Introduction

Dearest candidate!

To test your skills we'd like to see you implement a simple mini application called “Toshl Killer”. It's a single-page web application that helps people track expenses and incomes.

You'll be provided with detailed step-by-step instructions which you'll use to build the application incrementally. We'll also supply you with an external API that your application will consume for fetching, saving, updating and deleting records. If you get stuck or have any questions, don't hesitate to ask!

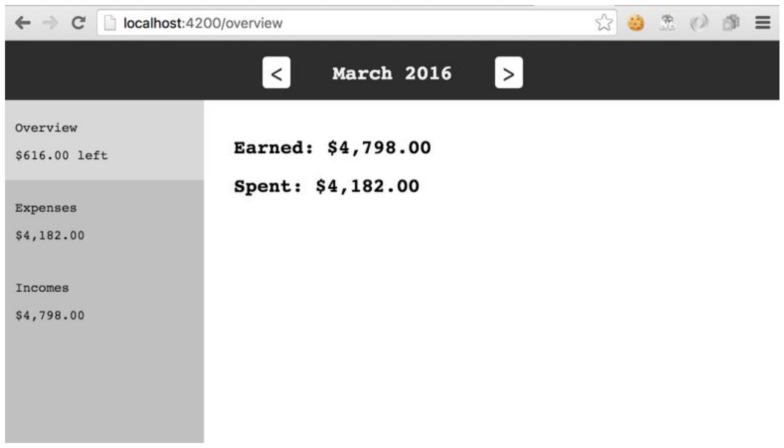
Before starting, we suggest to briefly read through the assignment and API documentation so you'll get a general idea of what needs to be done. Try to commit code regularly, preferably at the end of each finished task.

ASSIGNMENT

1. Create a repository for your application on Github.

2. Make a basic template with a sidebar, navbar and central area. Apply very basic styles using (S)CSS or Bootstrap, so the layout will look similar to images below. Clicking into different sections in sidebar should render the section content into the main area. Use routing. For now, hard-code all the values.

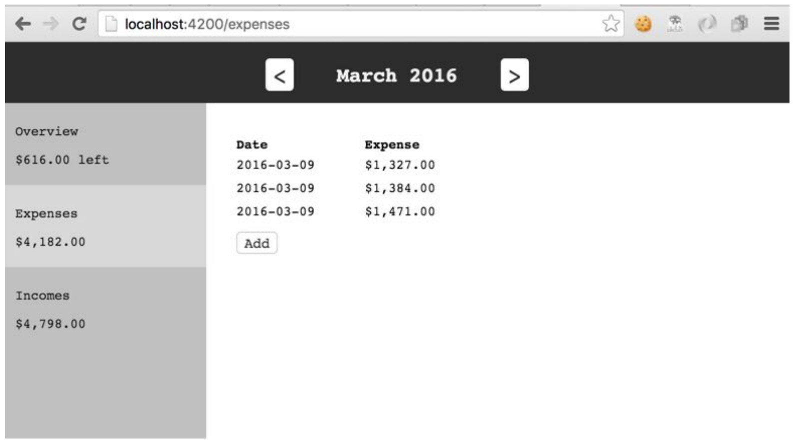
    a. Overview



Earned: $4,798.00

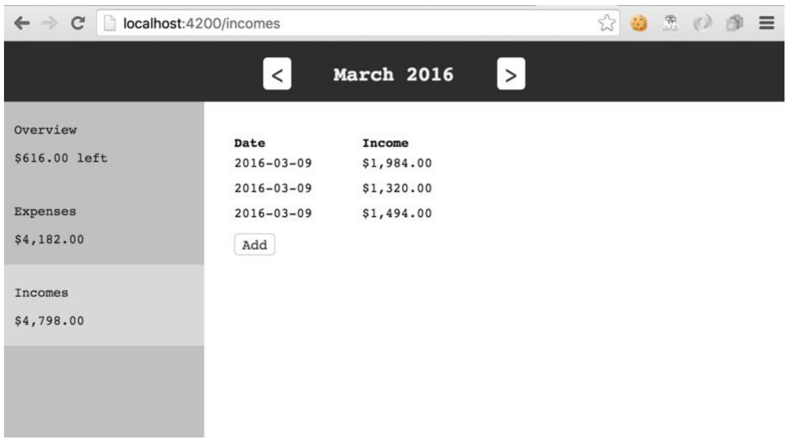
Spent: $4,182.00

    b. Expenses



Date	Expense
2016-03-09	$1,327.00
2016-03-09	$1,384.00
2016-03-09	$1,471.00

    c. Incomes



Date	Income
2016-03-09	$1,984.00
2016-03-09	$1,320.00
2016-03-09	$1,494.00

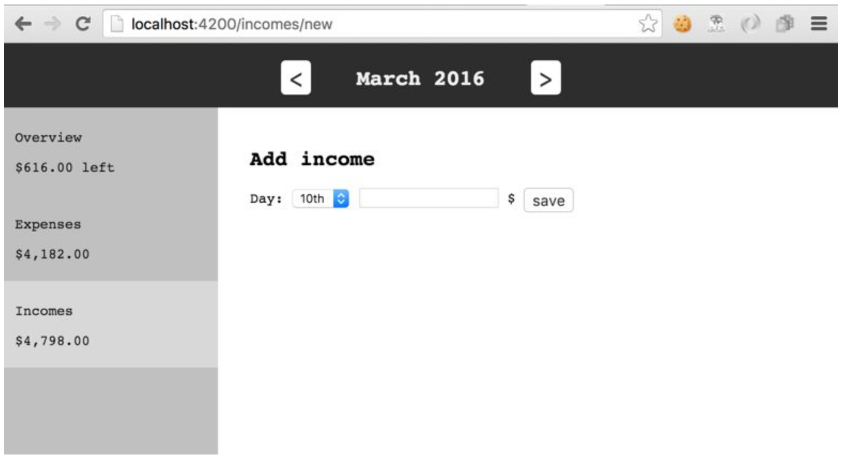
3. Make period selector work. Click on the “<” button should display previous month and click on the “>” button should display the next month. For instance, If there is “March 2016” selected, click on button “<” should replace it with “February 2016” and click on the button “>” should replace it with “April 2016”.

4. Fetch data from the API with the period filter (example is in API documentation) and replace hard-coded values with data sent from the server. Values come in cents, so format the display of value to display a nice formatted value. Every change of the period filter (i.e. click on “<” or “>” button) should refresh all data on the screen:

   a. Calculate the **sum of all expenses for selected period** and display it in the **sidebar** and **overview**
   b. Calculate the **sum of all incomes for selected period** and display it in the **sidebar** and **overview**
   c. Calculate the **difference of expenses and incomes for selected period** and display it in **the sidebar under “Overview”**. If we have any money left, it should say “$XXX left”, otherwise “$XXX in the red”.

5. With fetched data, populate incomes and expenses lists, as depicted in the upper images.

6. Implement adding incomes and expenses. Each of the lists should have an “Add” button in the bottom. On “Add” button click, a form similar to the image below should be rendered. First input is the day of the month, which should have a default value of the current day. Second input is value of the expense or income in cents. After save, user should be taken back to the list, and all displayed data should refresh.


localhost:4200/incomes/new

< March 2016 >

Overview

$616.00 left

Expenses

$4,182.00

Incomes

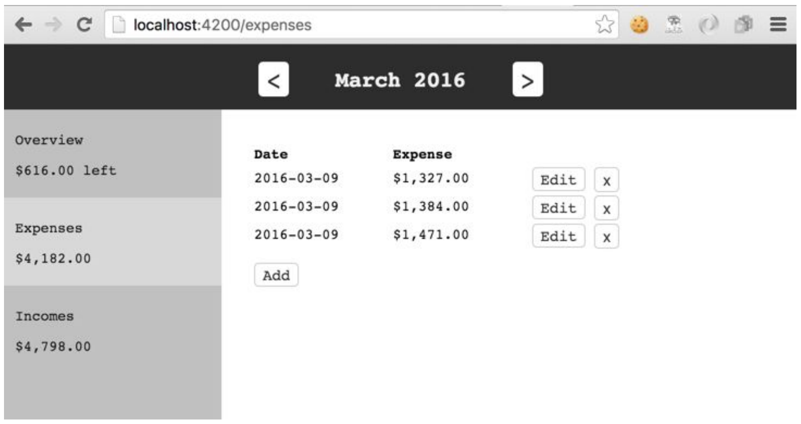
$4,798.00

Add income

Day: 10th $ save

7. BONUS: Implement deleting. Place an “X” icon next to balance changes that triggers a DELETE request to the server. Refresh data afterwards.

8. BONUS: Implement editing. Use a similar form as with adding.


localhost:4200/expenses

< March 2016 >

Overview

$616.00 left

Expenses

$4,182.00

Incomes

$4,798.00

Date	Expense		
2016-03-09	$1,327.00	Edit	x
2016-03-09	$1,384.00	Edit	x
2016-03-09	$1,471.00	Edit	x

Add

API DOCUMENTATION

URL: http://toshl-killer.herokuapp.com/api/v1/

GET api/v1/balance_changes

Filter params:

- period (optional, format: filter[period]=YYYY-MM), example: /api/v1/balance_changes?filter[period]=2016-01

Response example (200 OK):

```
{
  "data": [
    {
      "id": "798",
      "type": "balance_changes",
      "attributes": {
        "value": 11500,
        "change_type": "expense",
        "entry_date": "2016-01-31",
        "created_at": "2016-03-08T11:56:42.449Z"
      }
    },
    {
      "id": "797",
      "type": "balance_changes",
      "attributes": {
        "value": 18300,
        "change_type": "expense",
        "entry_date": "2016-01-31",
        "created_at": "2016-03-08T11:56:42.449Z"
      }
    }
  ]
}
```

entry_date format: YYYY-MM-DD

POST api/v1/balance_changes

Request payload:

```
{
  "data": {
    "attributes": {
      "value": 100,
      "change_type": "expense",
      "entry_date": "2016-03-08"
    }
  }
}
```

Request response (Status 201 Created)

```
{
  "data": {
    "id": "799",
    "type": "balance_changes",
    "attributes": {
      "value": 100,
      "change_type": "expense",
      "entry_date": "2016-03-08",
      "created_at": "2016-03-08T11:28:09.600Z"
    }
  }
}
```

PUT api/v1/balance_changes/:id

Request payload:

```
{
  "data": {
    "attributes": {
      "value": 101,
      "change_type": "expense",
      "entry_date": "2016-03-08"
    }
  }
}
```

Request response (Status 200 OK)

```
{
  "data": {
    "id": "799",
    "type": "balance_changes",
    "attributes": {
      "value": 101,
      "change_type": "expense",
      "entry_date": "2016-03-08",
      "created_at": "2016-03-08T11:28:09.600Z"
    }
  }
}
```

DELETE api/v1/balance_changes/:id

Request payload not necessary.

Response (Status 200 OK):

```
{
  "data": {
    "id": "799",
    "type": "balance_changes",
    "attributes": {
      "value": 101,
      "change_type": "expense",
      "entry_date": "2016-03-08",
      "created_at": "2016-03-08T11:28:09.600Z"
    }
  }
}
```