Fabric

Cell Allocation Exercise

We have a warehouse with a single 10X10 shelving unit.
Each cell in the shelving unit holds stock.
When receiving stock from suppliers, we need to decide where to place the received products based on some rules:

1. Each cell in the shelving unit can hold only one type of product.
2. Each cell in the shelving can hold a max of 10 items.
3. Chilled products (milk, yogurt etc) can only be stored in cells with a cooling system.
4. Hazardous products (bleach, stain removal etc) can only be stored in cells that are marked as such.

Your task is to write a server that given a request in the form of

POST /`allocateCell`

```
{
    "productId": <string>,
    "quantity": <number>
}
```

Returns a response in the form of

```
{
    "foundCell": <bool>,
    "cell": <string>
}
```

The server needs to comply with the storage business rules mentioned above and work with the known products and the shelving unit initial state.
Note that when sending a request to allocate a cell and a cell is found, the cell's capacity should be reduced by the requested products amount

Known products:

Chilled: milk, yogurt, cheese, insulin
Hazardous: bleach, stain removal, insulin
Others: bread, pasta, salt, bamba, apple
Shelving unit initial state:

	0	1	2	3	4	5	6	7	8	9
0		3*bread								(h)8*bleach
1										(h)
2										(h)
3				5*bamba						(h)
4										(h)
5										(h)
6										(h)
7								(c)	(c)	(c, h)
8								(c)	(c)	(c, h)
9								(c)	(c)	(c, h)

* c - a cell with a cooling system
* h - a cell for hazardous products

Highlights

1. **Get it to work** - we value working code over perfect code.
2. **Layering** - make sure you think about the different responsibilities of different components in your solution, and how they interact.
3. **Extensibility** - in the future we would probably need to support new types of rules for storing new types of products, so structure your code in a way that would easily support that. **For example**: **We might introduce different sizes of cells (small, medium, large) and some products won't fit in a small size cell according to their height/width.**
4. **Validations** - your solution can fail in multiple different cases, think about how you handle errors, bad inputs etc.

Examples:

Here are some acceptable requests and responses, given the above initial state.

****

Request:

```
POST  /allocateCell
{
    "productId": "bread",
    "quantity": 2
}
```

Response:

```
{
    "foundCell": true,
    "cell": "0,0"
}
```

****

Request:

```
POST  /allocateCell
{
    "productId": "bamba",
    "quantity": 6
}
```

Response:

```
{
    "foundCell": true,
    "cell": "2,0"
}
```

****

Request:

```
POST  /allocateCell
{
    "productId": "milk",
    "quantity": 6
}
```

Response:

```
{
    "foundCell": true,
    "cell": "7,7"
}
```

****

Request:

```
POST  /allocateCell
{
    "productId": "apple",
    "quantity": 11
}
```

Response:

```
{
    "foundCell": false
}
```

****

Request:

```
POST  /allocateCell
{
    "productId": "stain removal",
    "quantity": 7
}
```

Response:

```
{
    "foundCell": true,
    "cell": "9,1"
}
```

****

Request:

```
POST  /allocateCell
{
    "productId": "insulin",
    "quantity": 10
}
```

Response:

```
{
    "foundCell": true,
    "cell": "9,7"
}
```