Interface Pessoa-Máquina

Licenciatura em Engenharia Informática

Ficha Prática #09

Rafael Braga
d13414@di.uminho.pt

Daniel Murta
d6203@di.uminho.pt

José Creissac Campos
jose.campos@di.uminho.pt

(v. 2024)

Conteúdo

1 Objetivos

1. Praticar a a criação de um estado global numa aplicação Vue através da biblioteca Pinia.

Para ilustrar a utilização de Pinia, esta ficha propõe a implementação de uma nova versão da aplicação apresentada na Ficha 8 (ver Figura 1).

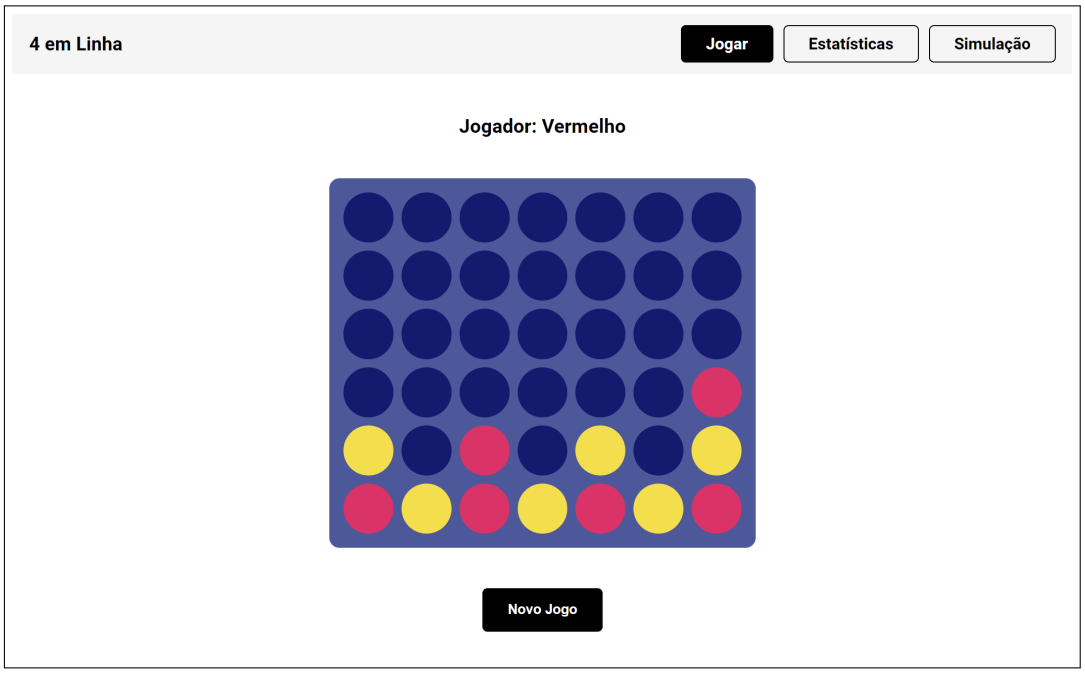

Figura 1: O jogo 4 em linha

Esta nova versão deverá melhorar a componente de simulação proposta na ficha anterior, que permitia realizar a animação das jogadas efetuadas durante o último jogo realizado na aplicação. No entanto, na versão anterior a simulação foi construída à custa da ferramenta *json-server*, onde se guardava o jogador inicial (amarelo ou vermelho) do último jogo realizado, assim como as jogadas efetuadas, para poderem posteriormente ser animadas durante a simulação.

Nesta ficha propõe-se uma solução que não dependa de um *backend* para realizar esta componente de simulação (reduzindo assim a carga que a aplicação coloca neste). Para isso pretende-se que se construa uma *store* com recurso à biblioteca *Pinia*, onde se poderá manter um estado partilhado que pode ser acedido pelos diversos componentes da aplicação.

2 Gestão de Estado com Pinia

Na construção de aplicações Vue.js, a partilha de informação entre componentes pode rapidamente tornar-se uma fonte de complexidade. À medida que a aplicação cresce, manter o fluxo de dados entre componentes pai, filhos e irmãos, torna-se difícil de gerir. A necessidade de utilizar propagação de eventos em cadeia, passagem de *props* entre múltiplos níveis e a dificuldade em manter o estado sincronizado entre componentes não relacionados, origina um código mais propenso a erros e uma maior dificuldade em manter e escalar as aplicações.

Pinia[1] é uma biblioteca de gestão de estado para Vue.js que permite definir *stores* para guardar estado de forma centralizada, e torná-las acessíveis em toda a aplicação, mantendo o código limpo e reativo. Foi criada como uma alternativa mais simples e flexível ao Vuex[2] e é a solução recomendada para Vue3.

As principais características de Pinia incluem:

- Reatividade: o estado das *stores* é reativo, permitindo a atualização em tempo real da interface pessoa-máquina.
- Modularidade: Permite a criação de múltiplas *stores*, cada uma responsável por uma parte específica do estado da aplicação.
- Simplicidade: foi desenvolvida para ser simples de utilizar.

Para utilizar Pinia num projeto é necessário indicar essa opção ao criá-lo (ver Figura 2). Para adicionar Pinia a um projeto existente pode usar-se o comando:

```
% npm install pinia@latest --save-dev
```

Estando o projeto configurado, para criar uma *store* em Pinia, começa-se por definir o seu estado, ações e *getters* (propriedades derivadas). É então possível importar a *store* nos componentes e aceder ao estado, *getters* e ações. Exemplos podem ser encontrados nos *slides* das aulas teóricas e no exemplo fornecido.

3 Exercícios

Resolva os seguintes exercícios.

[1] https://pinia.vuejs.org/, vsitado em 15/04,2024.

[2] https://vuex.vuejs.org/, vsitado em 15/04,2024.

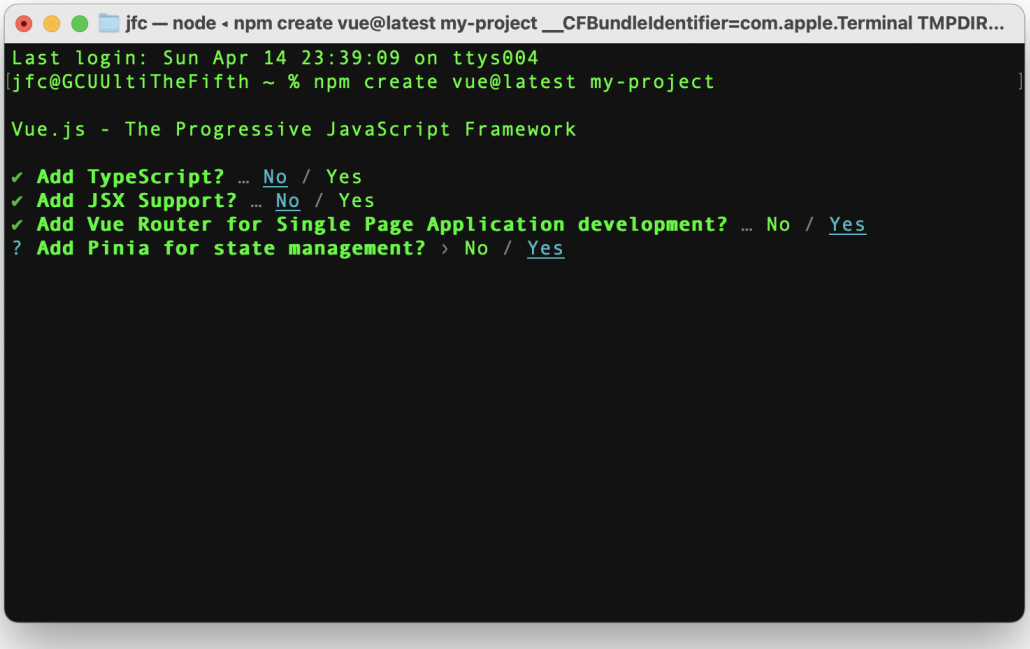

Figura 2: Creating a project with Pinia

3.1 Criação de um estado global de simulação para o Jogo 4 em linha

Com esta ficha, é fornecida uma base de trabalho para a implementação da capacidade de simulação mencionada. Esta base, contém uma implementação da aplicação desenvolvida na ficha anterior (diretoria `4inarow`), e uma implementação parcial da sua componente de simulação. Além disso, é fornecido um *backend*, implementado através da ferramenta *json-server* (diretoria `backend`). O *backend* permite:

- Consultar/atualizar as estatísticas do jogo.
- Consultar/guardar o histórico dos jogos.

Para consultar todas as funcionalidades da aplicação deverá instalar e correr o *backend*.

Tomando como base a implementação parcial da componente de simulação aplicação fornecida com esta ficha, resolva os seguintes exercícios.

1. Complete a *store* relativa à componente de simulação que está presente no ficheiro "simulationStore.js" (pasta "stores´´). Para isso, declare o estado da *store* e acrescente duas variáveis a esse estado:

- `startPlayer` - que indica o primeiro jogador (amarelo ou vermelho) que jogou no último jogo realizado. Inicialmente esta variável deverá ser nula.
- `plays` - que corresponde a um *array* de índices de colunas onde foram efetuadas as jogadas do último jogo.

2. Acrescente agora à *store* as seguintes ações:

   - `setStartPlayer` - que recebe um jogador como argumento e atualiza a variável `startPlayer`.
   - `addPlay` - que adiciona uma jogada ao *array* de jogadas.

3. No componente `OJogo`, sempre que efetuar uma jogada (método `jogar`), atualize o estado global da simulação. Para isso atualize a informação relativa ao jogador que começou o jogo (lembre-se que inicialmente a *store* tem o jogador a `null`) e a adicione a coluna jogada ao *array* de jogadas. Ainda neste componente, reverta o estado da *store* para o seu estado inicial, sempre que se faça `reset()`.

4. Acrescente à *store* o *getter* `containsSimulation` que deverá devolver verdadeiro caso exista um jogador inicial e o *array* de jogadas não esteja vazio.

5. No componente `Simulation` complete o código do método `simulate()`. O objetivo é ir buscar à store as informações para a simulação. Para conseguir isso, siga as instruções fornecidas em comentário no método.