Lizee / Software Engineering / Use-cases

Context

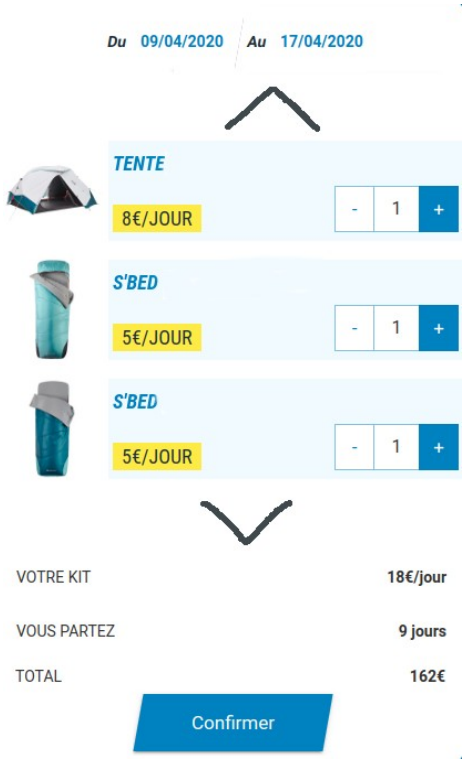
Adrenaline Heroes is a (fake) company that sells sportswear & outdoor equipment. They just signed a contract with Lizee to add a rental offering into their existing e-shop (which currently only *sells* products ; technically their current e-shop is a custom React frontend on top of Magento).

The rental offering aims at enabling customers to rent a subset of their product catalog at a daily rate. Some examples to provide context: trekking backpack (€4/day), tent (€4.5/day), sleeping bag (€2.5/day), sleeping mattress (€1.5/day), frontal headlamp (€0.5/day), etc.

There are discounts when renting longer: 10% for three or more days ; 20% > 7 days ; 30% > 14 days ; 50% > 21 days). For the sake of this use-case, delivery and returns are to/from parcel shops only.

Lizee provides Adreline Heroes with two technology blocks:

1. **A *rental cart* frontend component**. What's that ? It's a component that asks for the rental dates (like Airbnb), displays all the rentable products with + / - buttons to add them to the cart, the total per day, the duration of the rental, the total price and a checkout button (example on the right).



2. **An *availability* API** that returns the availability of the whole product catalog between two rental dates. The API exposes the following route :

POST /availability
Request body: two dates **from** and **to**
Response : an array of all the products with the availability for a rental during the given period
[{ "product_id": "34215", "available": 34 },
{ "product_id": "35231", "available": 2 },
…]

Your challenge ? Use this playground to demonstrate your skills

You are keen on frontend? Build a demo of a rental cart component that *does not* allow the customer to rent more products than available – focusing on usability (See **Frontend use-case**).

Skilled on backend? Build a prototype of the availability API focusing on business logic and scalability (see **Backend use-case**).

You are more full-stack? Feel free to make a mix... for example a browser-only JS app with a basic rental cart (not a lot of work on UX/UI) and some of the business logic.

What's important ? It does *not* have to be feature-complete. ***Show us the way you work, your expectations of quality, what you can do in a few hours***. And tell us what you would have done (and how) if you had a few more days to put it into production. What would be missing?

Frontend use-case

Your mission: build a *rental cart* frontend component that *does not* allow the customer to rent more products than available.

Assumptions: there are two APIs : 1. the availability API (ID, availability) and 2. a product catalog (ID, label, daily price). You can mock them all!

Expectations:
- a git project with a working demo of the component
- it should be possible to adjust availability (for example via mocks / fixtures) to test various states
- Tip : Storybook may help if you are used to it.
- a README with explanations of your choices, and what you would have done with more time.

How we'll challenge your
Does it work? Is usability good?
What UI can you do without artwork?
Does it handle edge cases correctly/gracefully?

Backend use-case

The logistics of a rental are the following:

- **Delivery**: A parcel arrives to the parcel shop 2 days after the day it's shipped from the warehouse.
- **Return**: After the customer returns the products to the parcel shop, it also takes 2 days to get it back to the warehouse.
- **Refurbishment**: cleaning takes 2 days.

				"from"				
Time (days) =>	***Shipping day (D)***		***D+2***	**Rental start**				
Steps =>	**Delivery**				**Rental**		**(...)**	
Events =>	Taken from stock		Available					
	& shipped from		at parcel shop					
	the warehouse							
		"to"						
Time (days) =>		**Rental end**	***Rental end +1***					
Steps =>	**(...)**	**Rental**	**Return**		**Cleaning**			
Events =>			Customer normally		Arrival at		Product back	
			returns product to		warehouse		in stock	
			the parcel shop					

A product is unavailable for every day that it spends in the various steps (delivery, rental, return, cleaning/repair).

Just to let you know, reality is always a bit more complex :)

- **Delivery**: It *normally* takes two days. The logistics provider commitment is 3 days maximum but … he's only reliable at 98 %.
- **Return**: Most customers will return the products on time, but 25 % of them are late (80 % one day late, 15% two days late, and 5% more than two days late).
- **Refurbishment**: apart from cleaning, 10 % of the products need repair (+2 days).

Here is an example of how availability can be computed for a demo customer (not taking reality - provider unreliability, late customers, necessary repairs - into account) :

- A customer wants to rent a single product (tent) for his camping trip. He visits the product page of the tent on April 2nd.
- He inputs the dates of this trip : from April 25th to May 9th (15 days).
- Among the 10 tents that are globally assigned to rental, none is currently in stock at the warehouse (they are all rented out). But before April 25th, there will be 3 returns to the warehouse: two on April 15h, and one on April 22th. There is currently 1 other rental starting April 22th.
- The customer needs to have it delivered on April 24th at the latest (1 day before the start of his trip) and it takes 4 days to re-deliver a product (2 days of refurbishment, and 2 days of delivery) therefore :
    - the one returning on April 22th cannot be used for this customer
    - among the two returning on April 15th, one will be assigned to the other rental, which leaves one available.
- Thus the API returns “available: 1” for this customer and these dates, which is enough to proceed to checkout (he only wants to rent one tent).

Your mission : Build a prototype of the availability API based on total stocks and future rentals. You can focus on business logic (build-in reliability, edge cases are covered) and/or on scalability (its architecture makes it possible to be deployed reliably to a website with heavy traffic of any size – hosting costs are the variable).

Assumptions:

- Programming language of your choice
- Taking *reality* (provider unreliability, late customers, necessary repairs) into account is a bonus, but that’s only if you’ve got a lot of time to spend :)

Expectations:

- A git project with a working demo of the *availability* API
- It should be possible to create rentals either in fixtures or the API (e.g. a */checkout* route)
- A README with explanations of your choices, and what you would have done with more time before putting it to production (assuming that you want to do that as soon as possible).

How we’ll challenge it:

We will stress test the API: does it work?
We’ll inspect the code and discuss the architecture: how is reliability built in? can it be scalable?

Happy coding!

Lizee
From selling to renting