Papier: A todo list app

- M Haidar Hanif (54411850)
- Freddy Arviando (52411954)
- Alvin (50411628)

Figure 1: Papier Screenshot

Sections

This documentation consists of the following topics:

- Requirements: Statements that identify attributes, capabilities, characteristics, or qualities of a system. This is the foundation for what shall be or has been implemented.
- Architecture: Quick overview of software. Includes relations to an environment and construction principles to be used in design of software components.
- Technical: Documentation, explanation (simple algorithms), and interaction details.
- Usage: Manuals for the end-user and other stuffs.

Introduction

Papier is a todo list app –in a form of virtual paper– for a very simple list or note management. "Papier" name is from German, French, or Dutch word of "Paper". Visit http://bit.ly/papierexp to view and use the app.

Requirements

Development Process

The whole process of developing this application is using "agile software development", which basically means there are constant update and change for converting idea into temporary and permanent implementation. So all of the requirements quickly delivered into design and working app/code like a minimum viable product (MVP). Thus building the real thing is more important or doing the inessential things is not done, then there are more practices and working things than just utterly a management and process.

Design

Interface segments:

- Title: The application's name
- Form: To type a text as an input
- Buttons: Naturally as an action text and symbol
- Status: Active list item count
- Style: Like a paper on a table

Features

- Write: Type a text then press `Enter`
- Edit: Double click a list then retype the text
- Mark complete: Hover a list then click the `green check` button
- Mark all as complete: Click the `down chevron` button on the input form
- Delete: Hover a list then click the `red cross` button
- Clear: Click the `clear completed` button
- Filter: Click `all`, `active`, `completed`
- Count all active list item automatically on the status

Early Mockup

The mockup is created as the initial design direction.

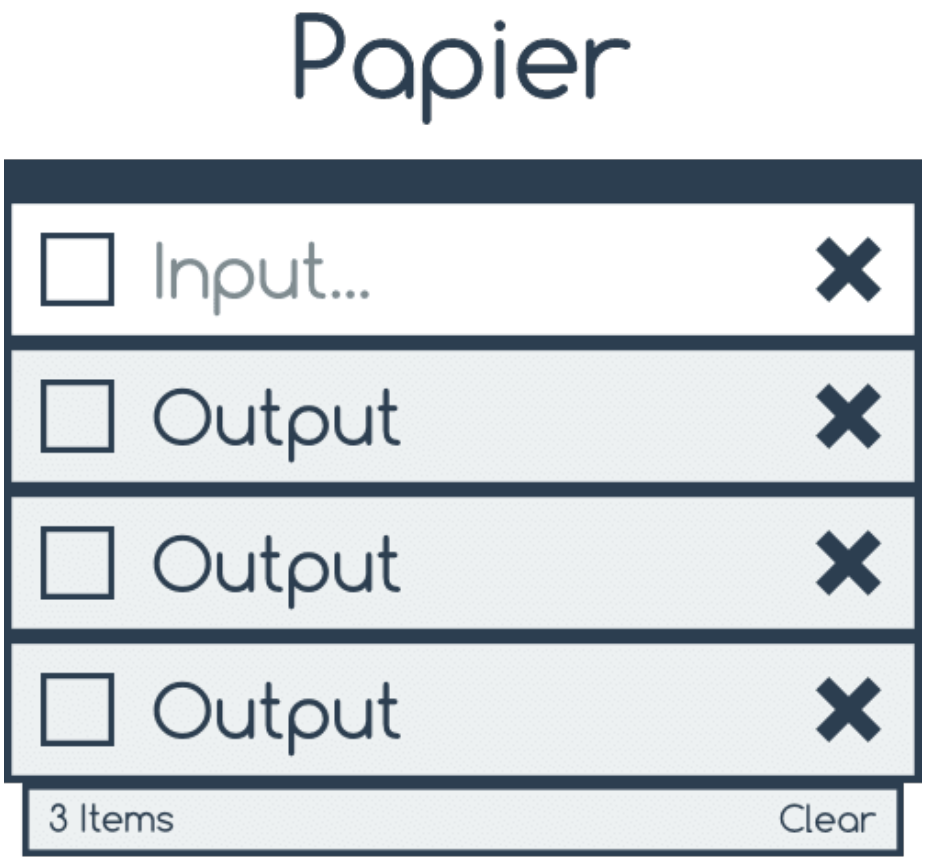

Figure 2: Papier Mockup

Architecture

Below are all of the architecture pieces used to implement the requirements.

Components

It's mainly built based on:

- A todo list style
- A one page JavaScript application, with HTML and CSS
- A JavaScript MVW (Model-View-Whatever) framework called AngularJS
- A browser's local storage

Main App Project Structure

```
| app
  | bower.json
  | index.html
  | css
  | fonts
  | images
  | bower_components
    | angular
    | todo-common
  | js
    | app.js
    | controllers
    | irectives
    | services
```

Todo List Style

A simple todo list functionality and style to write, edit, delete, or mark a list. Like a paper to keep any list.

HTML5 and CSS3

The front-end document is using HTML5 and interface style is using CSS3.

Application Framework

Using AngularJS as the MVW, closer to MVVM framework. It extends HTML vocabulary for application by declaring dynamic views in web-applications. The resulting environment is extraordinarily expressive, readable, and quick to develop.

The framework consists of:

- Model: State the specification and behavior, direct the ViewModel and be updated by that.
- View: Render the display and layout, managed by the ViewModel and send the information to the ViewModel.
- ViewModel: Or previously defined as Controller, as the logic of interaction. It modifies the View and reacted to action or change in the app.

To illustrate, the framework was like this back then:

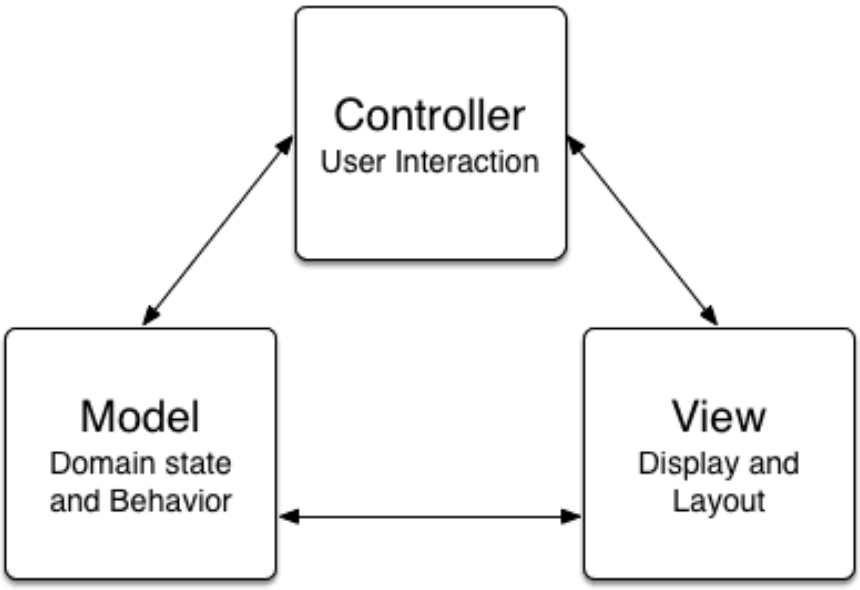

Figure 3: MVC Illustration

Now it's more likely this:

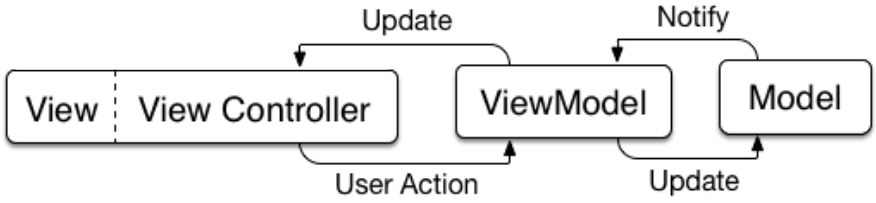

Figure 4: MVVM Illustration

Dependency Management

Using Bower for package management (see `bower.json` file). This allows to manage or update AngularJS framework neatly.

Storage

Rather than using a database, a quick local storage (aka `localStorage` or HTML5 Web Storage) is used. The data is stored in JSON (JavaScript Object Notation), a lightweight data-interchange format. This will dynamically persist the created lists to local storage. It's possible to use the keys id, title, completed for each list item.

For example, this is how it looks like:

```
Storage {
  todos-papier: "[
    {"title":"Software","completed":false},
    {"title":"Development","completed":false},
    {"title":"Process","completed":false}
  ]",
  length: 2
}
```

Routing

The following routes is implemented (`#!/` is also allowed):

- `#/` (all as default)
- `#/active`
- `#/completed`

When the route changes the lists should be filtered on a model level and the selected class on the filter links should be toggled. When an item is updated while in a filtered state, it should be updated accordingly. So if the filter is *Active* and the item is checked, it should be hidden. Make sure the active filter is persisted on browser reload.

Technical

In the lowest sense of logic layer, the application basically just process a JSON data with a user front-end interface.

Write

Write, add, or create a new list in the input form.

Edit

Edit the created list in the output. By double-clicking the `<label>`, editing mode is activated and the `.editing` class is toggled on its `<li>`. That also hides the other controls and bring forward an input that contains the current list title/content, which should be focused. Once saved, the `.editing` class is removed. Make sure to check the input should be valid and not empty. If it's empty the it should instead be deleted. If escape is pressed during the edit, the edit state should be left and any changes be discarded.

Mark

Mark a list as complete. This can be done by checking the check button on a particular list. Then this also updating its `completed` value and toggling the class `completed` on its parent `<li>`.

Mark all

Mark all available list as complete. Iterate over all of the lists then mark the list.

Delete

Delete particular created list. This can be done when a list is hovered over, then the delete button is shown.

Clear

Clear all the completed list. Basically it deletes every completed list. Iterate over all of the completed list then delete the list.

Filter

Filter between all available list, active list (not yet completed), and completed list. This also affects and affected by the routing.

Count all active list item

Passively show the active list items. This acts as a counter to display the number of active list items in a pluralized form. If the interface language is in English, make sure to pluralize the item word correctly.

Usage

This README also provide the usage manual at a time. The decent neat documentation is available at http://documentup.com/mhaidarh/papier and the presentation about this app is available at http://bit.ly/papierpres, both also combine as a manual.

Source code is available on GitHub at https://github.com/mhaidarh/papier. Feel free to view, download, or fork the project.

Changelog

0.1.1 (Roadmap)

- Sort descending/ascending based on various parameters

0.1.0 (May 2014)

- Initial public release
- Basic functionality

License

This project is licensed under MIT License. So basically, you can do whatever you want as long as you include the original copyright and license (https://tldrlegal.com/license/mit-license). Therefore you:

Can

- Use commercially
- Modify
- Distribute
- Sublicense
- Use privately

Cannot

- Hold liable

Must

- Include copyright
- Include license