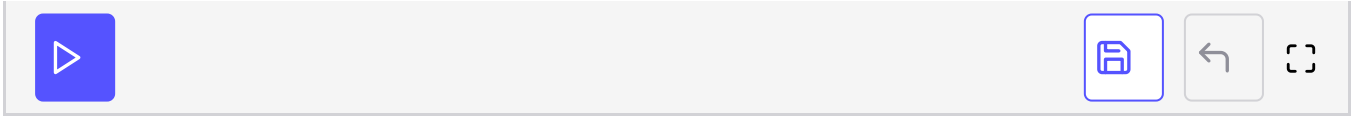
Step 3: Account Module

To process transactions, we need an `Account` class. We will use the class syntax of ES6 to write the code. A short explanation will follow the example. Before reading the explanation, try to figure out what the code does.

```
import { sortBy, first } from 'underscore';

class Account {
    constructor() {
        this.transactions = [];
    }
    getTopTransactions() {
        var getSortKey = transaction =>
            -Math.abs( transaction.amount );
        var sortedTransactions = sortBy(
            this.transactions,
            getSortKey
        );
        return first( sortedTransactions, 3 );
    }
    deposit( amount, date ) {
        this.transactions.push({
            amount : amount,
            date   : date
        });
    }
    withdraw( amount, date ) {
        this.transactions.push({
            amount : -amount,
            date   : date
        });
    }
};

export default Account;
```



Save the above code in `src/Account.js`. We will also need UnderscoreJs as it's defined as a dependency. Grab it using

```
npm install underscore --save
```

Underscore is a regular dependency, hence the `--save` flag. All other packages have been devDependencies, and they were added to the `package.json` using `--save-dev`. Development dependencies, such as transpilers or automated testing frameworks, are only used during development.

The code works in the following way:

- We import the `sortBy` and `first` functions from the Underscore functional programming utility belt
- When creating an account with the `new` operator, we initialize the transactions to `[]`
- To demonstrate how to import Underscore functions, we derive the top 3 transactions. After sorting the transactions based on descending absolute transaction amount, we take the first three elements from the list.
- Depositing and withdrawing are both straightforward: we push a new transaction object to the `transactions` array, setting its amount and date