EXAMEN: Sistemes i Tecnologies Web Juny 2024

Niu: Nom:

Permutació A

☞ GJeans

Després de pujar el codi a moixero.uab.cat ompliu el següent requadre.

Entrega Electrònica

The SHA1 checksum of the received file is:

.....

Time stamp is:

.....

Guardeu-vos una copia de l'arxiu que entregueu.

Guia de correció:

La nota serà...	Quan...	Guia de puntuació
De 0 a 5 punts	Quan no funciona tot i hi ha errors de concepte *greus* en algun dels següents elements clau: callbacks, clausures, classes, herència, promises, i els diversos elements de vue (reactivitat, directives, events, props, components, ...).	Es parteix d'un 5 i es resta 1 punt per cada error de concepte.
De 5 a 8 punts	Quan no s'aconsegueix fer funcionar tot l'examen i no hi ha errors de concepte greus.	Es parteix d'un 8 i es resten 0.25 punts per cada error.
De 8 a 10 punts	L'examen passa tots els tests i feu entrega electrònica.	Teniu un 10. Us l'heu guanyat.

Instruccions

Seguiu les següents instruccions per a arrencar la màquina del laboratori i importar l'esquelet del projecte.

- Arrenqueu la màquina si no l'heu arrencada abans i seleccioneu la partició de Linux (user: examen i passwd: examen).
- Obriu una consola: Applications → Terminal.
- Descarregueu-vos l'esquelet del projecte executant la comanda:

  ```
  wget https://moixero.uab.cat/ExamenSTW.7z
  ```

- Descomprimiu el fitxer.

  ```
  7z x ExamenSTW.7z
  ```

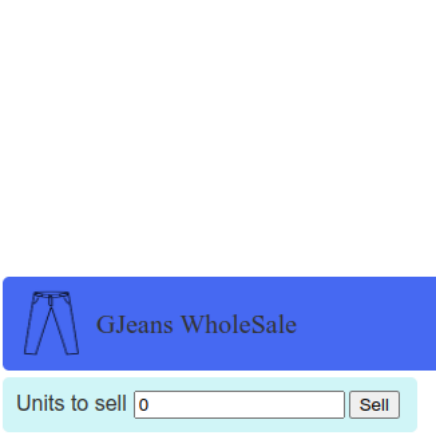
- L'esquelet el teniu dins el directori **stw**. Feu l'examen (podeu fer servir el mateix terminal que ja teniu obert, i obrir el directori **stw** amb el Visual Studio Code).
- Per executar l'aplicació:
  1. el client: **`npm run dev`**
  2. el servidor: Posicioneu-vos al directori **`src`** i executeu **`node exam.js`** o bé **`nodemon exam.js`**.
- Nota: Si utilitzeu Chrome, veureu que s'obre en mode incògnit i no hi ha disponible l'addon de Vue. Obriu un nou Chrome (Ctrl+N). La nova finestra ja no és incògnit i el addon de Vue es pot utilitzar.
- Un cop hagueu acabat de desenvolupar el projecte, haureu d'entregar el vostre codi electrònicament. **Si us funciona tota l'aplicació, aviseu-nos abans d'entregar electrònicament.**
- Per entregar electrònicament, creeu un zip de la següent manera:
  Posicioneu-vos en el directori **`src`**

  ```
  7z a sol.zip exam.js App.vue components/SellForm.vue components/Shop.vue
  ```

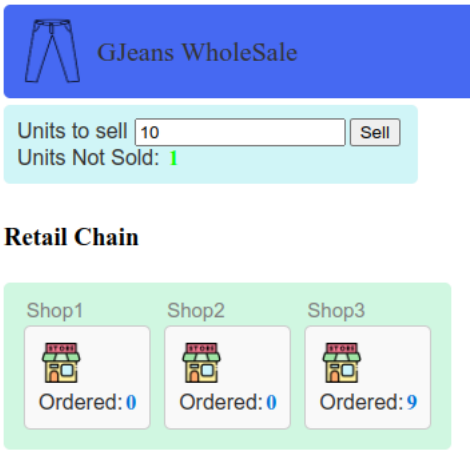
- Comproveu el contingut de l'arxiu que entregareu (obriu l'arxiu i mireu el contingut dels arxius que hi ha a dintre).
- Un cop sapigueu segur que voleu entregar aquest arxiu, pugeu-lo a: `https://moixero.uab.cat/`.
- Anoteu els dos valors (el checksum i el timestamp) a l'examen en paper i entregueu l'examen en paper.
- **Quan acabeu no sortiu de la sessió i no pareu la màquina!!**

Context

Cal que desenvolupeu el front i el back-end d'un portal per fer vendes a l'engròs de pantalons. A través d'aquest portal es podrà ofertar un stock de pantalons (Figura 1 (a)). El back-end rebrà l'estock de pantalons i els oferirà a diferents botigues les quals compraran una quantitat aleatòria. Quan totes les botigues hagin decidit la quantitat de producte que compren es mostrarà la distribució final de compra en el front-end (Figura 1 (b)).



(a) Pàgina principal del portal de WholeSale



(b) Resultat d'ofertar deu pantalons. Shop1 i Shop2 no han comprat cap pantaló. La Shop3 n'ha comprat 9. Ens ha quedat un estoc d'un pantaló sense vendre.

Figura 1: Portal de vendes a l'engròs.

Podeu arrencar aquest portal executant:

1. el client: **`npm run dev`**
2. el servidor: Posicioneu-vos al directori **`src`** i executeu **`node exam.js`** o bé **`nodemon exam.js`**.

Veureu la vista de la figura 2.

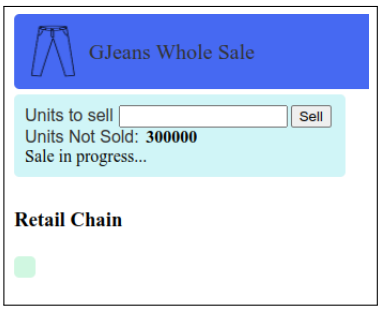


Figura 2: Vista de l'esquelet.

Exercici frontend

Cal que completeu el **`RootComponent`** (**`App.vue`**) i els components **`SellForm`** i **`Shop`**.
A continuació us indiquem les especificacions dels components.

RootComponent (App.vue)

Aquest component inclou el component **`SellForm`** de qui obté la distribució de compres. A partir d'aquesta distribució utilitza el component **`Shop`** per mostrar les unitats venudes a cada botiga.

Especificacions

- Centralitza la distribució de compres en la variable reactiva **`orders`**.
- Les compres (**`orders`**) es representen mitjançant un objecte JSON com el següent:

```
{remainingStock:1, details:[{id:"Shop1",ordered:9},{id:"Shop2",ordered:0}, {id:"Shop3",ordered:0}]}
```

  A on les opcions:

  remainingStock: Indica les unitats que no s'han venut.

  details: És un array de diccionaris. Cada diccionari està associat a una botiga. Per cada botiga mostrem el seu identificador i el nombre d'unitats que ha comprat. Noteu que pot ser que una botiga no compri cap unitat.

- Les compres (**`orders`**) les actualitza el component **`SellForm`** que utilitzarem de la següent manera:

```
<SellForm v-model="orders" />
```

- Itera sobre la llista de **`orders.details`** i utilitza el component **`Shop`** per mostrar la informació de compra de cada botiga d'aquesta llista.

Component: SellForm

Aquest component inclou un input per especificar les unitats a vendre i un botó per executar la venda (Figura 3 (a)). En executar la venda, fins que el servidor no ens torni la distribució final de la venda entre les diferents botigues mostrem un missatge (Figura 3 (b)). Quan el servidor ens torna la distribució de la compra mostrem el nombre d'unitats que no hem venut (Figures 3 (c) i (d)) o bé cap missatge si ho hem venut tot.

(a) Entrada d'unitats a vendre (b) Operació de venda en procés (c) Han quedat deu productes sense vendre (d) Ha quedat un producte sense vendre

Figura 3: Vistes del component **`SellForm`**.

Especificacions

- Rep, a través de la directiva **`v-model`**, el diccionari (**`orders`**) per a actualitzar-ho amb les compres. Recordeu que els dos tags següents són equivalents:

```
<Component v-model="data" />
<Component :modelValue="data" @update:modelValue="x => data = x" />
```

- En clicar el botó de **`sell`** es fa una petició al servidor per a que executi la venda de les unitats especificades en el input:

```
fetch(`http://localhost:3001/wholesale/10`).then(res => res.json()).then(json => {...})
```

  Noteu que **`/wholesale`** és un entry point del server i **`/10`** és el nombre d'unitats de pantalons a vendre.

- **Amb la resposta que rebrem del server actualitzarem les `orders` del `RootComponent`.**
- Un cop clicat el botó de **`sell`** i mentre espera la resposta del servidor, mostra el missatge *"Sale in progress..."* (Figura 3 (b)). Quan rebem la resposta del server no renderitzem aquest missatge.
- Si no s'han venut totes les unitats mostrem el missatge: *"Units Not Sold"* indicant les unitats que no s'han venut amb el següent codi de colors: **`vermell`** si no s'ha venut res, **`taronja`** si s'ha venut menys del 50%, i **`verd`** si s'ha venut més del 50%. Si voleu podeu adaptar el següent *snipped* de codi per determinar el color:

```
notSold == unitsToSell ? 'red' : (notsold < (unitsToSell*0.5)) ? 'lime' : 'orange'
```

Component: Shop

Aquest component mostra els detalls de compra d'una botiga especificats pel diccionari:

```
{id:"Shop3",ordered:6}
```

a on el camp **`id`** correspon a l'identificador de la botiga i **`ordered`** correspon al nombre d'unitats que la botiga ha comprat, tal i com es mostra a la figura 4.

Figura 4: Vista del component **`Shop.vue`**

Especificacions

– El component rep com a paràmetre l'objecte amb els detalls de la compra d'una botiga: `{id:"Shop3",ordered:6}`.

Test FrontEnd

- Us donem un server dummy que, passats dos segons, sempre retorna la següent distribució de compres:

```
{remainingStock:1, details:[{id:"Shop1",ordered:8},{id:"Shop2",ordered:0}, {id:"Shop3",ordered:1}]}
```

Notes

– No es pot utilitzar **`async/await`**.

Exercici backend

El backend consta d'un mòdul que actúa com a ditribuidor dels pantalons entre un array de botigues que té definides (*hardcoded*). Cada botiga és un objecte de tipus **`Shop`** que ja us donem totalment programat. El mòdul distribuidor (**`wholeSaleModule`**) oferirà les unitats a la venda que rebem del client a les diferents botigues. Cada botiga comparà un nombre aleatòri d'unitats o bé cap. Finalment el mòdul encapsularà la distribució de la compra de les botigues en un diccionari que retornarà al client.

Especificacions de la classe Shop

Atenció!! Aquesta classe ja us la donem acabada.

– El constructor rep com a paràmetre el **`id`** de la botiga i un **`timeout`**. Aquest timeout és el temps que té per decidir quantes unitats compra.

– Mètodes de la classe:

toPlainObject retorna un objecte que encapsula el **`id`** de la botiga i el nombre d'unitats que ha decidit comprar: `{id: 'shop3', ordered: 6}`

offer(units):promise Mètode que rep les unitats disponibles a la venda i retorna una promesa que, passat un timout, **resol al nombre d'unitats que ha decidit comprar** o bé **rejecta a res (`reject()`) si ha decidit no comprar res**.

Especificacions del mòdul wholeSaleModule

– Aquest mòdul el desenvolupareu seguint el patró *module pattern*. Aquí teniu un exemple.

```
const testModule = (()=>{
  let _a = 1;
  const increase = () => { _a++; },
  const value = () => { return _a; }
  return {
    increase,
    value,
  };
})();
```

– Conté una llista estàtica amb tres botigues:

```
let shops = [new Shop("Shop1", 1000), new Shop("Shop2", 700), new Shop("Shop3", 200)]
```

- **Funció getDetails**: Funció privada que us pot anar bé, és **opcional**. Itera la llista de botigues i retorna una llista amb la distribució de compra que han fet:

```
[{id:"Shop1",ordered:9},{id:"Shop2",ordered:0}, {id:"Shop3",ordered:0}]
```

  Us caldrà accedir al mètode **`toPlainObject`** de la classe **`Shop`**.

  `getDetails` L'utilitza la funció **`wholeSale`**, de manera que, un cop distribuïdes les unitats a les botigues, **`wholeSale`** utilitza aquesta funció per omplir el camp **`details`** de l'objecte que encapsula el resultat de la distribució de compra de les diferents botigues.

  Noteu que aquesta funció l'heu de cridar un cop sapigueu que totes les botigues ja han fet la compra.

- Funcions que exposa (retorna) el mòdul:

  wholeSale(units) : promise Funció que rep el nombre d'unitats especificat en el frontend i les ofereix a les botigues. Retorna una promesa que resol a l'objecte que conté el nombre d'unitats que no s'han venut i una llista amb el detall de compra de cada botiga:

```
{remainingStock:1, details:[{id:"Shop1",ordered:9},{id:"Shop2",ordered:0}, {id:"Shop3",ordered:0}]}
```

  Atenció!!! Hem d'oferir a la primera botiga totes les unitats disponibles, a la següent botiga totes les unitats disponbiles menys les que hagi comprat la botiga anterior (estoc remanent) i així successivament. Noteu que la funció `Shop::offer(units)` retorna una promesa que resol al nombre d'unitats que compra o rejecta si no compra res.

  Podeu programar aquesta funció de dues maneres diferents que us detallem a continuació. Trieu la que us vagi millor:

  1. **Opció Recursiva:**
     - Amb aquesta opció ens és igual el nombre de botigues que tinguem a la llista.
  2. **Opció Iterativa:**
     - Podeu tractar directament **`shops[0]`**, **`shops[1]`** i **`shops[2]`**.
     - Cal que utilitzeu una cadena de promeses. El següent codi no és una cadena de promeses i no s'acceptarà:

```
p.then(x1=>{p2.then(x2=>{p3.then(x3=>...).catch()}).catch()}).catch()
```

     - Pista: Refresqueu l'exercici 23 de problemes: *23. Fes un p.then(x => console.log(x))*...
     - Si opteu per a aquesta opció obligatòriament heu d'implementar la següent funció:

       stockUpdater(units): updateStock :
       - És una funció privada del mòdul **`wholeSaleModule`**, és a dir, que no l'exportem.
       - Aquesta funció rep el nombre d'unitats a vendre (estoc) del frontend i retorna una funció **`function(unitsSold): remainingStock`**
         - Aquesta funció retornada la podeu anomenar **`updateStock`** i com heu vist, rep com a paràmetre el nombre d'unitats que una botiga compra, actualitza l'estoc (estoc-=unitsSold) i retorna l'estoc actualitzat.
       - La funció **`stockUpdater`** implementa una **clausura**!

Especificacions del servidor web

- El servidor tindrà un endpoint:

  `/wholesale/:units` a on **`units`** correspon al nombre d'unitats a distribuir entre les diferents botigues del distribuidor. En rebre aquesta petició, el servidor executa la funció **`wholeSale(units)`** i retorna el resulat al client.

Test BackEnd

- Podeu provar el server directament des d'una finestra del navegador posant la url:

```
http://localhost:3001/wholesale/10
```

- El navegador ha de carregar un JSON com el següent:

```
{"remainingStock":1,
 "details":[{"id":"Shop1","ordered":9},{"id":"Shop2","ordered":0},{"id":"Shop3","ordered":0}]}
```


- Assegureu-vos que el nombre d'unitats inicial menys les unitats venudes és el valor de **`remainingStock`**.
- Podeu refrescar el navegador per fer una nova petició al servidor.

Notes

- No es pot utilitzar async/await
- **`Promise.resolve(0)`** retorna una promesa que immediatament resol a 0.
- **`Promise.reject(3)`** retorna una promesa que immediatament rejecta a 3.
- Exemple d'utilització de la funció **`map`**

```
const array1 = [1, 4, 9, 16];

// Pass a function to map
const map1 = array1.map((x) => x * 2);

console.log(map1);
// Expected output: Array [2, 8, 18, 32]
```

- Exemple d'utilització de la funció **`forEach`**:

```
const array1 = ['a', 'b', 'c'];

array1.forEach((element) => console.log(element));

// Expected output: "a"
// Expected output: "b"
// Expected output: "c"
```

Aquesta és una pàgina per fer esborranys. No la desgrapeu!

Aquesta és una pàgina per fer esborranys. No la desgrapeu!