Tomato Tasker
Brief Documentation

LINH LE ANH VIET

Table of Contents

1 Introduction

1.1 Purpose

This web application is at its core inspired by Pomodoro method. The Pomodoro method prescribes an iterative working habit whereby during working sessions 25 minutes are reserved for working and 5 minutes for a break. This application offers a customizable Pomodoro timer that is by default set to 25 minutes of work and 5 minutes of break. However, the application further offers task creation where the user can create tasks with their own individual working and break times and assign it to the timer.

1.2 Product Perspective

This application aims to complement the Pomodoro method by offering and interface for tracking the user's time as well as offering flexibility in creating their own individual sessions. Further additions to the application may include an automatic tasks queue with customizable order, a personal TODO list, calendars, notifications etc.

2 Brief Setup and User Guide

2.1 Setup

Make sure you have node.js installed on your computer. Copy and paste the following commands into the command line:

```
git clone https://github.com/leanhvie/tomato-tasker.git
cd tomato-tasker
npm install
npm run build
```

'npm install' command will download all the needed dependencies for running the application. The process may take a while.

From here there are two ways to run the application:

- Run the application with webpack-dev-server by typing 'npm run dev' into the command line, where changes to the code will affect the running application.
- Run the application by simply opening the 'index.html' file.

2.2 User guide

You can start using the timer by clicking on the 'Timer' navigation link. This will direct you towards the timer section with a radial progress timer. Click on the button below to start the timer. You can then click on it again to pause. The default setting for the timer are 25 minutes for work and 5 minutes for break.

If you want to assign a task to the timer, then firstly click on the 'Add task' button in the top-right corner of the application. Fill in the form by providing a title, description working and break times in hh:mm:ss format and number of cycles. After submitting the form, you will be redirected to the tasks page, where all the created tasks are listed.

Each task can be updated or deleted as well as assigned to the timer. By clicking of on the clock icon of the respective task, you will be redirected to the timer section where it will be ready to proceed according to the times you provided to the chosen task. Again, click on the button below to start the timer.

The application works offline as well, however it then only functions as a simply Pomodoro timer without the added tasks functionalities.

3 Used Technologies and their Implementation

Principal technologies used:

babel:	Transpiler for transpiling code written in ES6 standard to ES5 standard. Used for browser-compatibility.
Bootstrap:	A CSS framework that facilitates styling of web application/
ES5/ES6:	Scripting-language specification standard for JavaScript.
Firebase	Mobile and web application development platform. Here it is used mainly for its real-time database.
Flexible Box	A layout mode in CSS3, intended to accommodate different screen sizes and different display devices.
Flux	UI architectural patter for managing the state of React application.
ReactJS:	JavaScript framework for creating components used in the user interface**.**
webpack:	Module bundler that takes modules with dependencies and generates static assets representing those modules.

The layout of the web application is shaped be the 'flexbox' layout mode in CSS3. This layout mode allowed for easy positioning of various sections and components within the application.

The UI of the web application was built using the React framework. React separates the UI into multiple smaller components enabling a fine-grained control over individual pieces of UI, such as setting states and subcomponents for each of those. Furthermore, there are many pre-made components posted on GitHub which can be used in the application itself. One major library for pre-build components is react-bootstrap, inspired by the original CSS framework Bootstrap, on which the application heavily relies on creating the UI.

Each React component in the source code will have further comments to describe their role and purpose to the web application.

Navigation around sections is simply done by a CSS selector that displays content based on the URL. Redirection is mostly done with History API and location manipulation as the application is supposed to be a single-page application.

The front end architectural structure uses the Flux pattern developed by Facebook to control the flow of data around the UI. The concept of Flux revolves around stores and its interaction with React components which facilitated monitoring the flow of data between these two. It allows for separation of concerns, as the view (React component), the client logic (Action) and the UI state (Store) themselves are separated into three different entities, allowing for better maintainability and avoiding code duplication.

One challenge was adapting to the asynchronous nature of JavaScript and the client-server communication. Often, parts of code had to be rewritten because the React components have rendered with non-existent data since the fetching and rendering process is asynchronous. Fortunately, React provides the setState() method whereby each call to that method rerenders the component. Coupling this method with the Flux

architecture, where changes to the store, such as loading data from Firebase API, emit a ‘change’ event, the components rerendered correctly with the appropriate data.

The problem with this approach was that for a split second, the component changed its state twice, resulting in inconvenient effects for the user such as seeing flashes of UI components. This problem is remedied by setting flags whenever the application is fetching data from the server. Whenever the application is flagged, the UI simple renders a loading animation.

All the front-end code, except for webpack configuration file, is written in ES6 standard whose syntax supporting creating classes slightly like common OOP languages. This syntax compliments the component-oriented design of React. Furthermore, the arrow function notation feature of ES6 made it easier to use closure syntax of JavaScript, useful for higher-order functions and for its lexical ‘this’.

Webpack is then used to bundle and compile the front-end code into one .js file. Webpack offers several supports for plugins such as babel, an ES6 transpiler since not all browsers support ES6, or a plugin to minify and deduplicate code, making it ready for production.

4 Application Environment

4.1 Operating Environment

The first version of the application is web based meaning it can run on any device that has a web browser. A mobile application version may be developed in the future. Nevertheless, the application has a responsive design and can therefore be used on a mobile or tablet browser without any problems.

The front end of the application is developed with ReactJS, a JavaScript framework that relies on ES5. Thus, the application will not function properly on older browsers since these don't support ES5.

4.2 External Interface Requirements

4.2.1 Hardware Interfaces

The user can use the application from any device capable of running a web browser.

4.2.2 Software Interfaces

The code is written in JavaScript using ReactJS and the ES6 standard. Since ES6 is not supported in many browsers, a transpiler such as babel is needed to transpile ES6 code to ES5 which is supported more widely across web browsers. Other front-end dependencies used are pre-made React components and webpack for bundling different JavaScript files into one bundle.

5 Grading Table

Téma	Popis	Povinné	Body	
Dokumentace	cíl projektu, postup, popis funkčnosti, komentáře ve zdrojovém kódu	X	1	There was an attempt...
HTML 5			**10**	
Validita	Validní použití HTML5 doctype	X	1	There was an attempt...
Cross-browser kompatibilita	Fungující v moderních prohlížečích v posledních vývojových verzích (Chrome, Firefox, Edge, Opera)		2	Should hopefully work
Semantické značky	správné použití sémantických značek (section, article, nav, aside, ...)	X	1	There was an attempt...
Grafika - SVG / Canvas			2	SVG radial progress
Média - Audio/Video			1	Audio played at the end of timer...?
Formulářové prvky	Validace, typy, placeholder, autofocus		2	Forms for creating and updating tasks
Offline aplikace	využití možnosti fungování stránky bez Internetového připojení (viz sekce Javascript)		1	offline manifest
CSS			**8**	
Pokročilé selektory	použití pokročilých pseudotříd a kombinátorů	X	1	Non-reload page transition, child selectors in landing page
Vendor prefixy			1	Vendor prefixes in custom.css stylesheet
CSS3 transformace 2D/3D			2	SVG radial progress (rotate), tasks shrink on delete
CSS3 transitions/animations		X	2	SVG radial progress, tasks shrink on delete
Media queries	stránky fungují i na mobilních zařízeních i jiných (tedy nerozpadají se)		2	Media queries
Javascript			**12**	
OOP přístup	prototypová dědičnost, její využití, jmenné prostory	X	2	React components, ES6 classes
Použití JS frameworku či knihovny	použití a pochopení frameworku či knihovny JAK, jQuery, ..		1	React, webpack
Použití pokročilých JS API	využití pokročilých API (File API, Geolocation, Drag & Drop, LocalStorage, Sockety, ...)	X	3	Firebase API
Funkční historie	posun tlačítky zpět/vpřed prohlížeče - pokud to vyplývá z funkcionatilty (History API)		2	location, History API
Ovládání medií	použití Média API (video, zvuk), přehrávání z JS		1	Audio API
Offline aplikace	využití JS API pro zjišťování stavu		1	navigator
JS práce se SVG	události, tvorba, úpravy		2	SVG radial progress
Ostatní			**5**	
Kompletnost řešení			3	
Estetické zpracování			2	
		Celkem	**35**	