer um match com o campo password, e para isso utilizamos nele a rule de confirmed, onde o seu valor é o nome do campo que queremos comparar, com o prefixo @.

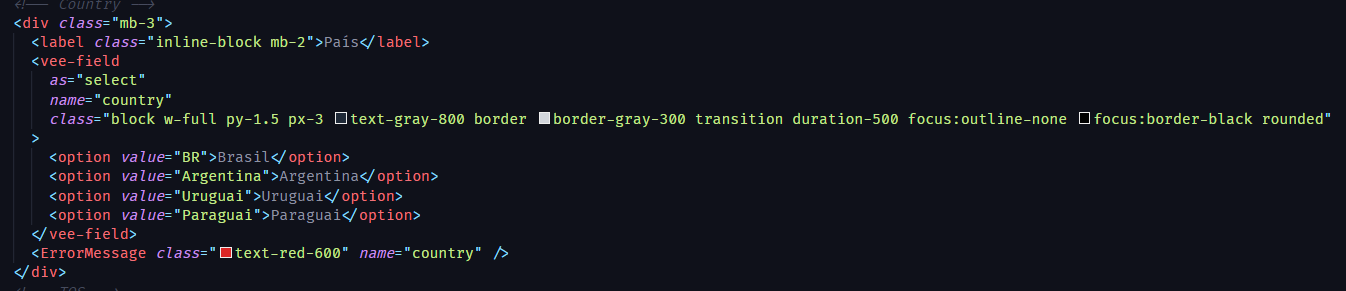


Para que elas funcionem, é necessário que essas rules estejam devidamente importadas no arquivo de validation.js:



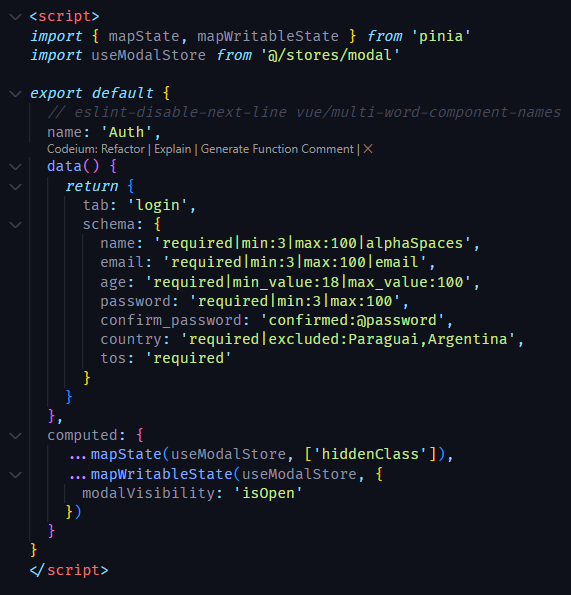
**DROPDOWN AND CHECKBOX FIELDS**

O dropdown precisa ser configurado da seguinte forma:

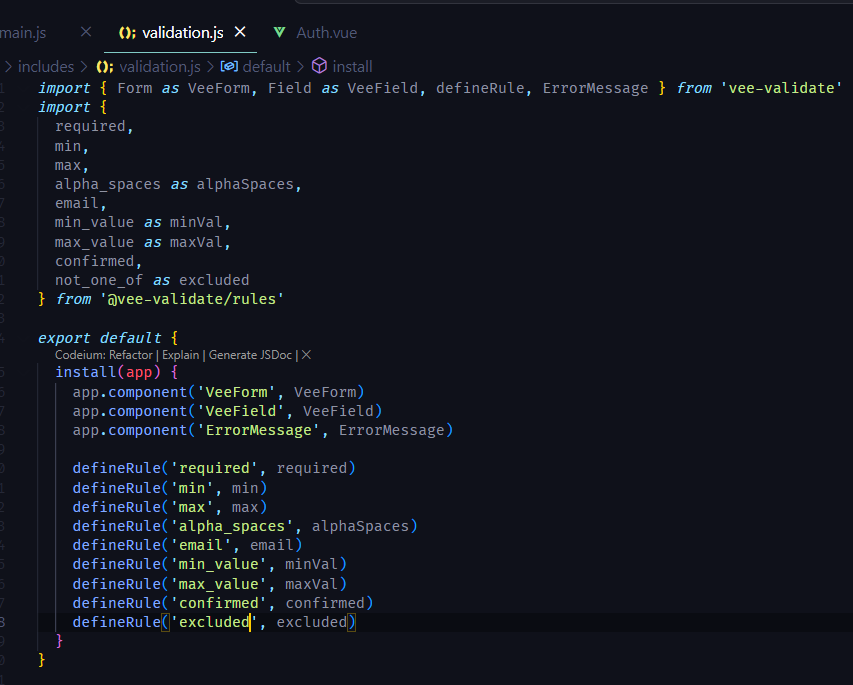
****

Perceba que a tag de select foi substituída por vee-field e foi adicionado um as=’select’ para formatar digamos assim aquela tag input em uma select. Além disso, ela ganha uma tag de fechamento depois das opções. Além disso, acrescentamos um name à tag.

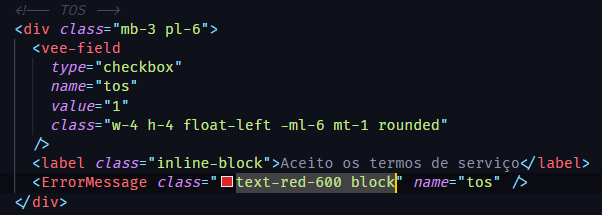
Para o dropdown, acrescentamos somente uma rule de required e também uma rule para o caso de que alguma das opções por algum motivo não deva ser marcada:



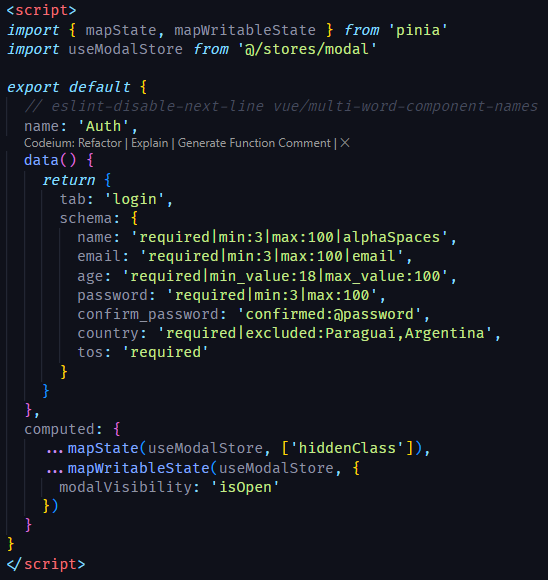
Essa rule de excluded, ela foi importada no arquivo de validation.js:



Por fim, a checkbox foi configurada dessa forma:

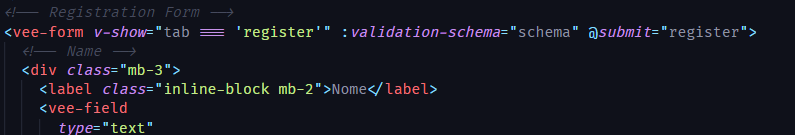


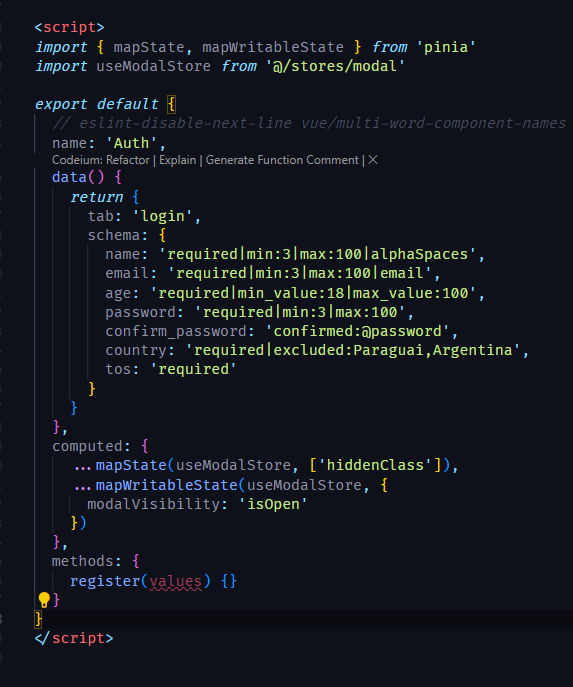
Substituído a tag original para vee-field e adicionado o name. A checkbox recebeu somente a rule de required:



**VALIDATING THE FORM**

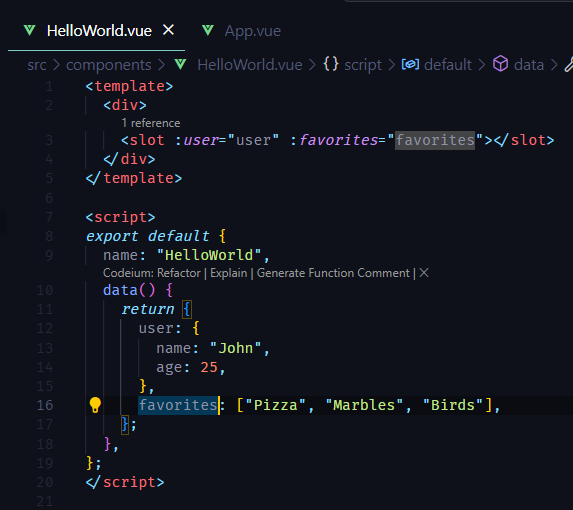
As validações citadas até o momento só irão acontecer caso ocorra um evento de mudança nos campos mencionados. Precisamos adicionar uma validação para o formulário não ser enviado até que todos os campos que forem required estejam devidamente preenchidos. Para isso, podemos adicionar na tag que abre o formulário, uma função que faça essa validação para nós:



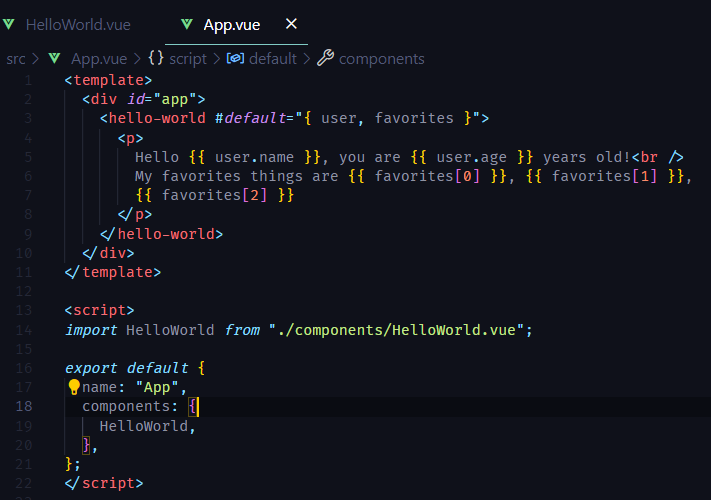


**SIDEBAR: SLOT PROPERTIES**

Funciona da seguinte forma, vamos considerar o componente Hello World abaixo:



É um componente simples, possui 2 tipos de data, user e favorites. Essas propriedades podem ser acessadas no componente que importa o helloworld se enviarmos por meio de slot. Perceba que montamos dois binds no código acima.

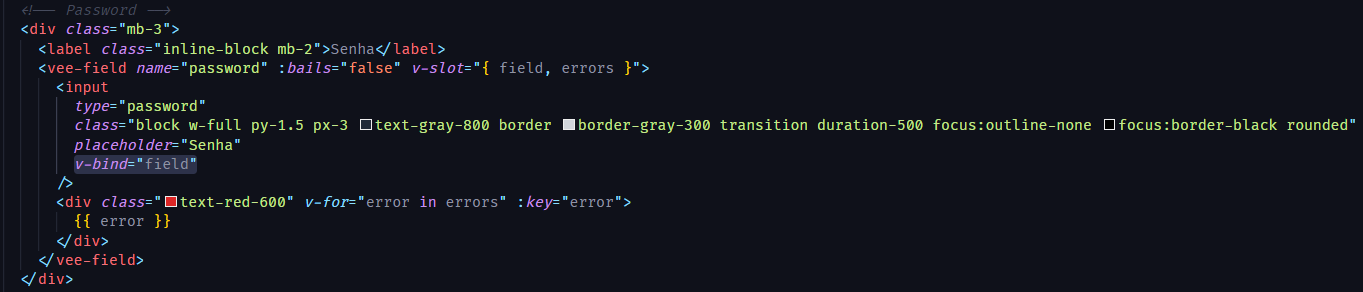


Neste componente que está fazendo a importação de helloworld, estamos apenas declarando ao vue que iremos usar esse componente e no momento de usá-lo, colocamos na tag do componente um #default passando as propriedades data que queremos usar e acessamos utilizando o ponto para pegar internamente os dados.

**RENDERING MULTIPLE ERROR MESSAGES**

A propriedade :bails irá dizer ao componente para não usar a estratégia de saída rápida, o que significa que cada regra irá verificar se uma regra anterior foi quebrada ou não. Para exibir múltiplos erros só existe uma forma:

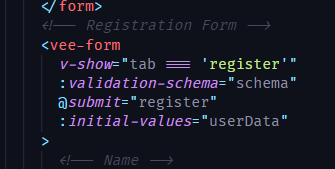
Vamos adicionar uma tag de fechamento se o componente tiver só uma tag fazendo a marcação de início e fim. Dentro dessa tag de vee-field vamos adicionar um input e passar as propriedades de class, type e password para esse novo input:



Perceba que temos uma div específica para os erros que serão iterados para os casos em que ocorram mais de um erro, exibir todos os que ocorreram.

**DEFAULT VALUES**

Para configurar valores padrões, basta fazer o seguinte: ir até a tag de abertura do formulário e adicionar ali um bind de :initial-values, veja:



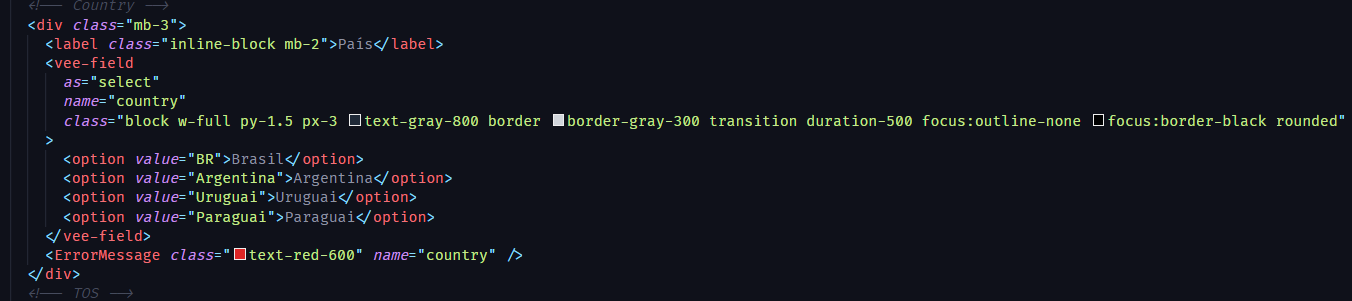
O userData é uma função onde podemos definir os valores padrões para os campos desejados:



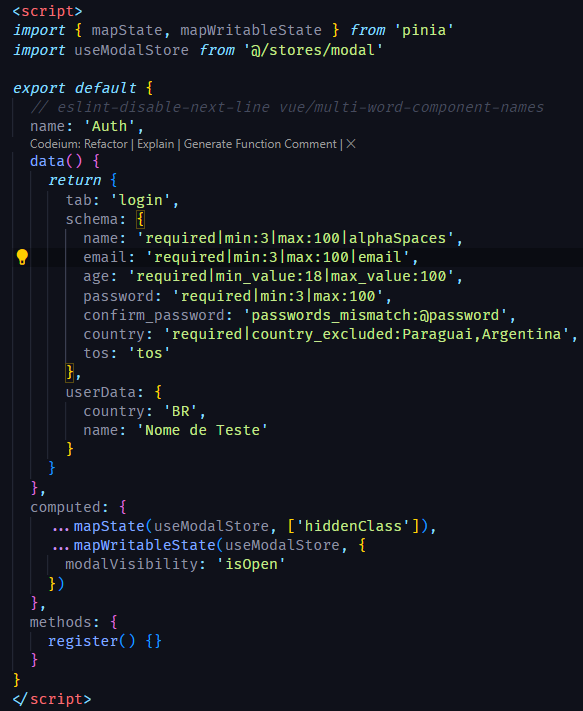
O detalhe importante que é preciso se atentar é o seguinte: dentro do data, colocamos o que estiver no value da option, por exemplo, ali o country tem o valor de BR, que é exatamente o valor do option do Brasil lá no Dropdown. Se colocar o nome da opção, não vai funcionar.

**CUSTOM ERROR MESSAGES**

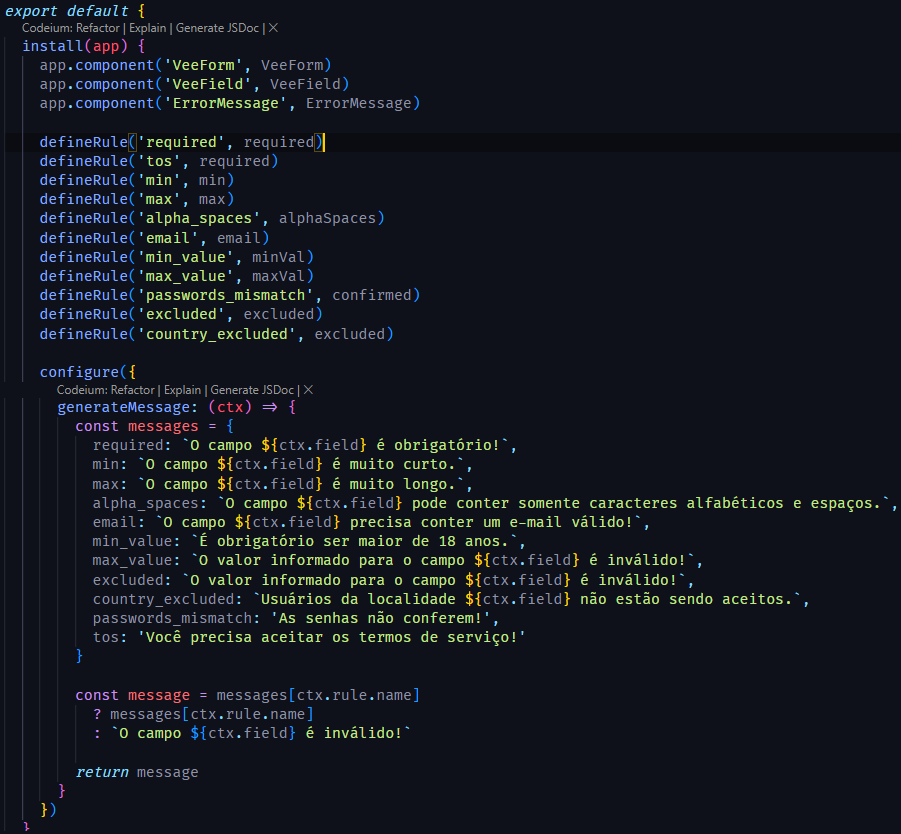
É possível criar mensagens de erro personalizadas, acompanhe o fluxo para relembrar:



Aqui temos um select de país, que tem o name ‘country’. Lá na função de export default, esse campo está recebendo algumas regras:



Uma das regras é o country\_excluded: Paraguai, Argentina. Se formos recorrer ao arquivo de validations.js, vamos conseguir consultar se há alguma mensagem personalizada para essa rule:



Perceba que em defineRule, estamos definindo uma rule personalizada chamada country\_excluded, e aplicando para ela as regras de excluded (que é uma rule nativa). Na função configure, existe ali uma função generateMessage, passando como parâmetro ctx que é o context, onde ali dentro existe uma constante que atribui mensagens à algumas rules, esse é o local onde podemos personalizar as mensagens das rules. Ainda, existe uma constante message que vai fazer um ternário com as mensagens: se existir uma mensagem personalizada, vai receber ela, senão, recebe a mensagem padrão da rule.

**VALIDATING TRIGGERS**

Quando um input é validado?

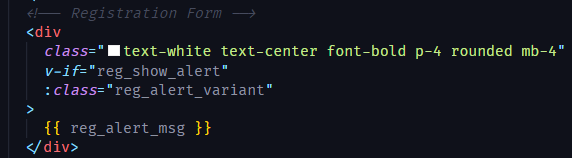
1. Após o evento de change.
2. Se a diretiva v-model é aplicada para um input, quando o model é mudado.
3. Após o evento de blur, ou seja, quando o input perde o foco.
4. Após o envio do formulário.

Existem esses 4 tipos de validação:



**SHOWING ALERTS**

A exibição de alertas funciona da seguinte forma: o primeiro passo é criar uma div onde irão ser exibidos os alerts:



O v-if criado é para o vue saber se deve ou não mostrar essa div de alertas. O :class é a classe que será exibida quando mostrar. Isso se refere à cor do background, verde para sucesso e vermelho para erro. O reg\_alert\_msg é o conteúdo da mensagem que será exibido. As duas últimas informações estão sendo configuradas no export default:



\*A validação de um formulário de registro e de login é basicamente a mesma coisa. O que varia são os campos que, em geral o registro possui em maior quantidade e tipo.