Fake cash machine

Let’s start with expectations

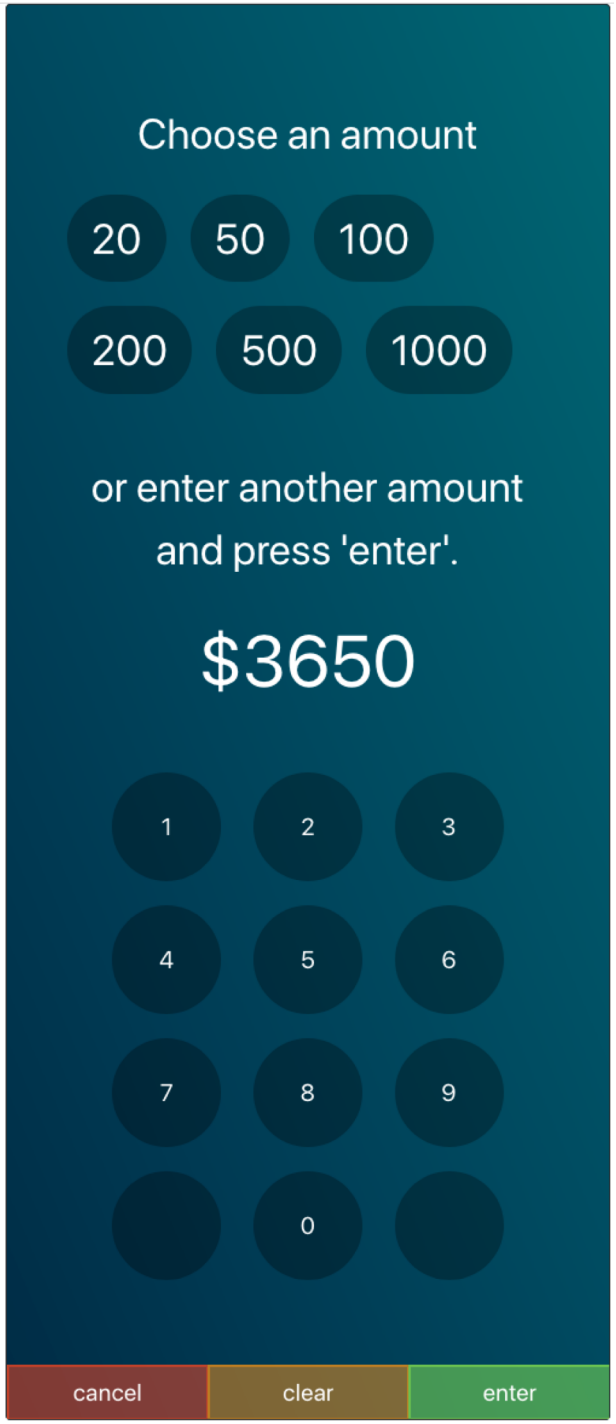
I’ve made a fake cash machine ment to deliver smallest quantity of notes using available notes set.

Here are assumptions:

- Number of notes is inifite
- Client balance is infinite
- Available notes are: $100, $50, $20 and $10

Here’s how it looks

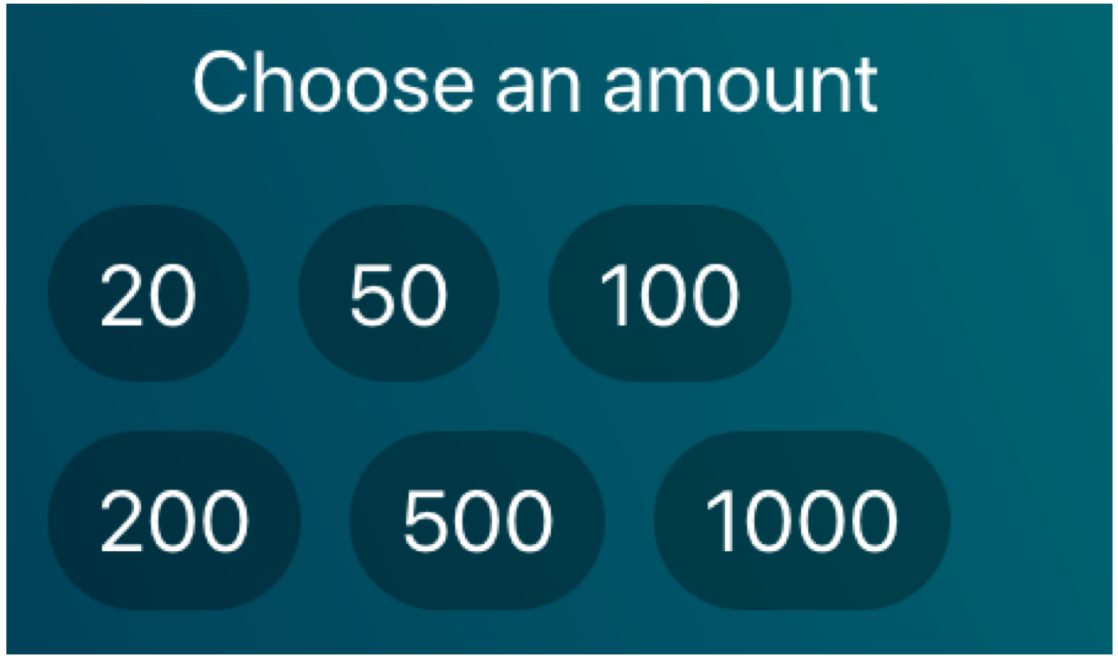
Most cash machines still looks like made in early 90’s. I’ve tried to approach it a bit more modern but still keeping familiar design so user won’t get lost.



In details. Top bar – suggested amounts

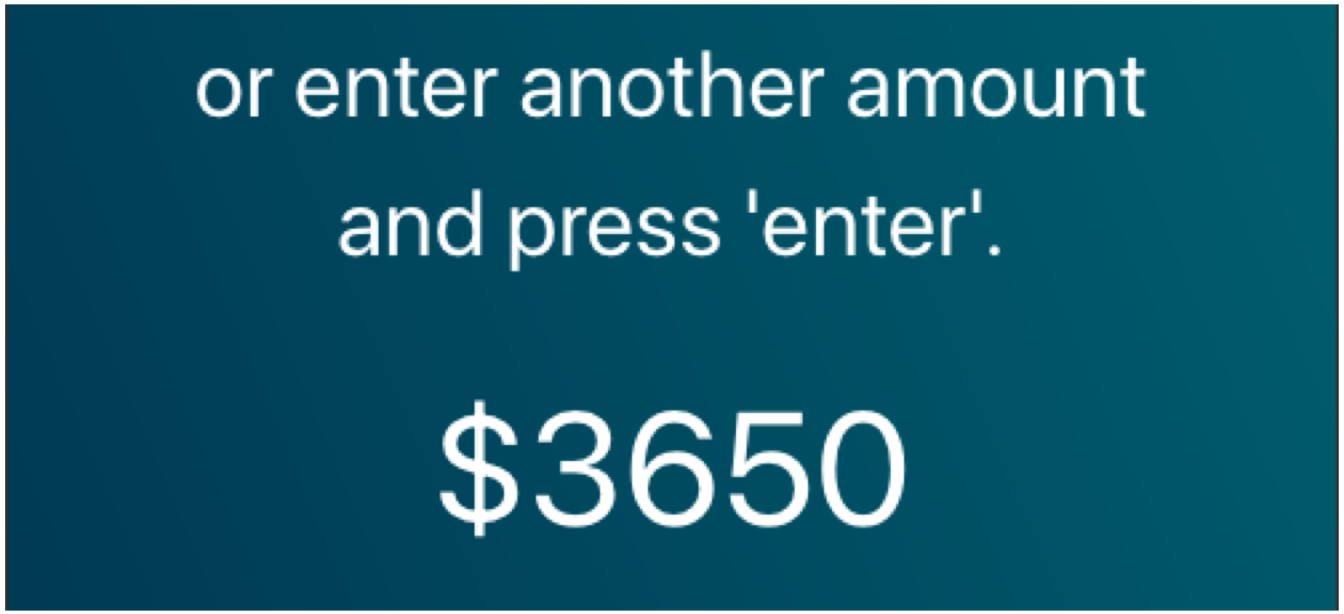
User is able to pick from suggested (i.e. most common) withdrawal amounts. After choice he still has to approve it with ”enter” buton.

Styled as classic tag flow to allow any numer of suggestions to be displayed.



In details. Amount – selected value

This part displayes selected or typed amount and notification about possibility to type anything on the keyboard.



In details. Keypad

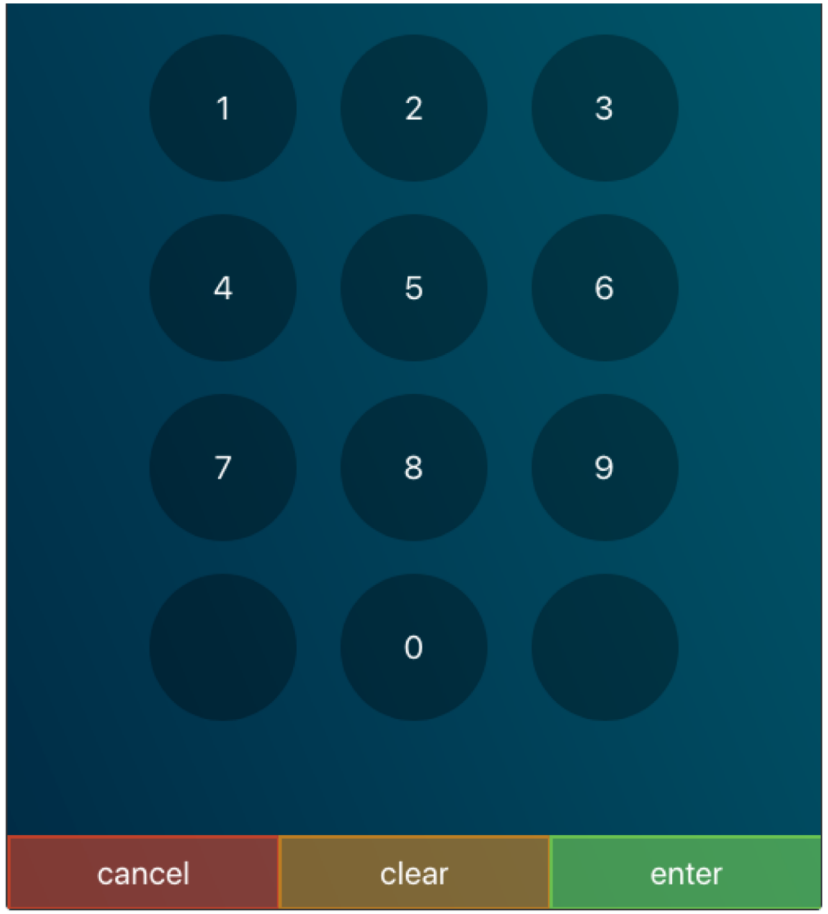
In here user can type desired amount, correct selected one and finish transaction.

Logic behind dial buttons simply add key value behind current value, i.e. `1 + 1 = 11` with exceptions like `0 + 1 = 1` or `0 + 0 = 0`

Clear removes last digit. (or resets to 0 if last digit was removed)

Cancel resets everything to 0

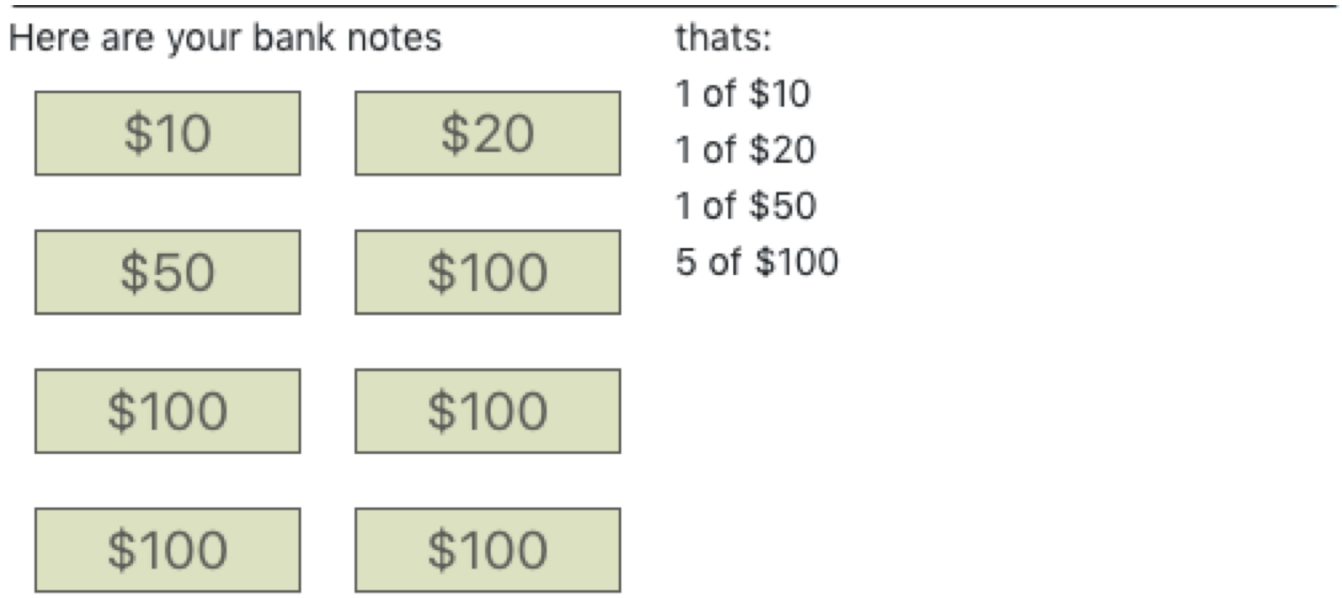
And Enter begins calculation of notes



In details. Simple representation of notes

This part calculates numer of notes and represents it in two ways.

1. Visual representation of notes
2. Summary of numer of each notes



Logic

Most important algorithm is the one that calculates number of notes. I've considered multiple approaches and while some of them (using es6 features) might look more readable or just shorter in form they won't provide equal performance as simple forEach loop.

In short we sort notes descending, and then for Each note we:

1. Check if value is natural number (incl. 0)
2. Check how many will fit inside value
3. Record note and quantity in aggregator
4. Leave the rest for next notes

5a. If we've runned out of notes and there's still some value – we don't have needed notes and throw an error with the smallest note we have as division reference.

5b. If rest is 0 we can return aggregator values

Tests

Cash mashine has only unit tests in TDD manner to make sure important calculations are valid. There are no visual tests as I wanted to finish within one day.

```
PASS src/helpers/MathHelper.test.js
  notes
    ✓ calculates number of each note needed to provide lowest quantity of notes (7ms)
    ✓ returns error if cannot return expected number using available notes
    ✓ returns error if not natural number (1ms)
    ✓ returns empty set when value is 0 or null (1ms)
  notesToArray
    ✓ returns notes as an array where each element represents each note (1ms)
  digitRemove
    ✓ removes last digit (1ms)
  digitInput
    ✓ inputs new digit (1ms)

Test Suites: 1 passed, 1 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        8.299s
Ran all test suites.

Watch Usage: Press w to show more.
```

Why redux?

Cash machine is perfect case for redux state machine. While for the part delivered we could rely only on component state – if I'd ever consider extending features on this projects state would quickly become hard to maintain. And cash machine has it's on phisical state for which it won't query any remote database (cash amount, notes, location, card inserted etc).

Why I didn't make any API calls in here? I could fit 2 calls inside – to check customer balance and to trigger payout. I've decided to not to as it's not required to deliver requested features so I could focus on other parts of app. Fetch API is rather simple one, similar to sagas so nothing to show off.

Thank you for your time!

krzysiek@sth.io