HANYANG UNIVERCITY

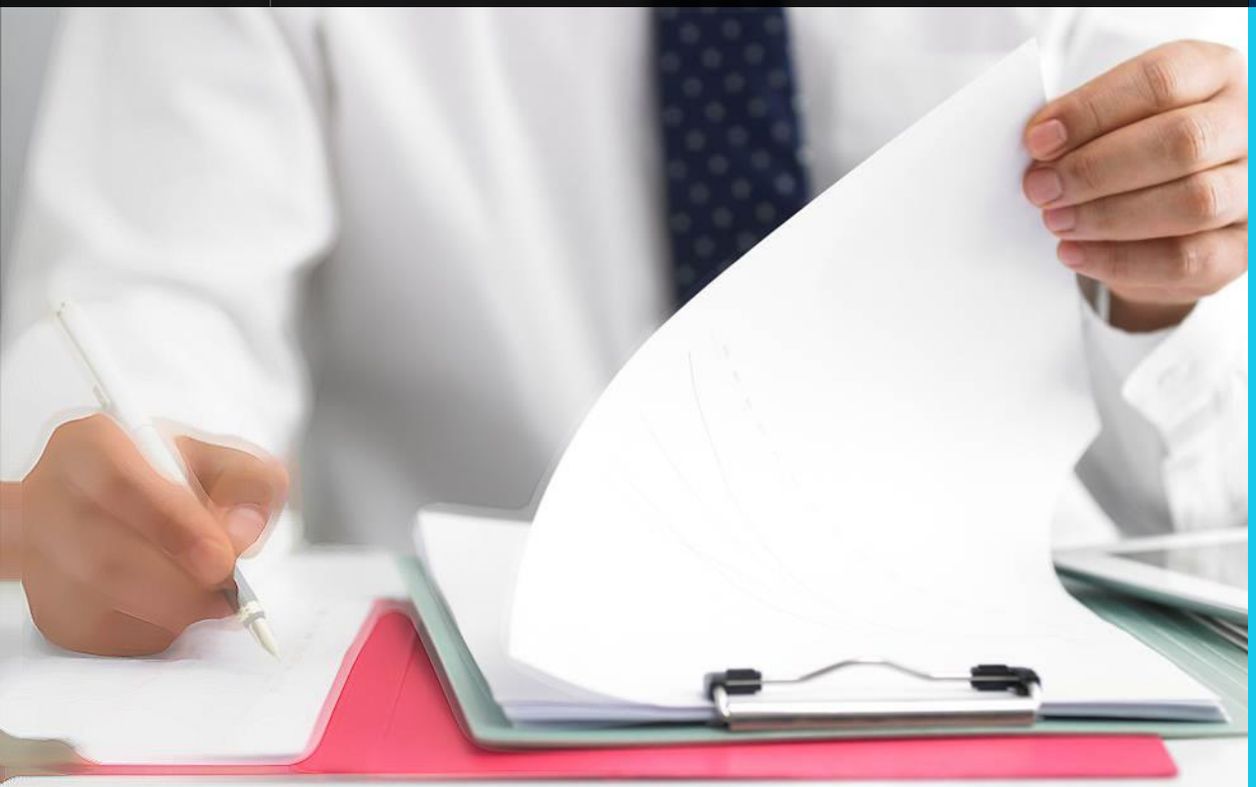
Web System Programming

한양대학교
HANYANG UNIVERSITY

Web System Programming

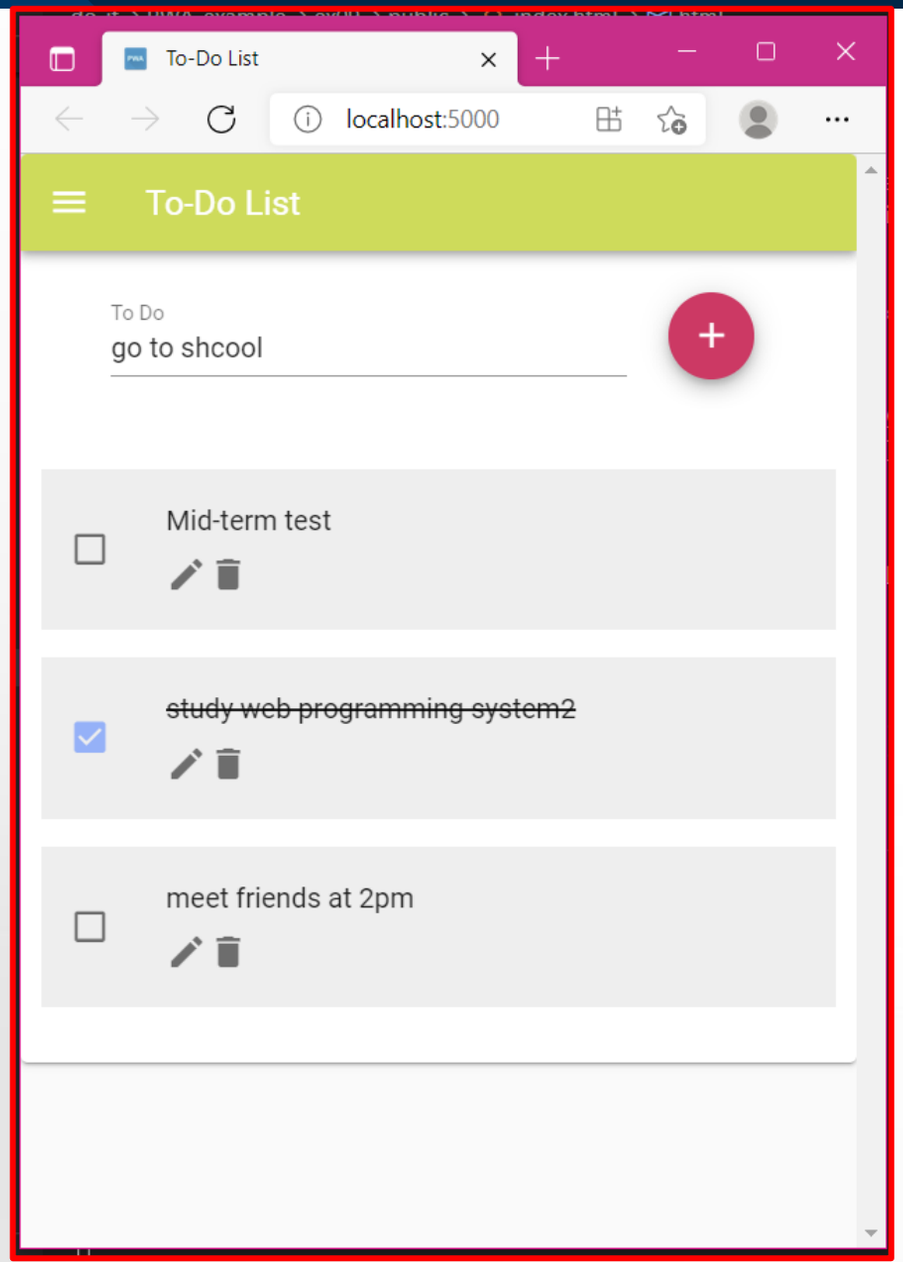
Create To-Do App

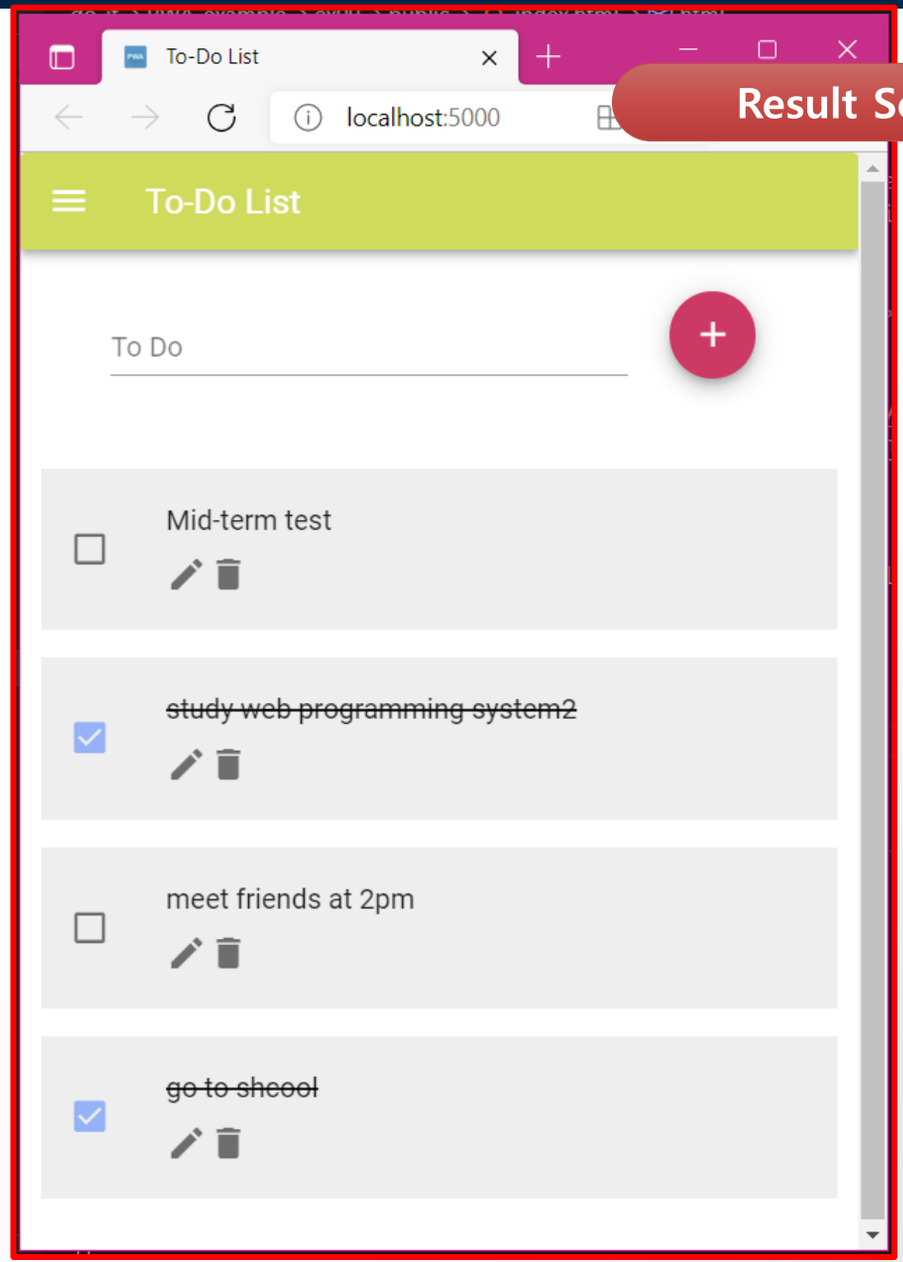
HANYANG UNIVERSITY



1. What is the To-Do App?
2. Writing the manifest
3. Offline Management by Workbox
4. Preparing Firebase Realtime DB
5. Create app launch screen

1. What is the To-Do App?





Result Screen

2. Writing the manifest

Vuefire https://vuefire.vuejs.org/ Vuefire Vuexfire API Cookbook GitHub

Vuefire

Realtime bindings between Vue/Vuex and Firebase

Get Started →

Note: This version currently supports Vue 2 and Firebase 7. Support for Vue 3 / Composition API and Firebase 8 is on the way.

2. Writing the manifest

Installing Vuefire

```
PS C:\Users\Angela\Documents\do-it\PWA-example\ex09> npm install firebase vuefire@next
```

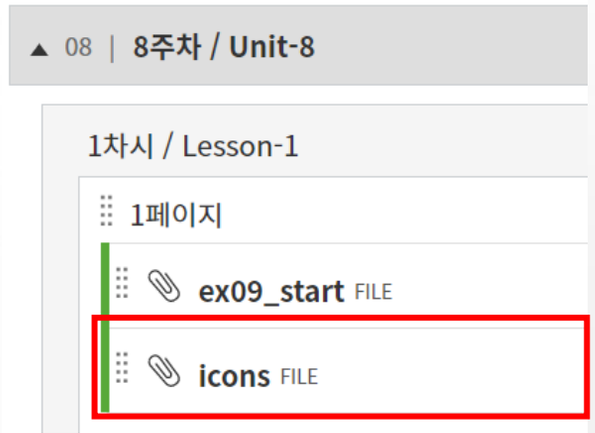
npm install firebase vuefire@next

```
PS C:\Users\Angela\Documents\do-it\PWA-example\ex09> npm install firebase@6.6.2 vuefire@2.2.0-alpha.0
```

npm install firebase@6.6.2 vuefire@2.2.0-alpha.0

Insert Image file

Copy image to path /public/img/icon



2. Writing the manifest

```
{
  "name": "To-Do List",
  "short_name": "To-Do List",
  "icons": [
    {
      "src": "./img/icons/android-chrome-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "./img/icons/android-chrome-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "start_url": "./index.html",
  "display": "standalone",
  "orientation": "portrait",
  "background_color": "#ffffff",
  "theme_color": "#ffffff"
}
```

Modify manifest.js

2. Writing the manifest

Modify manifest.js

- Title and description settings: In the name, short_name fields, enter a title and description that indicates the characteristics of the PWA.

```
"name": "To-Do List",
"short_name": "To-Do List",
```

- Display setting: display is set to standalone and set to the maximum screen that can be used on the device. And orientation runs as portrait, which can be viewed vertically.

```
"display": "standalone",
"orientation": "portrait",
```

2. Writing the manifest

Modify manifest.js

- Set background color and theme color: theme_color determines the color of the status bar. background_color determines the background color of the splash screen. All white was used to give the design a sense of unity. Just enter the color you want.

```
"start_url": "./index.html",
"display": "standalone",
"orientation": "portrait",
"background_color": "#ffffff",
"theme_color": "#ffffff"
```

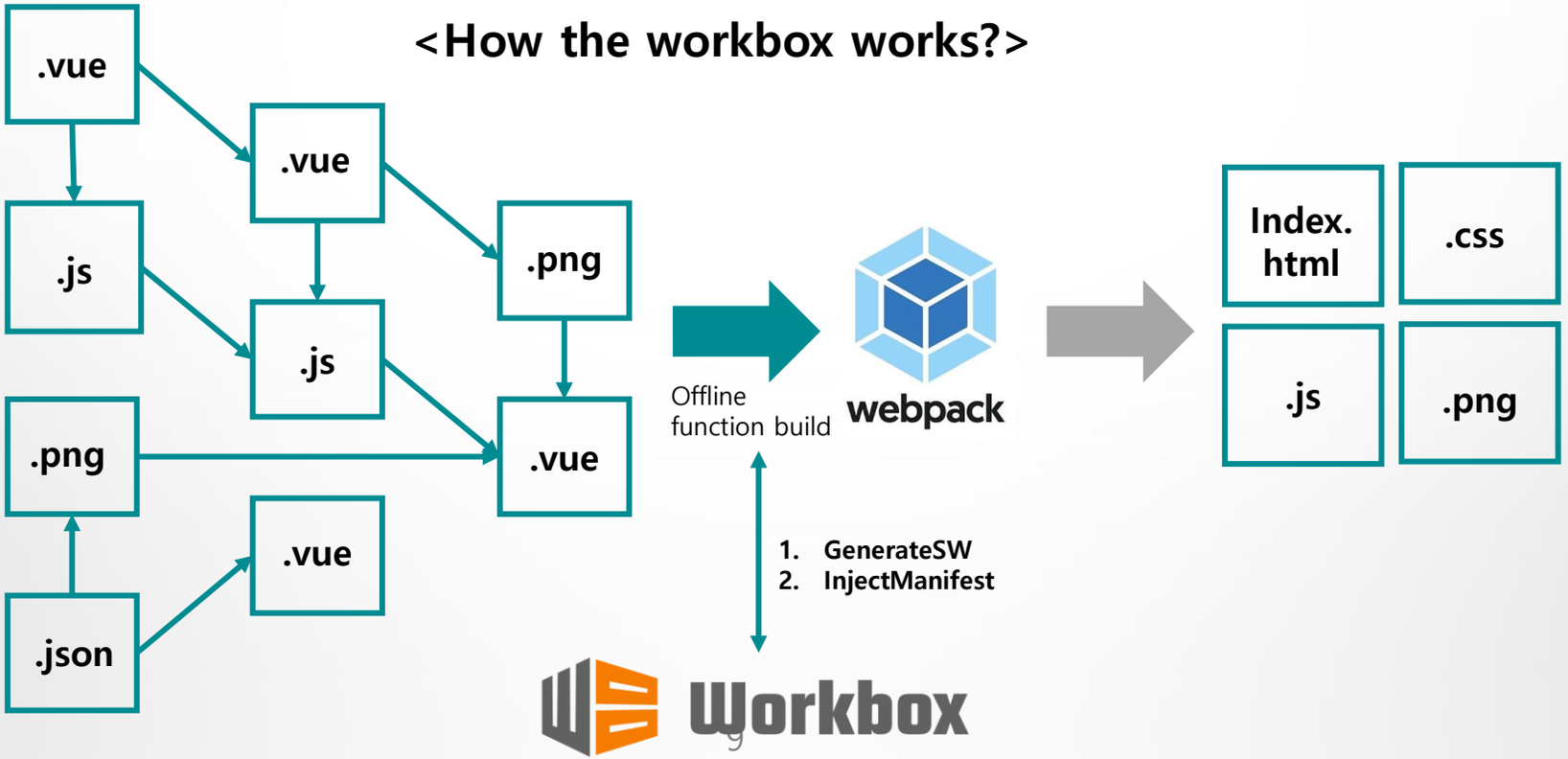
#ffffff = White color

3. Offline Management by Workbox : What is the Workbox Plug-in Mode?

<Workbox plugin mode>

Plug-In Mode	Description
GenerateSW	Assign workbox options to automatically created service workers
InjectManifest	Put the code directly into the service worker to create the final service worker file

<How the workbox works?>



3. Offline Management by Workbox : What is the Workbox Plug-in Mode?

Workbox Plug-in Mode

- The cache can be divided into pre-cache, which is specified in advance before execution, and runtime-cache, which specifies only the desired part when the program is executed.
- If you want to handle these two options by specifying simple settings, use the GenerateSW plugin mode. But if you want to put cache policy and program logic yourself, use InjectManifest plugin mode.

<How the workbox works?>

Plug-In Mode	Situation to Use	Limitations
GenerateSW	• When you need to quickly specify files to pre-cache • When you need a simple runtime-cache	• When you need to put your code in a service worker • When to use web push notifications
InjectManifest	• When you need to put your own code in the service worker • When you want to specify pre-cache and runtime-cache directly in code • When you want to use web push notifications	• It can get complicated so GenerateSW may be suitable for simple times

3. Offline Management by Workbox : How to use plugin mode?

GenerateSW mode

- The vue.config.js file is required to use the workbox's two plugin modes. vue.config.js is a configuration file that is located in the root folder of a Vue project and allows you to specify various Vue-CLI plugin options needed to run the program.
- This file does not exist when the template is first created, so you have to create it and add it yourself.
- First, let's look at an example of creating the GenerateSW plugin mode.

```
module.exports = {
    pwa: {
        workboxPluginMode : 'GenerateSW',
        workboxOption : {
            runtimeCaching : [{
            }]
        }
    }
}
```

3. Offline Management by Workbox : How to use plugin mode?

What is the Pre-cahce?

- Pronounced "pre-cashing," it refers to software that downloads data ahead of time in anticipation of its use. For example, when a Web page is retrieved, the pages that users typically jump to when they leave that page might be precached in anticipation.

<Pre-cache option>

Option	Means	Examples
include	Specifies the file to be used in the pre-cache. Since this is based on the file name, it is useful to use a regular expression that searches in bulk.	`include: [ /\.css$/, /\.js$/]`
exclude	Specifies the files to be removed from the pre-cache. How to use: Use regular expressions together like include. However, it should be noted that even if there is no file to be removed, it must be specified like exclude: [] for proper operation.	`exclude: [ /\.json$/, /\.png$/]`

3. Offline Management by Workbox : How to use plugin mode?

What is the Regular Expression?

- Regular expressions are specially encoded text strings used as patterns for matching sets of strings. They began to emerge in the 1940s as a way to describe regular languages, but they really began to show up in the programming world during the 1970s.
- Regular expressions later became an important part of the tool suite that emerged from the Unix operating system—the ed, sed and vi (vim) editors, grep, AWK, among others. But the ways in which regular expressions were implemented were not always so regular.
- "A regular expression is a pattern which specifies a set of strings of characters; it is said to match certain strings." —Ken Thompson

3. Offline Management by Workbox : How to use plugin mode?

What is the Regular Expression?

<Special symbols used in regular expressions>

Special Characters	Means	Examples
^	Search only those that must start with the letter following "^"	/^hi/ finds only those beginning with 'hi', so hi, hihello, etc. can be returned
$	Search only those ending with the character before "$"	/hi$/ finds only those ending in 'hi', so hi, hellohi, etc. can be returned
.	"." Search for whatever comes in place	/hi.css/ matches any character between 'hi' and 'css', so hi.css, hi_css, hi-css, etc. may be returned
₩	The character following "₩" is considered a regular symbol, not a special symbol	/hi₩.css/ must have a dot (.) between 'hi' and 'css', so hi.css, hellohi.css, etc. may be returned.

3. Offline Management by Workbox : How to use plugin mode?

InjectManifest Mode

- To use the InjectManifest mode, you must first declare the workboxPluginMode option.

```
module.exports = {
    pwa: {
        // InjectManifest plugin mode must be declared
        workboxPluginMode : 'InjectManifest',
        workboxOptions : {
            //Service worker file must be specified in InjectManifest mode
            swSrc : 'src/serviceworker.js',
        }
    }
}
```

3. Offline Management by Workbox : Create a vue.config.js file

Create veu.config.js

```
module.exports = {
    pwa: {
      workboxOptions: {
        // Specifying files to pre-cache
        include: [/^index\.html$/, /\.css$/, /\.js$/,
 /^manifest\.json$/, /\.png$/],
        // Exclude must be entered for normal operation
        exclude: []
      }
    }
}
```

3. Offline Management by Workbox : Create a vue.config.js file

vue.config.js

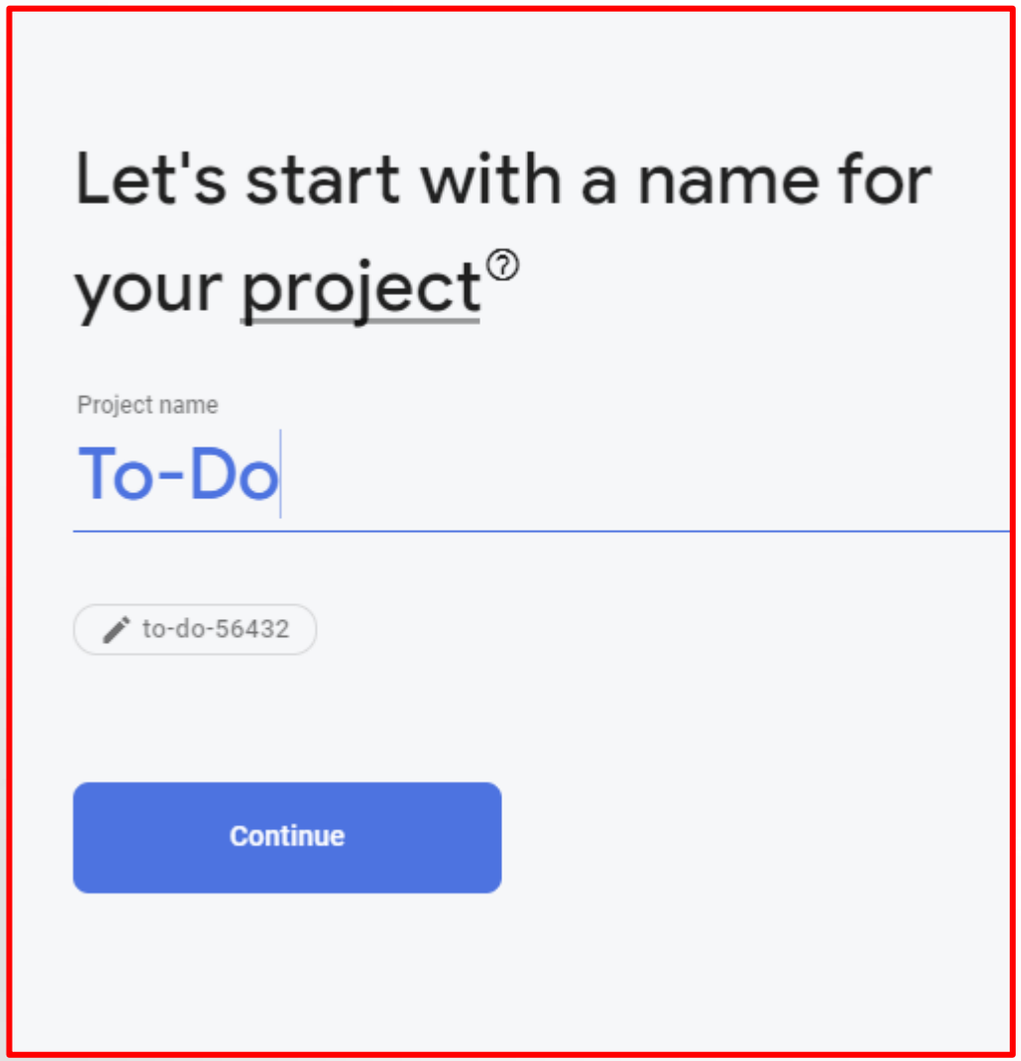
- Specifying files to pre-cache with include: The include option can specify static files to cache through pattern characters. Here, we specified that index.html, *.css, *.js, manifest.json, and *.png should be pre-cache.

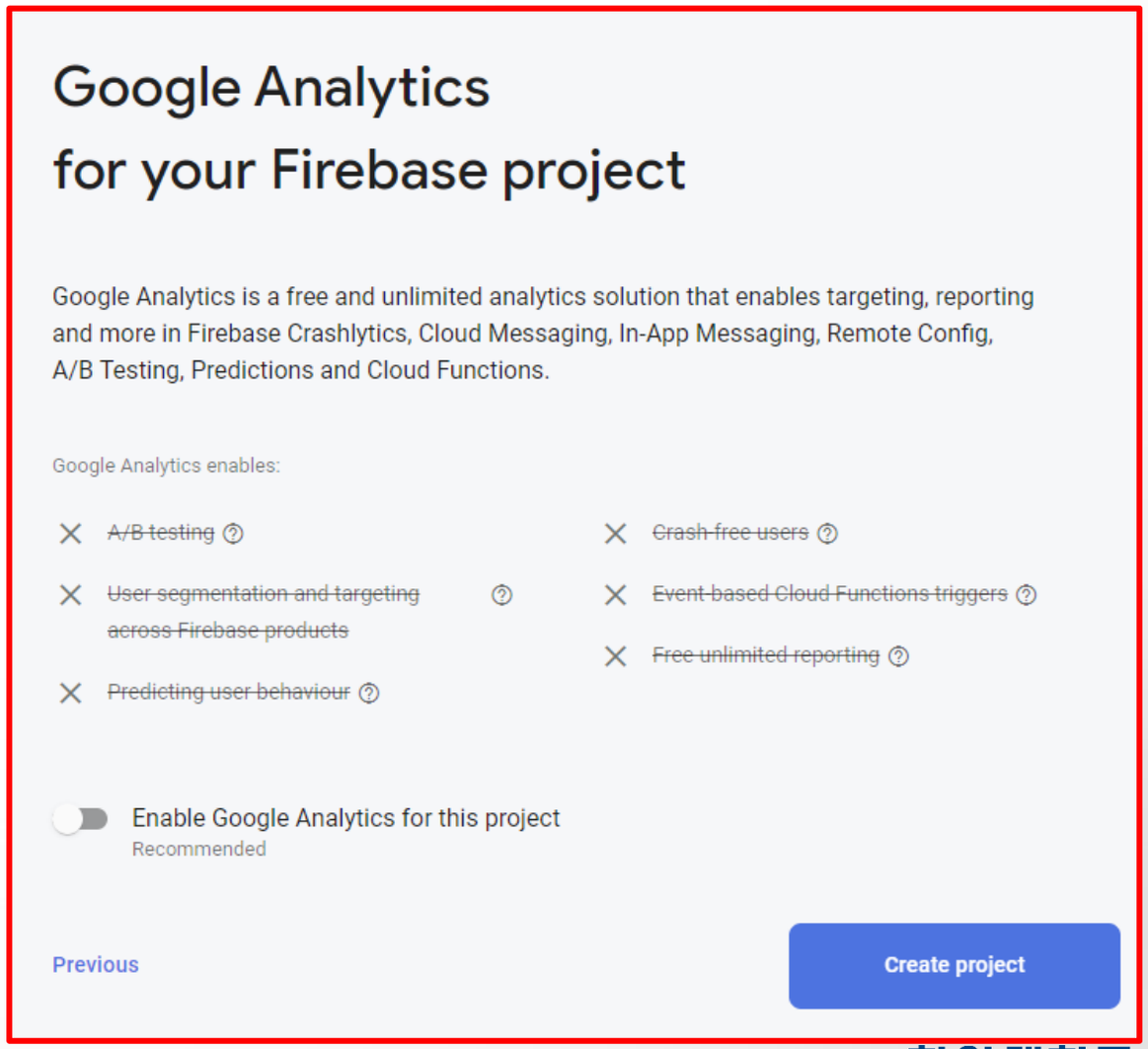
```
            include: [/^index\.html$/, /\.css$/, /\.js$/,
/^manifest\.json$/, /\.png$/],
```

- Specify the files to exclude from the pre-cache with exclude : You can use exclude to specify the files to be removed from the pre-cache. Note that when using this option, you must write the exclude option even if there are no files to remove. You have to be careful, otherwise it won't be cached the way you want it to.

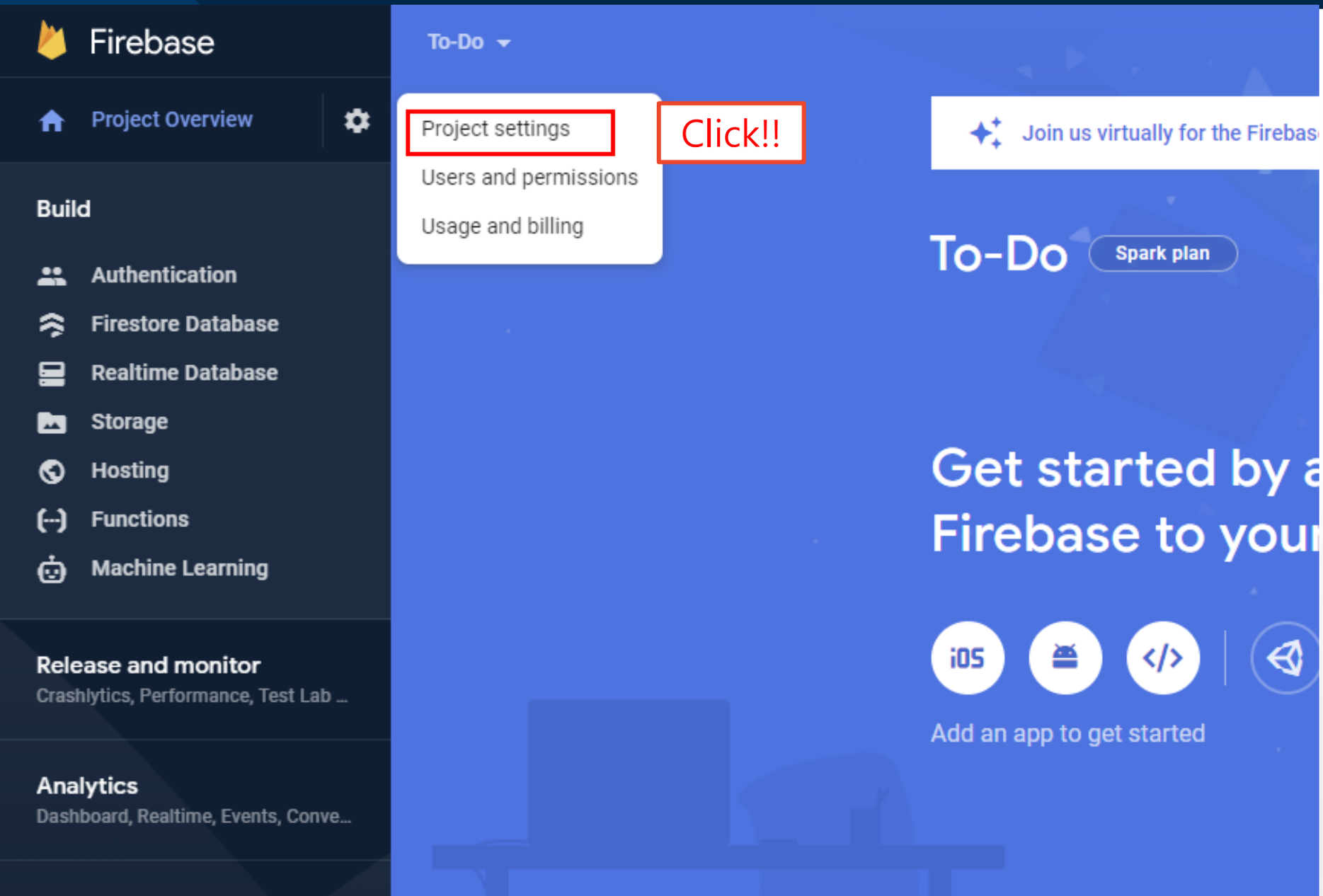
```
// Exclude must be entered for normal operation
exclude: []
```

4. Preparing Firebase Realtime DB : Create a firebase project

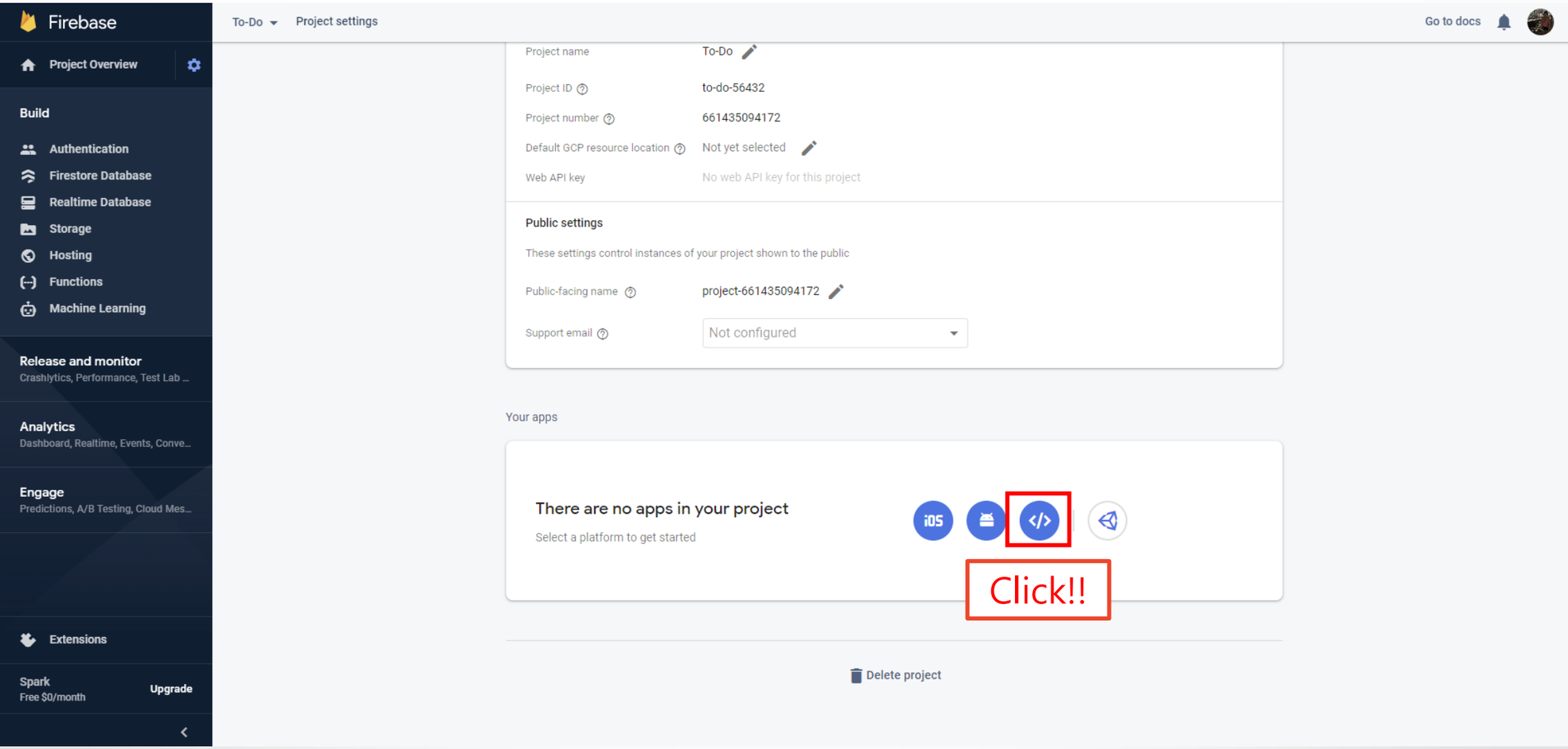




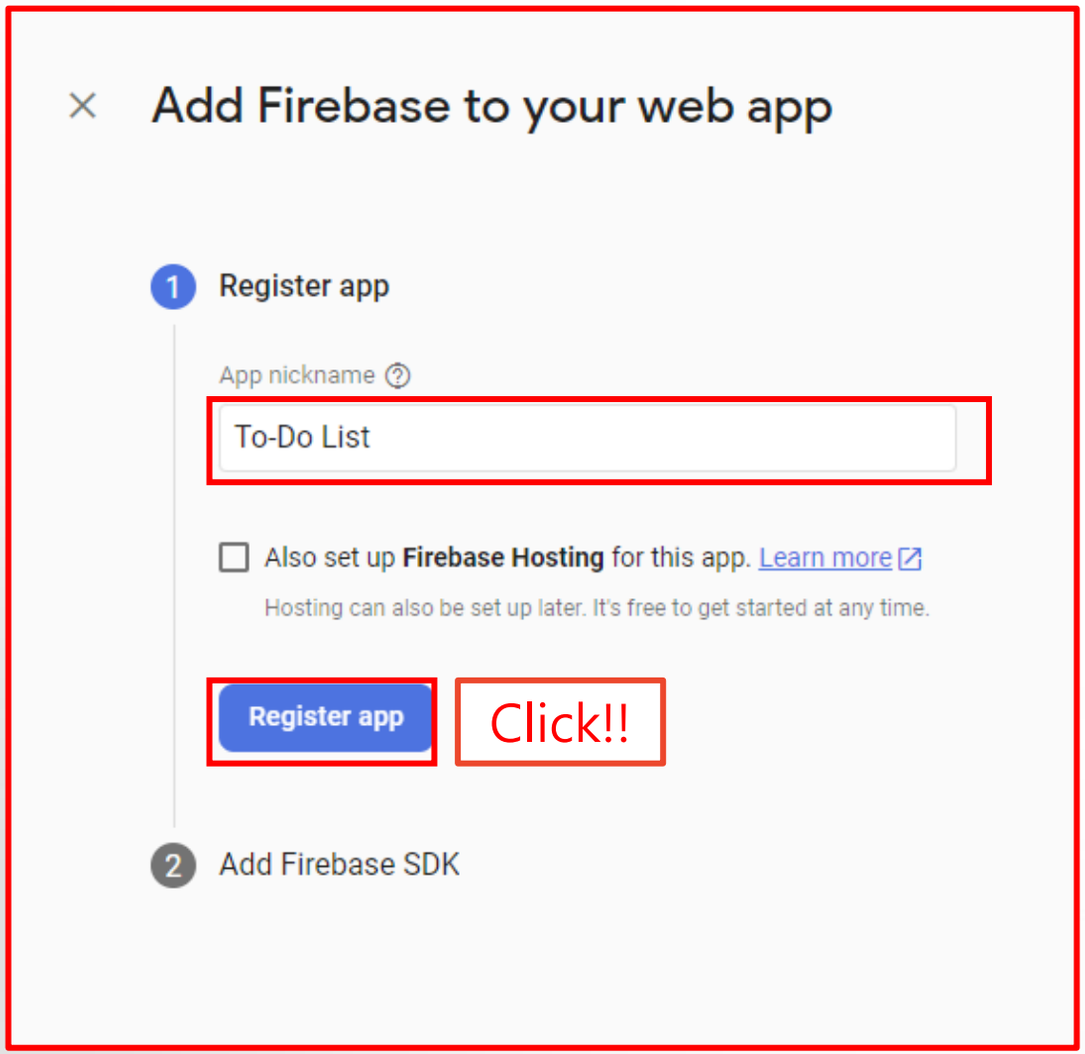
4. Preparing Firebase Realtime DB : Create a firebase project

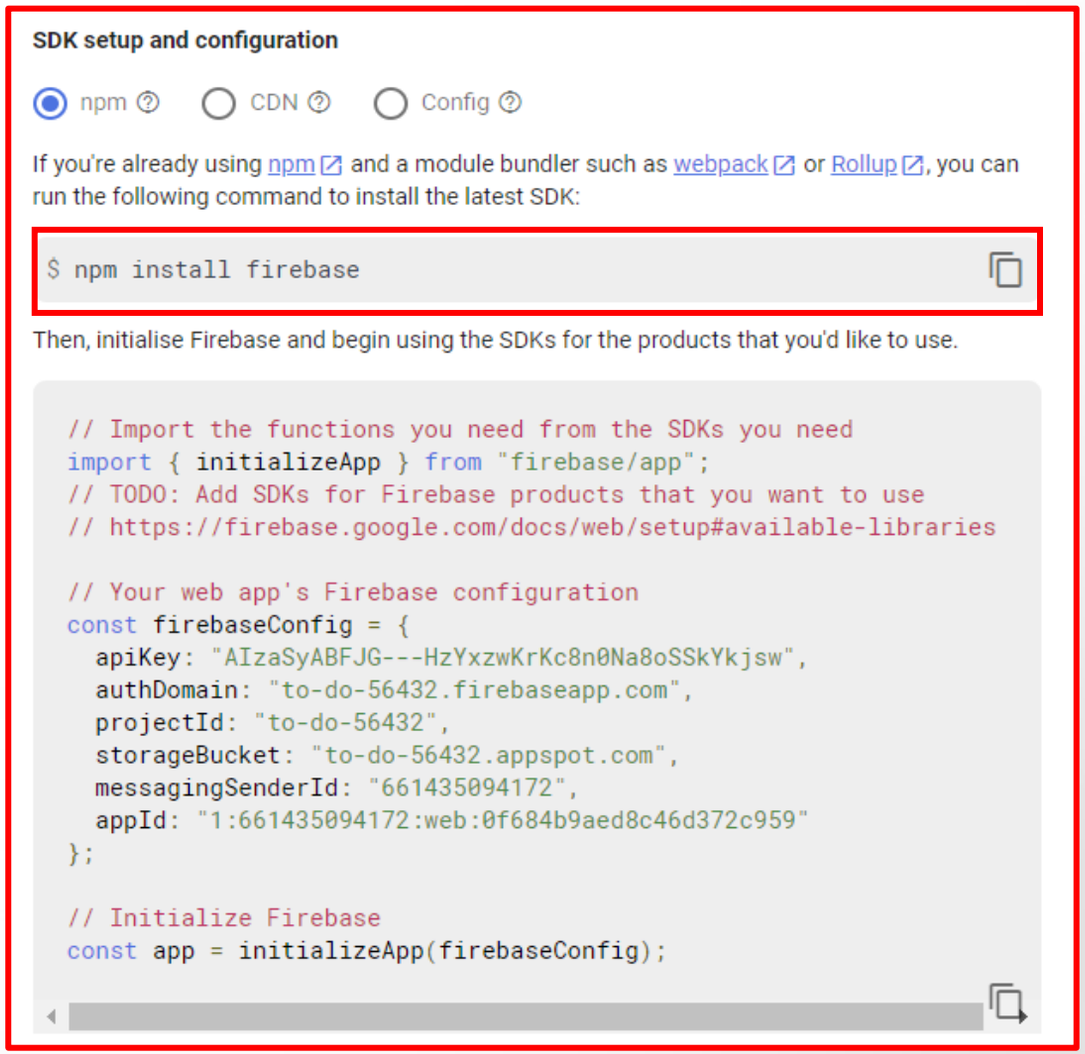


4. Preparing Firebase Realtime DB : Create a firebase project

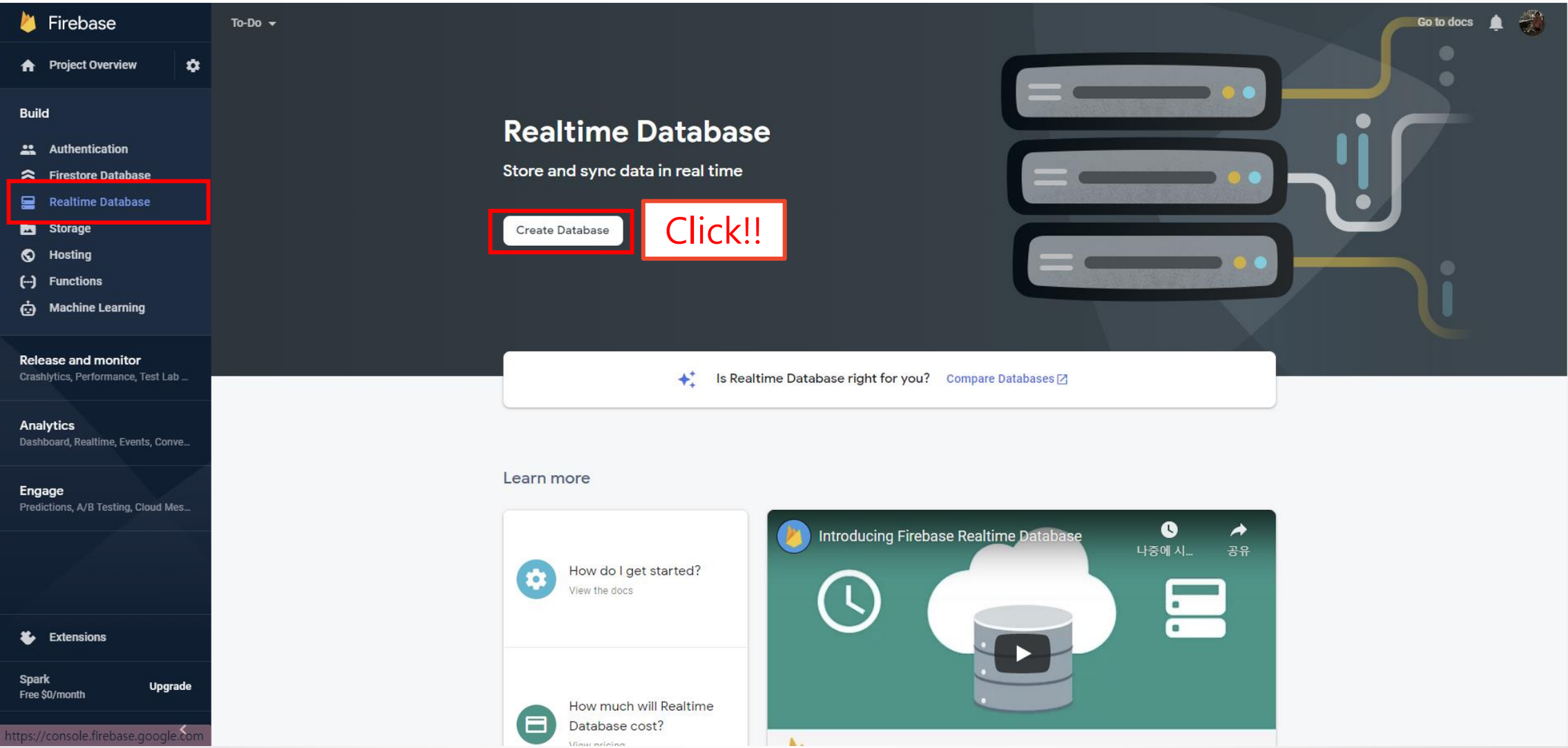


4. Preparing Firebase Realtime DB : Create a firebase project

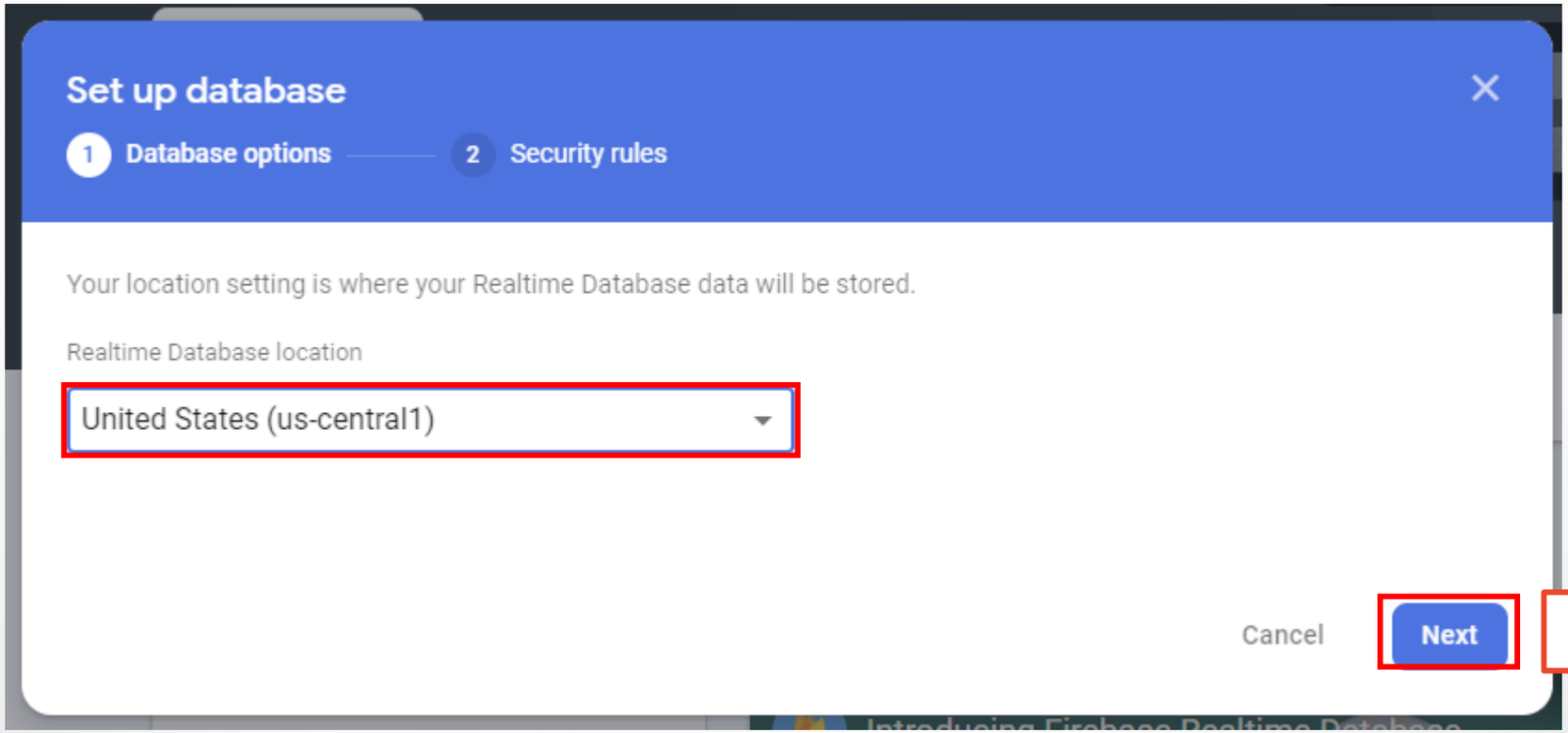




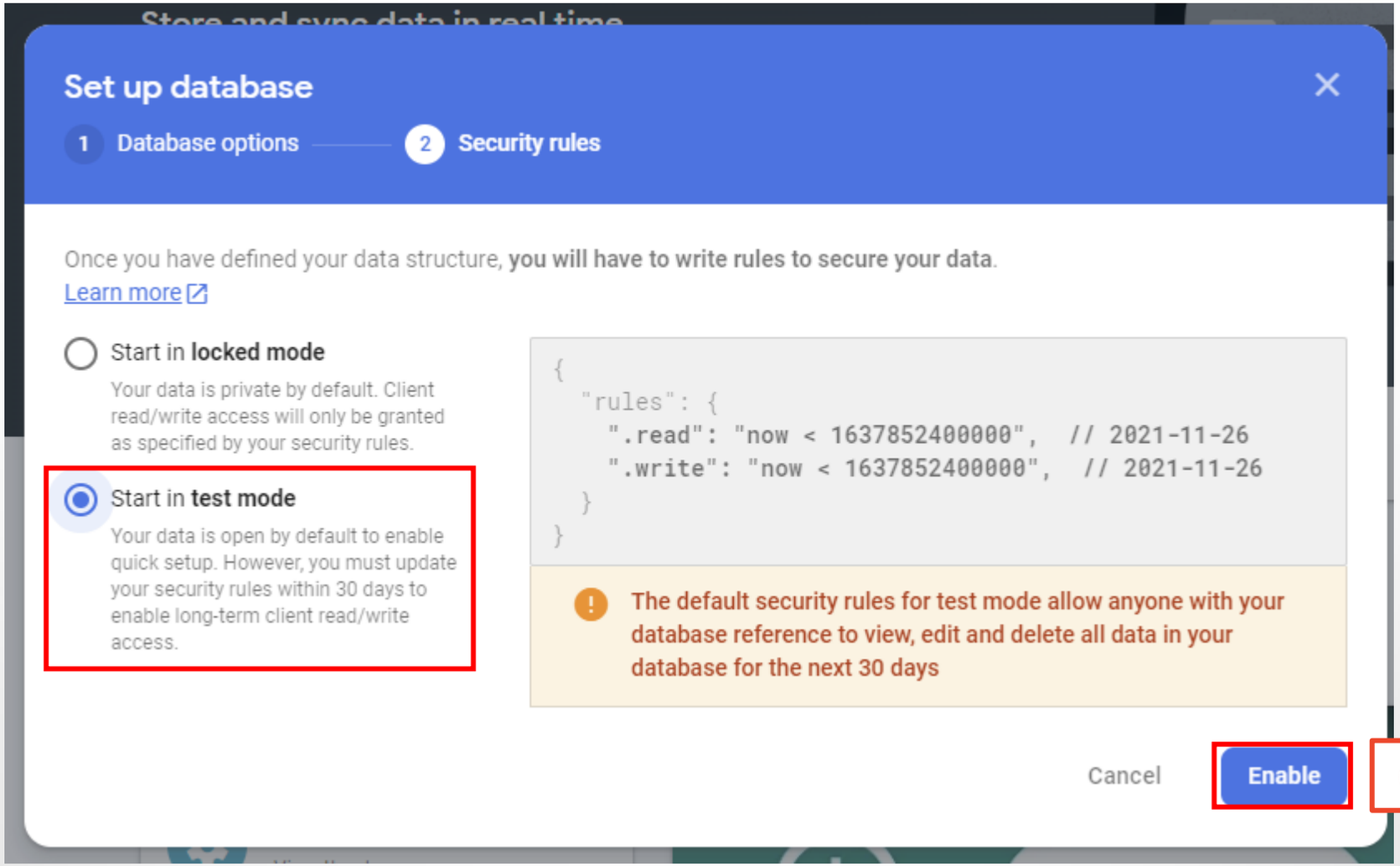
4. Preparing Firebase Realtime DB : Create a firebase project



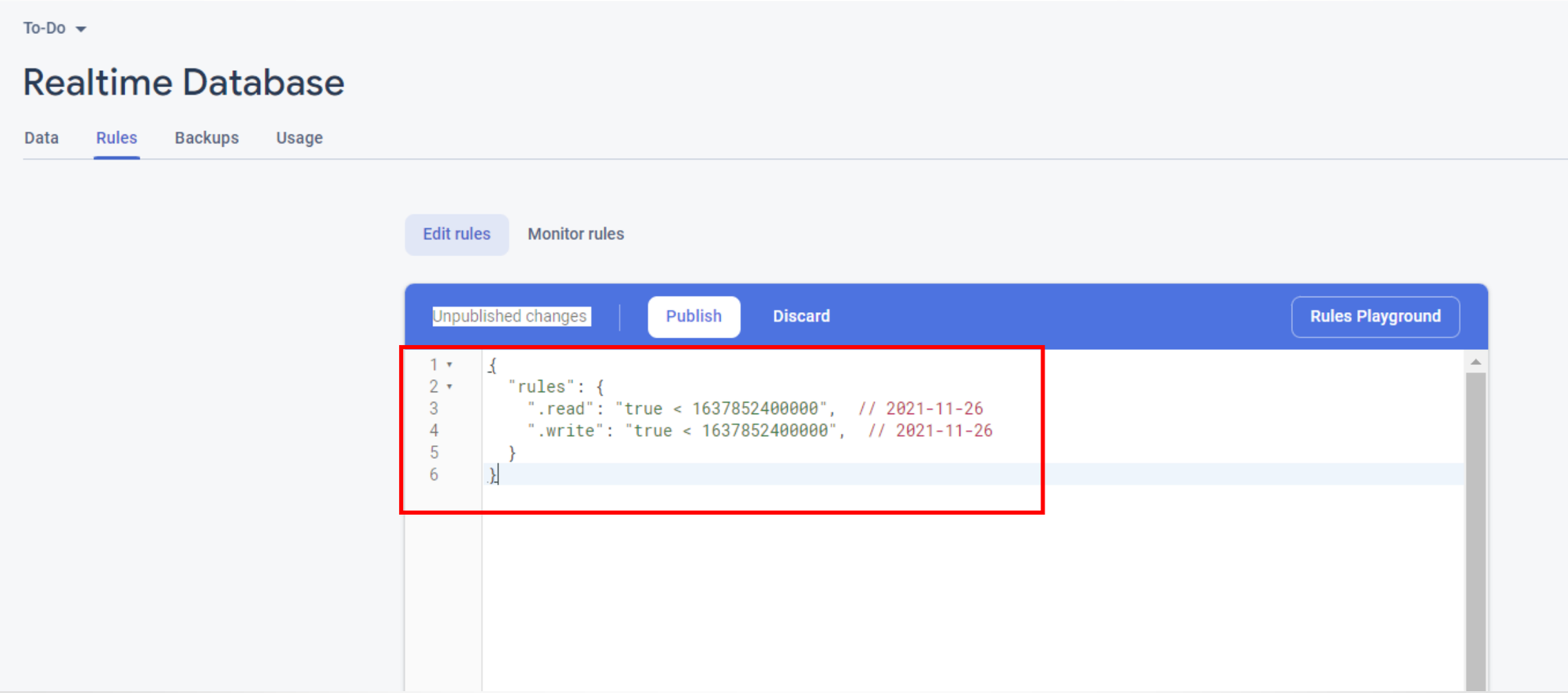
4. Preparing Firebase Realtime DB : Create a firebase project



4. Preparing Firebase Realtime DB : Create a firebase project

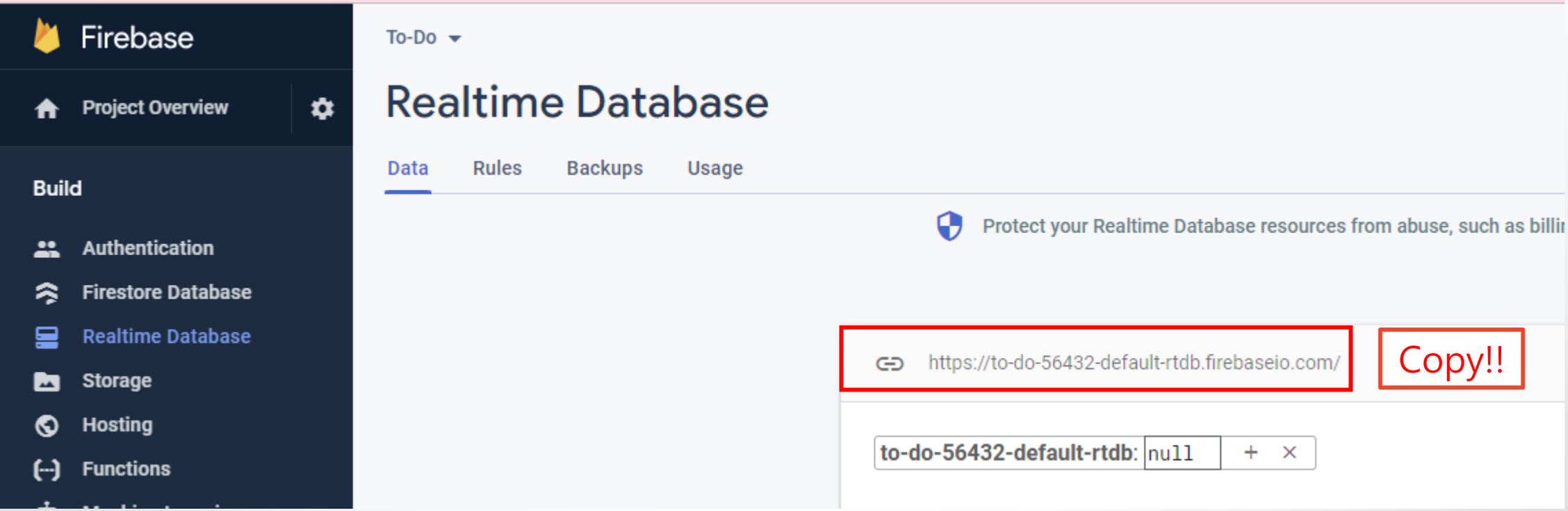


4. Preparing Firebase Realtime DB : Create a firebase project



4. Preparing Firebase Realtime DB : Create a firebase project

Modify firebase.js



4. Preparing Firebase Realtime DB : Create a firebase project

Modify firebase.js

```
1   // Import Firebase App Object Module
2   import firebase from 'firebase/app'
3   // Import firebase package module
4   import 'firebase/firebase-database';
5
6   // Initialize Firebase DB and connect
7   const oDB = firebase.initializeApp({
8     // Copy and paste from the Firebase console
9     //databaseURL: "https://pwa-to-do.firebaseio.com",
10    databaseURL: "https://to-do-56432-default-rtdb.firebaseio.com/",
11  }).database();
12
13  // Among the Firebase DB objects, the todos item is released for use elsewhere
14  export const oTodosinDB = oDB.ref('todos');
15
```

4. Preparing Firebase Realtime DB : Create a firebase project

Modify firebase.js

- Import Firebase Module: To use Firebase, you need to import two package files named app and firebase-database from the Firebase folder, respectively. Specifically, the app package is stored in an object named firebase for future use.

```
// Import Firebase App Object Module
import firebase from 'firebase/app'
// Import firebase package module
import 'firebase/firebase-database';
```

4. Preparing Firebase Realtime DB : Create a firebase project

Modify firebase.js

- Initialize and connect Firebase DB: Initialize by passing databaseURL information as a parameter to the initializeApp() function of the firebase object. The databaseURL uses the address copied earlier. And if the database() function is executed on the returned object, the final object that can access the Firebase DB is returned, which is stored in the oDB variable.

```
// Initialize Firebase DB and connect
const oDB = firebase.initializeApp({
  // Copy and paste from the Firebase console
  databaseURL: "https://to-do-56432-default-rtdb.firebaseio.com/",
}).database();
```

- Open todos item to be used elsewhere: Access todos item through ref() function in oDB and store the result in oTodosinDB object variable. At this time, the oTodosinDB variable declares export so that other files in the project can access it.

```
export const oTodosinDB = oDB.ref('todos');
```

4. Preparing Firebase Realtime DB : Create a firebase project

Modify main.js

```
1   import Vue from 'vue'
2   import App from './App.vue'
3   import './registerServiceWorker'
4   import vuetify from './plugins/vuetify';
5
6   //Import the viewfire node module and connect it to Vue
7   import {rtdbPlugin} from 'vuefire'
8   Vue.use(rtdbPlugin);
9
10  Vue.config.productionTip = false
11
12  new Vue({
13    vuetify,
14    render: h => h(App)
15  }).$mount('#app')
```

4. Preparing Firebase Realtime DB : Create a firebase project

Modify main.js

- Vuefire Connection: With Vuefire, we provide plugins that help you connect Vue applications with two DB formats: Firebase's RTDB (realtime database) and Firestore (Cloud Store). Since this example will use RTDB, import the rtdbPlugin module and connect it to Vue.

```
//Import the viewfire node module and connect it to Vue
import {rtdbPlugin} from 'vuefire'
Vue.use(rtdbPlugin);
```

5. Create app launch screen

Modify index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <!-- Change status bar theme color to white -->
    <meta name="theme-color" content="#ffffff">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title>To-Do List</title>
    <!--Add Material Design Icon-->
    <link href="https://fonts.googleapis.com/css?family=Roboto:100,300,400,500,700,900|Material+Icons" rel="stylesheet">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@mdi/font@latest/css/materialdesignicons.min.css">
  </head>
  <body>
    <noscript>
      <strong>We're sorry but ex09 doesn't work properly without JavaScript enabled. Please enable it to continue.</strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

5. Create app launch screen

Modify index.html

- Specify language and status bar theme: Set the HTML language to English. And change the theme color of the status bar to white.

```
<!-- Change status bar theme color to white -->
<meta name="theme-color" content="#ffffff">
```

- Change browser caption and material design icon: Change the title displayed in the caption area in the browser to the desired content. And edit the link to use the Material Design icon.

```
<!--Add Material Design Icon-->
<link href="https://fonts.googleapis.com/css?family=Roboto:100,300,400,500,700,900|Material+Icons" rel="stylesheet">
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@mdi/font@latest/css/materialdesignicons.min.css">
```

5. Create app launch screen

Modify App.vue(1)

```
<template>
  <v-app>
    <!-- Change the entire area to the card UI to maintain color consistency -->
    <v-card>
      <v-app-bar dark color="lime">
        <!-- put menu icon on the left -->
        <v-app-bar-nav-icon></v-app-bar-nav-icon>
        <v-toolbar-title>To-Do List</v-toolbar-title>
      </v-app-bar>
      <v-content>
        <v-container>
          <v-row my-5>
            <v-col cols="8" offset="1">
              <!-- Set autofocus to have input focus as soon as it runs -->
              <v-text-field label="To Do" autofocus v-model="sTodoTitle">
              </v-text-field>
            </v-col>
            <v-col cols="2" my-2>
              <v-btn fab max-height="50px" max-width="50px" color="pink" dark @click="fnSubmitTodo()">
                <v-icon>add</v-icon>
              </v-btn>
            </v-col>
          </v-row>
```

5. Create app launch screen

Modify App.vue(2)

```
24          <v-row>
25            <v-col cols="12">
26              <v-list two-line v-for="item in oTodos" :key="item.key">
27                <!-- Displayed only in read mode through item.b_edit value-->
28                <v-card flat color="grey lighten-3" v-if="!item.b_edit">
29                  <!-- Takes items one by one and displays them in tile units -->
30                  <v-list-item class="py-2">
31                    <v-list-item-action>
32                      <!-- Show the checkbox and save the change status DB when selected -->
33                      <v-checkbox v-model="item.b_completed" @change="fnCheckboxChange(item)"></v-checkbox>
34                    </v-list-item-action>
35                    <!-- Show title and strikethrough when checked -->
36                    <v-list-item-content>
37                      <v-list-item-title :class="{'style_completed':item.b_completed}"> {{ item.todo_title }}
38                      </v-list-item-title>
39                      <!-- Place the icon on the second line -->
40                      <v-list-item-subtitle class="mt-2">
41                        <!-- When the edit icon is displayed and clicked, it changes to edit mode. -->
42                        <v-icon class="pointer" @click="fnSetEditTodo(item['.key'])">create</v-icon>
43                        <!-- If the delete icon is displayed and clicked, the corresponding item is deleted.-->
44                        <v-icon class="pointer" @click="fnRemoveTodo(item['.key'])">delete</v-icon>
45                      </v-list-item-subtitle>
46                    </v-list-item-content>
47                  </v-list-item>
48                </v-card>
```

5. Create app launch screen

Modify App.vue (3)

```
49                    <!-- Display dark in edit mode through item.b_edit value -->
50                    <v-card v-else dark>
51                      <v-list-item class="py-2">
52                        <v-list-item-action>
53                          <v-checkbox v-model="item.b_completed"></v-checkbox>
54                        </v-list-item-action>
55                        <!--For text input and button use in v-list-item
56                            Using the v-card element -->
57                        <v-card-text>
58                          <!-- Moves focus directly to the input box and adds a delete icon-->
59                          <v-text-field autofocus clearable v-model="item.todo_title"></v-text-field>
60                        </v-card-text>
61                        <v-card-actions>
62                          <!-- Click the 'Save' icon in edit mode to save the item -->
63                          <v-icon class="pointer" @click="fnSaveEdit(item)">save</v-icon>
64                          <!-- Click the 'Cancel' icon in edit mode to cancel and return to reading mode-->
65                          <v-icon class="pointer" @click="fnCancelEdit(item['.key'])">cancel</v-icon>
66                        </v-card-actions>
67                      </v-list-item>
68                    </v-card>
69                  </v-list>
70                </v-col>
71              </v-row>
72            </v-container>
73          </v-content>
74        </v-card>
75      </v-app>
76    </template>
```

5. Create app launch screen

Modify App.vue (4)

```
78  // Get Firebase DB
79  import {
80    oTodosinDB
81  } from '@/datasources/firebase'
82  export default {
83    name: 'App',
84    data() {
85      return {
86        oTodos: [], // To-do data list storage variable
87        sTodoTitle: '' // To-do title storage string variable
88      }
89    },
90    // Changed to oTodos variable to make firebase easier to use
91    firebase: {
92      oTodos: oTodosinDB
93    },
94    methods: {
95      // Save to-do title, completion, and edit mode status values in DB
96      fnSubmitTodo() {
97        oTodosinDB.push({
98          todo_title: this.sTodoTitle,
99          b_completed: false,
100         b_edit: false
101       })
102       this.sTodoTitle = ''
103     },
104     // Remove the delivered task from DB
105     fnRemoveTodo(pKey) {
106       oTodosinDB.child(pKey).remove()
107     },
108     // Change b_edit of the delivered task to edit mode
109     fnSetEditTodo(pKey) {
110       oTodosinDB.child(pKey).update({
111         b_edit: true
112       })
113     },
```

5. Create app launch screen

Modify App.vue

```
114         // Change b_edit of the delivered task to read mode
115         fnCancelEdit(pKey) {
116           oTodosinDB.child(pKey).update({
117             b_edit: false
118           })
119         },
120         // Save the modified value of the delivered to-do in the DB
121         fnSaveEdit(pItem) {
122           const sKey = pItem['.key']
123           oTodosinDB.child(sKey).set({
124             todo_title: pItem.todo_title,
125             b_completed: pItem.b_completed,
126             b_edit: false
127           })
128         },
129         // When the checkbox is selected, the change value of b_completed is saved in DB.
130         fnCheckboxChange(pItem) {
131           const sKey = pItem['.key']
132           oTodosinDB.child(sKey).update({
133             b_completed: pItem.b_completed
134           })
135         }
136       }
137     }
138   </script>
139   <style>
140     .pointer {
141       /* Change the mouse pointer to a hand shape */
142       cursor: pointer;
143     }
144
145     .style_completed {
146       /*Change the title of a to-do to strikethrough */
147       text-decoration: line-through;
148     }
149   </style>
```

5. Create app launch screen

Modify App.vue

- Put the hamburger menu icon on the left side of the app bar: If you use the v-app-var-icon element, a hamburger menu icon is created on the left side of the app bar.

```
<!-- put menu icon on the left -->
<v-app-bar-nav-icon></v-app-bar-nav-icon>
```

- Set autofocus to allow the user to enter a task right away: If you want the input cursor to appear directly in the text box when the To-Do app is launched, you can specify the autofocus attribute on the v-text-field element. Note that this value is bidirectionally bound to the data property with the sTodoTitle variable.

```
<!-- Set autofocus to have input focus as soon as it runs -->
<v-text-field label="To Do" autofocus v-model="sTodoTitle">
```

5. Create app launch screen

Modify App.vue

- <+> button Executes fnSubmitTodo() function when clicked: <+> is an Add Todo button. If you click on it, the to-do should be registered, right? First, let's register the button and the fnSubmitTodo() function to be applied to the button.

```
      <v-btn fab max-height="50px" max-width="50px" color="pink" dark @click="fnSubmitTodo()">
        <v-icon>add</v-icon>
      </v-btn>
    </v-col>
  </v-row>
  <v-row>
    <v-col cols="12">
      <v-list two-line v-for="item in oTodos" :key="item.key">
        <!-- Displayed only in read mode through item.b_edit value-->
        <v-card flat color="grey lighten-3" v-if="!item.b_edit">
```

5. Create app launch screen

Modify App.vue

- Displaying to-dos: The to-do list can be printed using the v-list element. The values of oTodos, a Firebase object that stores all to-dos, are stored in an array form, and these values are extracted one by one and put in item and output.

```
<v-list-item-action>
  <!-- Show the checkbox and save the change status DB when selected -->
  <v-checkbox v-model="item.b_completed" @change="fnCheckboxChange(item)"></v-checkbox>
</v-list-item-action>
<!-- Show title and strikethrough when checked -->
<v-list-item-content>
  <v-list-item-title :class="{'style_completed':item.b_completed}"> {{ item.todo_title }}
```

5. Create app launch screen

Modify App.vue

- At this time, I set the two-line attribute to express the v-list element in two lines. If cols="12", the last part of the to-do list is changed to an ellipsis mark when it is long. Please refer to the picture below for the details of the layout.

```
    </v-list-item-title>
    <!-- Place the icon on the second line -->
    <v-list-item-subtitle class="mt-2">
      <!-- When the edit icon is displayed and clicked, it changes to edit mode. -->
      <v-icon class="pointer" @click="fnSetEditTodo(item['.key'])">create</v-icon>
      <!-- If the delete icon is displayed and clicked, the corresponding item is deleted.-->
      <v-icon class="pointer" @click="fnRemoveTodo(item['.key'])">delete</v-icon>
    </v-list-item-subtitle>
  </v-list-item-content>
```

5. Create app launch screen

Modify App.vue

- Create checkbox: The checkbox to the left of the to-do list indicates whether the task has been completed or not. The shape of the checkbox is indicated using the v-checkbox element. The state of the checkbox is managed by executing the fnCheckboxChange() function and changing the value of item.b_compete to true or false. Of course, the b_compete value must be wired with the v-model directive.

```
<!-- Display dark in edit mode through item.b_edit value -->
<v-card v-else dark>
  <v-list-item class="py-2">
    <v-list-item-action>
      <v-checkbox v-model="item.b_completed"></v-checkbox>
    </v-list-item-action>
```

5. Create app launch screen

Modify App.vue

- Create strikethrough based on checkbox selection: When the checkbox is checked, the task is finished. To inform the user of this, we used class binding to make the title appear strikethrough. Combined item.b_compete values (true, false) into class binding. If you want to display the title of the task properly, you need to put the v-list-item-title element inside v-list-itme-content.

```
<v-list-item-title :class="{'style_completed':item.b_completed}"> {{ item.todo_title }}
```

- Completing the edit function: If you click the pencil-shaped edit icon, item[', key'] is passed to the fnSetEditTodo() function to change the list to edit mode. But item[', key'] holds the value of the current item.

```
<!-- When the edit icon is displayed and clicked, it changes to edit mode. -->
<v-icon class="pointer" @click="fnSetEditTodo(item['.key'])">create</v-icon>
```

5. Create app launch screen

Modify App.vue

- Completing the edit function: If you click the pencil-shaped edit icon, item[', key'] is passed to the fnSetEditTodo() function to change the list to edit mode. But item[', key'] holds the value of the current item.

```
<!-- When the edit icon is displayed and clicked, it changes to edit mode. -->
<v-icon class="pointer" @click="fnSetEditTodo(item['.key'])">create</v-icon>
<!-- If the delete icon is displayed and clicked, the corresponding item is deleted.-->
<v-icon class="pointer" @click="fnRemoveTodo(item['.key'])">delete</v-icon>
```

- Completing the delete function: If you write the value between v-icon elements as delete, you can set a trash can icon. The class value is set to pointer so that when the mouse cursor is placed over the trash can icon, it becomes a finger shape. And when the trash can icon is clicked, the fnRemoveTodo() function works. In the figure below, the left is the list before clicking (deleting items) the trash can icon in the first task, and the right is the list after clicking.

5. Create app launch screen

Modify App.vue

- Display when in edit mode via item.b_edit value: When the user clicks the pencil icon on an item, the item should be changed to be editable. We will call this 'edit mode'. To distinguish this, based on the item's b_edit value, we divide the group into two v-card elements called v-if and v-else as follows.

```
<v-card v-else dark>
  <v-list-item class="py-2">
    <v-list-item-action>
      <v-checkbox v-model="item.b_completed"></v-checkbox>
    </v-list-item-action>
    <!--For text input and button use in v-list-item
        Using the v-card element -->
    <v-card-text>
      <!-- Moves focus directly to the input box and adds a delete icon-->
      <v-text-field autofocus clearable v-model="item.todo_title"></v-text-field>
    </v-card-text>
    <v-card-actions>
      <!-- Click the 'Save' icon in edit mode to save the item -->
      <v-icon class="pointer" @click="fnSaveEdit(item)">save</v-icon>
      <!-- Click the 'Cancel' icon in edit mode to cancel and return to reading mode-->
      <v-icon class="pointer" @click="fnCancelEdit(item['.key'])">cancel</v-icon>
    </v-card-actions>
  </v-list-item>
</v-card>
```

5. Create app launch screen

Modify App.vue

- When the save icon is clicked in the edit mode, the corresponding item is saved: In the edit mode, the save icon to reflect the modified contents in the DB should appear. So, use the v-icon element to specify the save value as follows. As before, we changed the mouse cursor shape to a hand shape with the pointer class selector. And when clicked, fnSaveEdit(item) is executed and the modified value of the delivered task is saved in the DB.

```
<!-- Click the 'Save' icon in edit mode to save the item -->
<v-icon class="pointer" @click="fnSaveEdit(item)">save</v-icon>
```

- Click the cancel icon in edit mode to return to reading mode: In edit mode, the cancel icon should also be placed next to the save icon, so assign a cancel value to the v-icon element. And when the user clicks the icon, fnCancelEdit(item[', key']) is executed to change b_edit of the delivered task to DB in read mode.

```
<!-- Click the 'Cancel' icon in edit mode to cancel and return to reading mode-->
<v-icon class="pointer" @click="fnCancelEdit(item['.key'])">cancel</v-icon>
```

5. Create app launch screen

Modify App.vue

- Import Firebase DB: To use Firebase DB, you need to import the firebase.js module located in the src/datasources folder. In this case, the symbol @ means the src folder. Please remember to use it frequently in the future.

```
// Get Firebase DB
import {
  oTodosinDB
} from '@/datasources/firebase'
export default {
```

5. Create app launch screen

Modify App.vue

- Using Vuefire Objects: If you want to easily use CRUD in your Firebase DB, we recommend getting help from Vuefire. So, declare a Vuefire object called oTodos in the firebase property and connect it with oTodosinDB. The way to use Vuefire is to declare the DB object name to be used in the firebase property only once. And this is used to read the contents in order from within the v-list element of the Firebase DB as shown below. The source below reads the items in the oTodos Vuefire object one by one using the v-for directive and stores them in the item variable. Then, it binds the todo_title content in it and outputs it. When reading a value using Vuefire, I use this method.

```
data() {
  return {
    oTodos: [], // To-do data list storage variable
    sTodoTitle: '' // To-do title storage string variable
  }
},
```

5. Create app launch screen

Modify App.vue

- Declare a variable to store the title of the To-Do app: In the data property, set the object variable to store the to-do list as the initial value. That is, the data to be used throughout the program must be declared as a variable in the data property. In addition, the initial value is defined as ' ' so that the title of the task to be input by the user in the UI is stored in a variable called sTodoTitle and can be bound.

```
// Changed to oTodos variable to make firebase easier to use
firebase: {
  oTodos: oTodosinDB
},
```

5. Create app launch screen

Modify App.vue

- Completing the Save To-Do feature: Enter a to-do in the To-Do list and press the <+> button to add a new entry to the Firebase database. This function is the creation of CRUD functions. To add new data to Firebase, you can use the push() function. At this time, the new data was passed in JSON format.

```
// Save to-do title, completion, and edit mode status values in DB
fnSubmitTodo() {
  oTodosinDB.push({
    todo_title: this.sTodoTitle,
    b_completed: false,
    b_edit: false
  })
  this.sTodoTitle = ''
```

5. Create app launch screen

Modify App.vue

- Completing the To-Do Delete function: The function to delete an item from the To-Do list is implemented using the remove() function. DB values stored in Firebase are organized in node units. The fnRemoveTodo() function receives the key value of the node to be deleted as a parameter pKey, finds the node and deletes it. oTodosinDB is the root node, and if you use child(pKey), you can directly select the desired child node.

```
// Remove the delivered task from DB
fnRemoveTodo(pKey) {
  oTodosinDB.child(pKey).remove()
},
```

5. Create app launch screen

Modify App.vue

- Completing the to-do edit function: To edit a desired item in the Firebase DB, use the update() function. As in the to-do delete function, fnSetEditTodo receives the key value (pKey) of the node as a parameter and uses the update() function to change b_edit, a value that indicates the edit status, to true.

```
// Change b_edit of the delivered task to edit mode
fnSetEditTodo(pKey) {
  oTodosinDB.child(pKey).update({
    b_edit: true
  })
},
```

- Switching from edit mode to reading mode: If you click the Cancel icon in edit mode, you need to switch back to reading mode. I changed the b_edit value of the Firebase DB to false using the fnCancelEdit() function to switch from edit mode to read mode.

```
// Change b_edit of the delivered task to read mode
fnCancelEdit(pKey) {
  oTodosinDB.child(pKey).update({
    b_edit: false
  })
},
```

5. Create app launch screen

Modify App.vue

- Saving the modified contents in the edit mode to Firebase DB: To save the modified contents in the edit mode, use the set() function, not the update() function. The reason is that the set() function, unlike the update() function, can be understood as a larger modification function that affects all child nodes. To modify data stored in Firebase, you must first find the node to modify. The title (todo_title), completion status (b_completed), and edit mode status (b_edit) were modified by finding the node to be modified with the child() function and passing the contents to be modified in JSON format to the set() function..

```
// Save the modified value of the delivered to-do in the DB
fnSaveEdit(pItem) {
  const sKey = pItem['.key']
  oTodosinDB.child(sKey).set({
    todo_title: pItem.todo_title,
    b_completed: pItem.b_completed,
    b_edit: false
  })
},
```

5. Create app launch screen

Modify App.vue

- Saving the checkbox's selection status to the Firebase database: When the checkbox is clicked, the state value should be changed. The status value of the checkbox was passed to the function called pItem.b_competed and changed to the update() function.

```
// When the checkbox is selected, the change value of b_completed is saved in DB.
fnCheckboxChange(pItem) {
  const sKey = pItem['.key']
  oTodosinDB.child(sKey).update({
    b_completed: pItem.b_completed
  })
}
```

5. Create app launch screen

Modify App.vue

- Changing the title of a task to be completed to strikethrough: I set the cursor property of CSS to pointer to change the mouse pointer to a hand shape when the mouse cursor is over the edit, delete, save, or cancel icons.

```
<style>
  .pointer {
    /* Change the mouse pointer to a hand shape */
    cursor: pointer;
  }
```

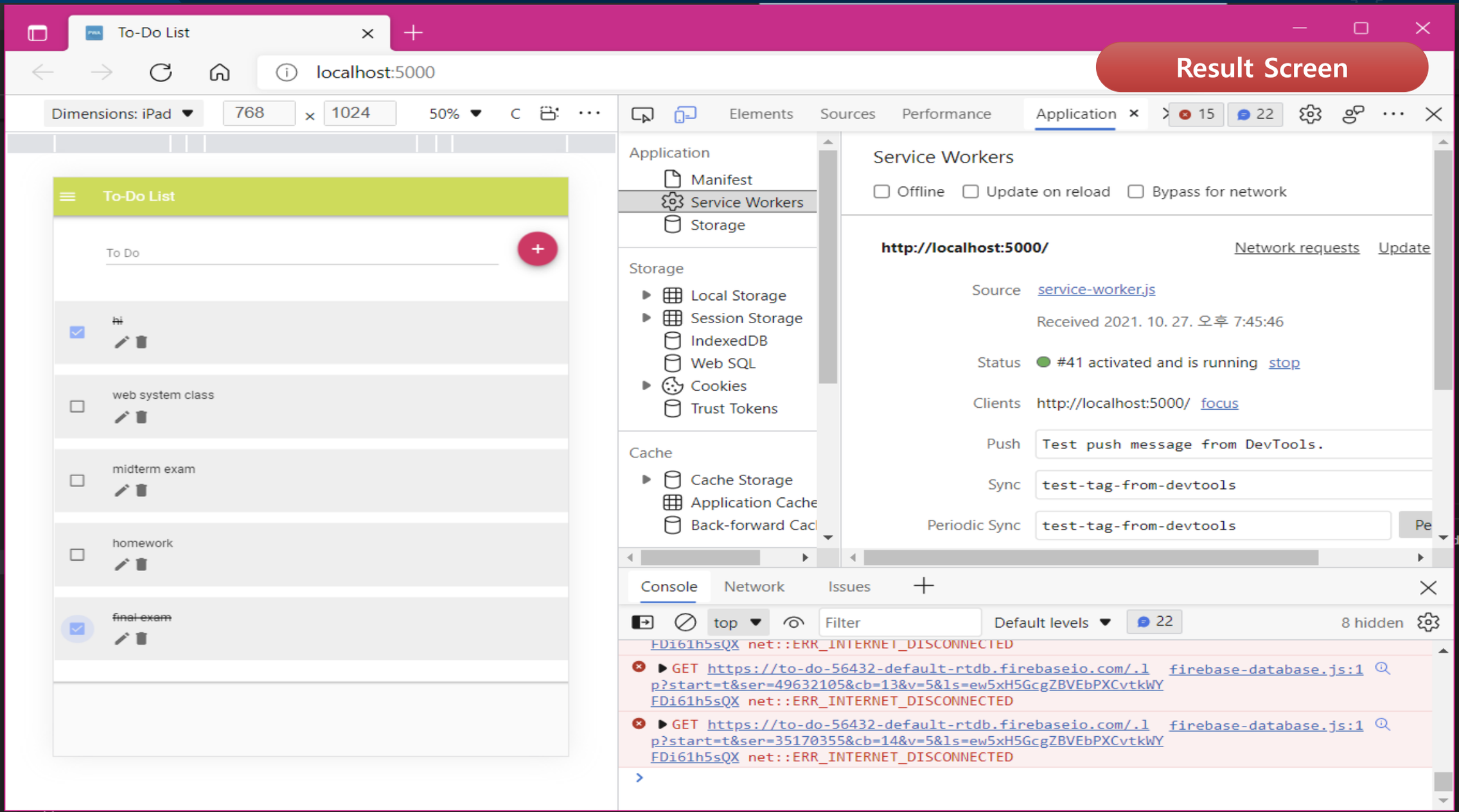
5. Create app launch screen

Modify App.vue

- CSS style to change the title of a to-do to strikethrough: If the checkbox is checked, the title should appear as strikethrough. To do this, simply set the text-decoration property of CSS to a line-through value. Use it after declaring it as the style_completed class selector name as follows. The class selector declared in this way is utilized to display the title of the To-Do item we have already looked at as follows. That is, whether the style_completed class selector is activated or not depends on whether the value of item_b_completed is true or false.

```
.style_completed {
  /*Change the title of a to-do to strikethrough */
  text-decoration: line-through;
}
</style>
```

5. Create app launch screen



Close

감사합니다

질 의 응 답