Code Documentation

The Distraction Shield

General project details

General:

A thing that is implemented is encapsulation. Keep all data private to the objects that are hold this data. Define getters and setters for this object's fields. The user of classObject.variableName should not occur. Only classObject.get/setVariableName. JavaScript does not enforce any of this but it helps a lot if it is done this way. Because you know project wide that is used like this. So you can make changes to the class without needing to go through the entire project to find out who uses what data in which way and where it is manipulated. The only place where the standard changing should happen is within the class-definition itself.

Classes:

These are files that contain two things. The first being a class definition in a very JAVA like style. Second are classNameSerialize and classNameDeserialize. These are for the storage module to use when writing and reading from storage to make sure they do not lose functionality. Classes are meant as data structures of which we will have multiple instances. Preferably each instance should be different from others. UserSettings is at this point some sort of exception to this rule.

Modules

modules are a special type of classes. These classes are not ones you make an instance of, like a blockedSite item. But they are a class definition together with one instantiation of the object at the end of the same file. This variable is per definition called "moduleName". (Example: synchronizer.js instantiates one variable called "synchronizer" which is of the type Synchronizer) The objects we find in these modules are independent. They only take data and do something with it. They do not hold important data themselves. They are simply objects, included in other files that we then may call methods upon. (url_formatter formats urls for the blockedSites, synchronizer syncs given data to background and storage, etc). This way there is no code mix between functionality, you can just include the file and have all functionality. Best of all there is no need to sync these objects. Every place that uses a module has its own instantiation of the object and that is fine. Still everything keeps working. There is no need for serialization nor for deserialization since these objects can be re-instantiated time and time again, because they function independently of the data they get. So there is no need to keep one instance going.

Folder and File specific details

From here on we will be using the following template:

Name

--[*Class*|*Module*|*Folder*|*Background Script*]--”./path/to/file/fileName.fileTypeExtension”
|>"./path/to/dependency/dependencyName.fileTypeExtension”
|>"./another/part/of/the/code/this/part/is/dependent/on.fileTypeExtension"

Basic Functionality:
Here we describe the use of this named part of the code

Variables:
Here we list and describe the variables that occur in this named part of the code.

Main flow:
Here we describe how this named part of the code works and/or should be used.

BlockedSite

-- Class -- “./classes/BlockedSite.js”

|> "./constants.js"

Basic Functionality:

It is one of the most basic and important classes of the project. An instance of this class is one object concerning a website the user wants to be kept from. It has getters for all variables and setters for the checkboxValue and the counter, since those are the only one that should be allowed to change.

Variables:

url	= String holding url (“*://www.website.com/*”)
domain	= String holding pretty version of url (“www.website.com”)
name	= The tab title of the url (“Welcome to website”)
icon	= String of Html code which fetches the pages favicon
checkboxVal	= Boolean value of the checkbox in the table
counter	= Amount of times this page has been intercepted

BlockedSiteList

-- Class -- “./classes/BlockedSiteList.js”

|> "./classes/BlockedSite.js"

|> "./constants.js"

Basic Functionality:

Contains a list of blockedSites and all the functionality we want to have on a list of these items. It makes sure all elements are unique, has the ability to add items, remove items and also filter on the checkboxValue. Furthermore it can also map function like getting only the urls, which is used by the webRequest listener.

*Variables**:***

List	= A unique array of blockedSites

UserSettings

-- Class -- “./classes/UserSettings.js”

|> "./constants.js"

|> ”./modules/synchronizer.js”

Basic Functionality:

This object contains the settings about the extension’s behaviour that is specified by the user. It knows whether the interception-behaviour is turned On or Off and for how long. This is all found in the status object. It also holds the mode and the interval in minutes that should occur between interceptions. Besides holding all this data the object is responsible for turning the interception On and Off.

*Variables**:***

status	= Object having time at which we set state, until when and the state itself.
sessionID	= sessionID holding the user’s login ID, (not yet used)
mode	= self explanatory
interceptionInterval	= self explanatory

synchronizer

-- Module -- “./modules/synchronizer.js”

|> "./classes/BlockedSiteList.js"

|> "./modules/storage.js"

Basic Functionality:

This module is meant as a simple way to let any “third-party-page” that changes data that needs to be synched across the extension is synched. A “third-party-page” is a page that influences data that we are trying to save and need to be persistent and saved to the background. Things like changing the UserSettings or adding a new BlockedSite to the BlockedSiteList.

Variables:

bg	= link to background page of extension

Main Flow:

Include file in third party page and call methods with “synchronizer.methodName(param)”. You call methods on the synchronizer where the parameter is the object you want to sync. So

updating the entire blacklist means passing the newBlacklist as parameter. Same goes for UserSettings.

storage

-- Module -- “./modules/storage.js”

|> "./classes/BlockedSite.js"
|> "./classes/BlockedSiteList.js"
|> "./classes/UserSettings.js"

Basic Functionality:
This module is meant to save and load items from the chrome.storage.sync. It will also serialize and deserialize the items it wants to save/load. That’s why it is dependant on the classes it wants to be able to store/load

Variables:
-None-

Main Flow:
Include file and call methods with “storage.methodName(param)”. The param is almost always one of the following two. It is either the item we want to store, or a callback-function that takes as only parameter the object you want to load from storage. Standard format of the functions follow from having a look at the file.

urlFormatter

-- Module -- “./modules/urlFormatter.js”

|> "./constants.js"

Basic Functionality:
This module is meant to format a given non-perfect url to one that the extension can work with. You call methods on the formatter with the url you want to format as a parameter.

Variables:
url_requester = class local to the formatter. It takes care of the internet related part of the formatting.

Main Flow:
Include file to be able to pass a url and make it function for the extension. The urlFormatter takes a url and formats it for the a getRequest, which is performed by the urlRequester. This the in the end returns the url and the title of the tab. These are specialised to function as input for creating a new BlockedSite item.

blockedSiteBuilder

-- Module -- “./modules/blockedSiteBuilder.js”

|> "./classes/BlockedSite.js"

|> "./classes/urlFormatter.js"

Basic Functionality:

Does mostly as the name suggests. Based on a non-formatted url and a callback function takes the newUrl through the urlFormatter and goes on to create a new blockedSite item. This items is then passed as a parameter to the callback function.

*Variables**:***

dateUtil

-- Module -- “./modules/dateutil.js”

Basic Functionality:

Formatting of dates and times is done in multiple locations in the codebase. This module is meant for helping doing this.

Variables:

-None-

Main Flow:

Methods defined in this module can be called to receive date and time related strings formatted in the correct way.

exerciseTime

-- Module -- “./modules/statistics/exerciseTime.js”

Basic Functionality:

This module contains the code related to tracking the time spent on the exercise page.

Variables:

-None-

Main Flow:

The tracker module tracks the amount of time spent on the exercise page. Whenever the amount of time spent on a page needs to be updated, methods from the exerciseTime module are called.

interception

-- Module -- “./modules/statistics/interception.js”

Basic Functionality:
This module contains methods with regards to tracking information about the interceptions.

Variables:
oneDay = Integer holding the amount of milliseconds contained in one day.

Main Flow:
The interception module contains 3 methods:

calcInterceptData
Calculates the amount of interceptions in the last day, last week, last month and the total amount of interceptions.

incrementInterceptionCounter
Receives the url from the parameter, and searches the correct blockedSite item from the blockedsite list. Then the interceptioncounter for this item is incremented by 1. Also the global interceptioncounter is incremented by one.

addToInterceptDateList
Adds the current date to the interceptionDateList.

tracker

-- Module -- “./modules/statistics/tracker.js”

|> "./constants.js"

Basic Functionality:
This module contains methods with regards to tracking the amount of time spent on the exercise page.

Variables:
idle = Boolean which is true when the user is idle, and false if the user is not idle.
tabActive = String holding the url of the tab which is currently active.
zeeguuRegex = String holding the Zeeguu url regex.
activeTime = Integer holding the amount of time the user has been active on the exercise page.

Main Flow:
Every second, the activeTime is increased if the user is not idle and is currently on the Zeeguu exercisepage. Every 5 seconds it is checked if the activeTime is larger than 0, if so, the incrementTodayExerciseTime method from the exerciseTime module is called, and the activeTime variable is set to 0.
The idle-ness of the user is tracked using the chrome.idle module.

api

-- Module -- “./modules/authentication/api.js”

Basic Functionality:
This module contains methods with regards to communicating with the Zeeguu api. The api url is specified as the apiUrl variable. The http requests are always fired towards this url.

Variables:
apiUrl = String holding the url for the Zeeguu api.

Main Flow:
The api module contains 3 methods:

request
This method receives a methodtype, an url and a parameter string. The method then fires an http request to the url path specified as a parameter. The methodtype and arguments are

specified as parameters. The response of the api to which is connected is returned as a promise.

postRequest
This method fires a POST http request by calling the *request* method with the method parameter being “POST”.

getRequest
This method fires a GET http request by calling the *request* method with the method parameter being “GET”.

auth

-- Module -- “./modules/authentication/auth.js”
|> "./modules/authentication/api.js"

Basic Functionality:
This module contains methods with regards to authenticating the user. Logging in and out and validating the user is all done through this module.

Variables:
<task>Url = Strings holding the url which need to be passes to the api module in order to perform the <task>

Main Flow:
After a user has created an account at the main Zeeguu webpage, the user is able to log in into The Distraction Shield. A user can then be authenticated by calling methods from the auth module. This includes logging in(which returns a session), logging out(which destroys the session) and validating the user session(validating the session).

init

-- Background script -- “./init.js”

|> "./classes/BlockedSiteList.js"
|> "./classes/UserSettings.js"
|> "./modules/storage.js"
|> "./background.js"

Basic Functionality:
This script is responsible for 2 things. Initializing all variables upon installation of the extension. This means making sure that nothing in the storage will be null anymore when we retrieve it from there. Further it is responsible for initializing every session that the user starts

*Variables**:***

background

-- Background script -- “./background.js”

|> "./classes/BlockedSiteList.js"
|> "./classes/UserSettings.js"
|> "./modules/blockedSiteBuilder.js"
|> "./modules/storage.js"

Basic Functionality:
This script is responsible for the main work of the background script. It holds the all the variables local to the background scripts and also the functions to update them from the storage or from a passed item. Furthermore it contains the logic for the interception that the extension performs. It is able to add, remove and replace the webRequestListeners.

Variables:

blockedSites	= BlockedSiteList holding the blacklist of the extension
interceptDateList	= Array that holds all the times that we intercepted
localSettings	= Local variant of the UserSettings object

loginPage

-- Folder -- “./loginPage/”

|> "./dependencies/bootstrap/js/bootstrap-min.js"
|> "./dependencies/jquery/jquery-1.10.2.js"
|> "./constants.js"
|>"./modules/authentication/auth.js"

Basic functionality
This folder contains the files for the webpage the user needs to login to Zeeguu with. The page allows for entering email address and password, and upon submission it will send these to the zeeguu server for validation. If the validation is successful on Zeeguu’s end, a sessionID is returned which will be stored in the localsettings object through the auth object.
It is also possible to go to Zeeguu’s sign up page from here.

Main flow
The user enters their Zeeguu credentials. The function that is called when the login button is clicked sends them to the login part of the Zeeguu api and then waits the response. When the response is received and it contains a session, login was successful. Now the session is stored. If the user was sent here after they were redirected from a blacklisted site, but their session was invalid, they will be immediately sent on to the exercise and can continue as usual.

Use per file

- loginPage.html

Contains the markup of the page.

- loginPage.css

Contains the styling for the html that is needed next to the standard bootstrap styling to make the page look good.

- loginPage.js
  |> "./dependencies/bootstrap/js/bootstrap-min.js"
  |> "./dependencies/jquery/jquery-1.10.2.js"
  |> "./constants.js"
  |>"./modules/authentication/auth.js"

Adds all the functionality to the html page.

optionsPage

-- folder -- “./optionsPage”

|> "./dependencies/jquery/jquery-1.10.2.js"
|> "./dependencies/bootstrap/css/bootstrap.min.css"
|> “./modules/blockedSiteBuilder.js”
|> "./modules/synchronizer.js"
|> "./modules/storage.js"
|> "./classes/BlockedSiteList.js"
|> "./classes/BlockedSite.js"
|> "./classes/UserSettings.js"
|> "./constants.js"

Basic Functionality:

This folder contains everything for the main options page where the user can modify the extension's behaviour to make it function to their desires.
The user can specify:

- The interception mode
- The interval between interceptions
- add/edit/delete items in the blacklist
- Turn the extension's interception off for the desired time
- Turn the extension's interception back on

Furthermore the user can look at the general interception counter and click on a link to continue to the statistics.

Variables:

blacklistTable	= Object of type BlacklistTable
intervalSlider	= Object of type GreenToRedSlider
turnOffSlider	= Object of type TurnOffSlider
settings_object	= Object of type UserSettings
blacklist	= Object of type BlockedSiteList
interceptionCounter	= amount of interceptions since installation

Main Flow:
User opens the options-page
The options.js “initOptionsPage”-function is called. This first gets the blacklist, settings_object and interceptionsCounter from the storage (chrome.storage.sync). After loading this data it continues to connecting the javascript code to the html-functionality. This is done through either the htmlFunctionality.js or through one of the classes we wrote (BlacklistTable.js, GreenToRedSlider.js, TurnOffSlider.js). Finally it continues to connect the data from the now local variables to the html, to actually give back a visual representation.
This is where step 2 starts, the use of the page. Everytime the user interacts with the page and changes a setting, it is send through the synchronizer to make these changes sync with the background scripts and the storage. Updating settings consists of 3 parts, first we update the local variables. Then the html elements, then we sync it across the extension.

Use per file:

- options.html
    |> "./dependencies/bootstrap/css/bootstrap.min.css"

Define all the html for this page using bootstrap. The only html not in here is generated by user behaviour and for now specified in one of the other files.

- options.css

Contains some final positioning of elements on the page. The things where bootstrap was not all it took.

- options.js
    |> "./dependencies/jquery/jquery-1.10.2.js"
    |> "./modules/synchronizer.js"
    |> "./modules/storage.js"
    |> "./classes/BlockedSiteList.js"
    |> "./classes/UserSettings.js"
    |> "./optionsPage/htmlFunctionality.js"
    |> "./optionsPage/connectDataToHtml.js"
    |> "./optionsPage/classes/TurnOffSlider.js"
    |> "./optionsPage/classes/BlacklistTable.js"

Functions as the main body for all the scripts. This script takes care of initialization of the options page and of further manipulation of its local variables.

- connectDataToHtml.js
    |> "./dependencies/jquery/jquery-1.10.2.js"
    |> "./classes/BlockedSiteList.js"
    |> "./classes/BlockedSite.js"
    |> "./optionsPage/classes/BlacklistTable.js"
    |> "./optionsPage/classes/GreenToRedSlider.js"

Functions that only connect the html-representation of the local variables.

- htmlFunctionality.js
    |> “./modules/blockedSiteBuilder.js”
    |> "./modules/synchronizer.js"
    |> "./classes/UserSettings.js"
    |> "./constants.js"
    |> "./optionsPage/options.js"
    |> "./optionsPage/classes/BlacklistTable.js"
    |> "./optionsPage/classes/GreenToRedSlider.js"

Functions that define the behaviour of all the html elements. How to respond and the functions that the event listeners call afterwards.

- classes/BlacklistTable.js
    |> "./dependencies/jquery/jquery-1.10.2.js"
    |> "./modules/urlFormatter.js"
    |> "./modules/synchronizer.js"
    |> "./classes/BlockedSite.js"

The dynamically updating table that holds the blacklist of blockedSites. This is a special Html-table with some added behaviour as to what to do with the data and how to respond to the user’s input.

- classes/GreenToRedSlider.js
    |> "./dependencies/jquery/jquery-1.10.2.js"

A custom class that requires a certain structure of html to be predefined. Then we can just call this function with that html element and a custom function as to what we want to do with the value of the slider. This makes is really reusable. It is used to set the interval between intercepts.

- classes/TurnOffSlider.js
    |> "./dependencies/jquery/jquery-1.10.2.js"
    |> "./classes/UserSettings.js"
    |> "./constants.js"
    |> "./optionsPage/classes/GreenToRedSlider.js"
    |> "./optionsPage/htmlFunctionality.js"

Special Case of the GreenToRedSlider with even more functionality and a little more specific. It is used for turning the extension off for certain amounts of time and back on again.

statisticsPage

-- folder -- “./statisticsPage”

|> "./dependencies/jquery/jquery-1.10.2.js"
|> "./dependencies/bootstrap/css/bootstrap.min.css"
|> "./modules/storage.js"
|> "./constants.js"
|> "./classes/BlacklistStatsTable.js"
|> "./classes/InterceptionCounterTable.js"
|> "./classes/ExerciseTimeTable.js"
|> "./constants.js"

Basic Functionality:
This folder contains everything for the main statistics page where the user can see statistics tracked by the extension.
The user can see:

- The amount of interceptions in the recent history.
- The amount of interceptions per blocked site.
- The amount of time spent on exercises each day.

Variables:

interceptionCounterTable	= Object of type interceptionCounterTable
blacklistTable	= Object of type BlacklistTable
exerciseTimeTable	= Object of type exerciseTimeTable

Main Flow:
The statistics.js “initStatistics”-function is called. This first gets the interceptionDateList and the exerciseTimeList from the storage. After loading these, the data is connected to the table classes. These classes are then able to render the table unto the html page.

Use per file:

- statistics.html
  |> "./dependencies/bootstrap/css/bootstrap.min.css"

Define all the html for this page using bootstrap. The only html not in here is generated by user behaviour and for now specified in one of the other files.

- statistics.css

Contains css code for styling the statistics page.

- statistics.js
  |> "./dependencies/jquery/jquery-1.10.2.js"
  |> "./modules/storage.js"
  |> "./statisticsPage/classes/BlacklistStatsTable.js"
  |> "./statisticsPage/classes/ExerciseTimeTable.js"
  |> "./statisticsPage/classes/InterceptionCounterTable.js"

Functions as the main body for all the scripts. This script takes care of initialization of the statistics page and of further manipulation of its local variables.

- classes/BlacklistStatsTable.js
  |> "./dependencies/jquery/jquery-1.10.2.js"

The table that holds the blacklist of blockedSites. Data can be loaded into this class using the setData method, and the table can be rendered using the render method.

- classes/ExerciseTimeTable.js
  |> "./dependencies/jquery/jquery-1.10.2.js"

The table that holds the table containing the time spent on exercises each day. Data can be loaded into this class using the setData method, and the table can be rendered using the render method.

- classes/InterceptionCounterTable.js
  |> "./dependencies/jquery/jquery-1.10.2.js"

The table that holds the table with the amount of interceptions in the recent history. Data can be loaded into this class using the setData method, and the table can be rendered using the render method.

introTour

--folder-- “./introTour”

|>”./dependencies/bootstrap-tour-standalone.min.css”

|>”./dependencies/bootstrap-tour-standalone.min.js”

|>”./dependencies/bootstrap-tour.min.js”

Basic Functionality:

This folder contains everything needed to implement the introduction tour that will run upon installation.

Variables:

id = integer that holds the id of the tab that the tour is running on

Main Flow:

Upon installation the user is given a tour of the extension and it’s functionality. The user can navigate through the tour by clicking the next button, or can choose the end the tour at any time by pressing the end tour button. Once the tour has ended, the user will be taken to the login page.

Use per file:

- introTour.js

Contains the javascript to run the tour. Contains a few other functions that handle ending of the tour.

- introTour.css

Contains some styling for introTour.html

- introTour.html

Contains html for the tour landing page.

- loginCopy.html
- tooltipCopy.html
- optionsCopy.html

These 3 html files are for loading a copy of each of the respective pages for use in the tour. We use copies of the actual pages, as we do not want to run our javascript on the live pages, as this can result in bugs when undertaking the tour.

tooltipPage

-- folder -- “./tooltipPage”

|> "./dependencies/jquery/jquery-1.10.2.js"

|> "./dependencies/bootstrap/css/bootstrap.min.css"

Basic Functionality:

This folder contains everything for the little tooltip page that shows up when the user clicks the icon of the extension to the right of the URL-bar. The user can quickly navigate to one of the extension-related pages and also add the current page to their blacklist.

Variables:

auth	= The authentication object of the bg-script.
bg	= The background of the extension. Gives access to its functions

Main Flow:

User clicks the icon to the right of the address bar.

The user clicks one of the four buttons and then either the user is redirected to the page relating to that button or the current page is added to the blacklist.

Use per file:

- tooltip.html

  |> "./dependencies/bootstrap/css/bootstrap.min.css"

Define all the html for this page using bootstrap.

- tooltip.css

Contains some final positioning of elements on the page. The things where bootstrap was not all it took.

- tooltip.js

contentInjection

- folder -- “./contentInjection”

Basic Functionality:
This folder contains the code regarding the injection into the Zeeguu page. In order to notify the user that he has been intercepted and redirected to the Zeeguu page, we inject into the page a notification. This notification is personalized based on the user settings.

Variables:

mainFlow	=	the function that starts the main flow and sets the mode
initBasis	=	the function that initializes the basic flow, regardless of the mode
getDest	=	the function that retrieves the original destination of the user

Use per file:

- inject.html:
    - Represents the html code that is injected into the Zeeguu page, namely a panel that describes why the user was intercepted
- inject.js
    - Represents the JavaScript code that injects the “inject.html” and personalise it based on the mode that the user selected.