TABLE OF CONTENTS

Synchronous Framework Documentation for Functionality Writers

1 Introduction:

Synchronous Framework is built upon an abstraction model where the User interface is automatically generated with respect to the MySql database schema. It provides user authentication system with different Access control lists (ACL).

2 Abstraction:

The framework is not dependent on any foreign data. When working on the framework, one should always maintain the abstraction layer. Additionally, one should not do any work based on any particular data for e.g. specific modification of any particular menu item which will break the abstraction layer.

3 Getting Started:

4 Prerequisites:

1. **Git**
2. **Node:** Install version X or greater. Node version 9.0 and up would be ideal.
3. **Npm** or **yarn** (See Yarn website for installation instructions)

4. A **clone** of the repository
5. **Prettier**: See Prettier website for installation instructions.

5 Installation:

1. `cd synopi-live-server` to go into the project root.
2. `yarn` to install the project's npm dependencies (or `npm install`, if not using Yarn).

6 Running locally:

1. `yarn start` to start the development server (or `npm start`, if not using Yarn).

2. Open http://localhost:3000/ to open the project in your desired browser.

7 Directory Structure:

The following is a high level overview of relevant files and folders.

```
.
├── data-flow-diagram.png
├── file-structure.html
├── Makefile
├── package.json
├── package-lock.json
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── login-page-logo.png
│   └── navbar-band-logo.png
├── README.md
├── src
│   ├── chat
│   │   ├── ChatBox.js
│   │   ├── Chat.js
│   │   ├── Chatroom.js
│   │   └── Message.js
│   ├── components
│   │   ├── actions
│   │   │   ├── ClickableColumnAction.js
│   │   │   ├── ClickableDataAction.js
│   │   │   ├── delete
│   │   │   │   ├── DeleteAction.js
│   │   │   │   ├── DeleteIcon.js
│   │   │   │   └── DeleteModal.js
```

```
│   │   │   ├── edit
│   │   │   │   ├── EditAction.js
│   │   │   │   ├── EditIcon.js
│   │   │   │   └── EditModal.js
│   │   │   └── info
│   │   │       ├── InfoAction.js
│   │   │       └── infoBody.js
│   │   ├── App.js
│   │   ├── Buttons.js
│   │   ├── CreateButton.js
│   │   ├── EditButton.js
│   │   ├── fieldType
│   │   │   ├── Binary.js
│   │   │   ├── CheckBox.js
│   │   │   ├── CheckList.js
│   │   │   ├── Date.js
│   │   │   ├── DateTime.js
│   │   │   ├── index.js
│   │   │   ├── List.js
│   │   │   ├── Number.js
│   │   │   ├── Password.js
│   │   │   ├── Radio.js
│   │   │   ├── ReadOnly.js
│   │   │   ├── TextBox.js
│   │   │   └── Time.js
│   │   ├── Form.js
│   │   ├── HelpModal.js
│   │   ├── liveFeed
│   │   │   ├── ConfermationModal.js
│   │   │   ├── index.js
│   │   │   ├── LiveFeedActions.js
│   │   │   ├── LiveFeedData.js
│   │   │   └── LiveFeedRow.js
│   │   ├── localActions
│   │   │   ├── DragNDropVideoPlayer.js
│   │   │   ├── map
│   │   │   │   ├── BasicMarker.js
│   │   │   │   ├── MapViewer.js
│   │   │   │   ├── MapVisibilityButtons.js
│   │   │   │   ├── MarkerInfo.js
│   │   │   │   ├── Markers.js
│   │   │   │   └── MarkersWithLabel.js
│   │   │   ├── SelectorModal.js
│   │   │   └── VideoPlayer.js
│   │   ├── Login.js
│   │   ├── menu
│   │   │   ├── MenuBar.js
│   │   │   ├── MenuItems.js
│   │   │   ├── Menu.js
│   │   │   ├── MenuToggler.js
│   │   │   ├── NavbarBrand.js
│   │   │   └── Submenu.js
│   │   ├── MessageModal.js
│   │   ├── modal
│   │   │   ├── index.js
│   │   │   ├── ModalContent.js
```

```
|   |   |   |   └── ModalHeader.js
|   |   |   ├── Model.js
|   |   |   ├── pagination
|   |   |   |   ├── en_US.js
|   |   |   |   ├── KeyCode.js
|   |   |   |   ├── Options.js
|   |   |   |   ├── Pager.js
|   |   |   |   └── Pagination.js
|   |   |   ├── Search.js
|   |   |   ├── Select.js
|   |   |   ├── subtab
|   |   |   |   ├── index.js
|   |   |   |   ├── SubModel.js
|   |   |   |   ├── SubTabDetails.js
|   |   |   |   ├── SubTable.js
|   |   |   |   ├── TabBar.js
|   |   |   |   └── TabItems.js
|   |   |   ├── table
|   |   |   |   ├── TableBody.js
|   |   |   |   ├── TableHeaders.js
|   |   |   |   └── Table.js
|   |   |   ├── tabView
|   |   |   |   ├── TabBar.js
|   |   |   |   ├── TabBody.js
|   |   |   |   └── TabView.js
|   |   |   ├── userAccount
|   |   |   |   ├── Accounts.js
|   |   |   |   ├── Credentials.js
|   |   |   |   ├── Dropdown.js
|   |   |   |   ├── index.js
|   |   |   |   └── Profile.js
|   |   |   ├── UserGuide.js
|   |   |   └── windowGrid
|   |   |       ├── WindowBar.js
|   |   |       └── WindowGrid.js
|   |   ├── data_client.js
|   |   ├── file_manager
|   |   |   ├── data.json
|   |   |   └── index.js
|   |   ├── images
|   |   |   ├── drop-down-arrow-2.png
|   |   |   ├── drop-down-arrow.png
|   |   |   ├── resources
|   |   |   |   ├── lowVolume.svg
|   |   |   |   ├── maximize.svg
|   |   |   |   ├── minimize-fs.svg
|   |   |   |   ├── minimize.svg
|   |   |   |   ├── mute.svg
|   |   |   |   ├── pause.svg
|   |   |   |   ├── pip-fs.svg
|   |   |   |   ├── pip.svg
|   |   |   |   ├── play.svg
|   |   |   |   ├── setting.svg
|   |   |   |   └── unmute.svg
|   |   |   ├── sms-26.svg
|   |   |   ├── sms.svg
```

```
│   │   └── walk.png
│   ├── index.js
│   ├── styles
│   │   ├── chat.css
│   │   ├── datepicker.css
│   │   ├── list.css
│   │   ├── main.css
│   │   ├── numberInput.css
│   │   ├── pagination.css
│   │   ├── timepicker.css
│   │   └── videoplayer.css
│   └── util.js
├── yarn-error.log
└── yarn.lock
```

8 index

index initiates the init method from data client. A callback method is passed through init method.

```
DataClient.init(() => {
  window.app_query = app_query;
  window.unregister = unregister;
  window.uuid = uuid;
  ReactDOM.render(<App />, document.getElementById('root'));
});
```

9 Data Client

In data client we export 3 methods

1. init
2. unregister
3. app_query

9.1 Init

It first connects with the websocket through a socket address. There are 4 websocket method that we used.

- #### 9.1.1 onopen

  It initiates the callback method which invokes app.js

- #### 9.1.2 onmessage

  The response from the server after sending an app query- is handled by onmessage

- #### 9.1.3 onclose

  It checks if websocket is closed and responds accordingly.

- #### 9.1.4 onerror

  In case of an error of the websocket, onerror is invoked.

9.2 Unregister

It deletes the cid

9.3 Window App Query

Using app query, the user receives a response. This response is used to generate the user interface. It is also used in creating and authenticating different data, for e.g. login.

Basic app query structure:

window.app_query(object, action, argument, client_id, callback_method)

The object, action, client_id, callback_method arguments stays in same format generally. But for the argument part, it is defined from back-end. It varies from queries to queries.

When an app query is initiated, it sends a request to the server by websocket.send(req) method. req is an object and it includes cid, object, action, args.

For new app queries, the user must know in what format the argument should be sent, before sending the query.

Below are some representations of how to use app query in common scenarios.

Login Action

```
 window.app_query(
      "user",
      "login",
      { name: username, secret: secret },
      cid,
      this.onMessage
    );
```

Table Render

```
 window.app_query(
      this.props.tableName,
      "read",
      { limit: [startRow, pageSize] },
      cid,
      this.onMessage
    );
```

Table Row Creation

```
window.app_query(tableName, "create", { set: payload },
cid, this.onMessage);
```

Table Row Update

```
 window.app_query(
      tableName,
      "update",
      { set: payload, where: { [pk]: row[pk] } },
      cid,
      this.onMessage
    );
```

Table Row Delete

```
 window.app_query(
      tableName,
      "delete",
      { where: { [pk]: rowID } },
      cid,
      this.deleteResponse
    );
```

App Query for Init

```
window.app_query(tableName, "init", {}, cid,
this.onMessage);
```

App Query for Remote Actions

```
window.app_query(tableName, action, row, cid,
this.onMessage);
```

App Query for Location Actions

```
  window.app_query(
      this.props.tableName,
      "push",
      selectorRow,
      cid,
      this.onMessage
```

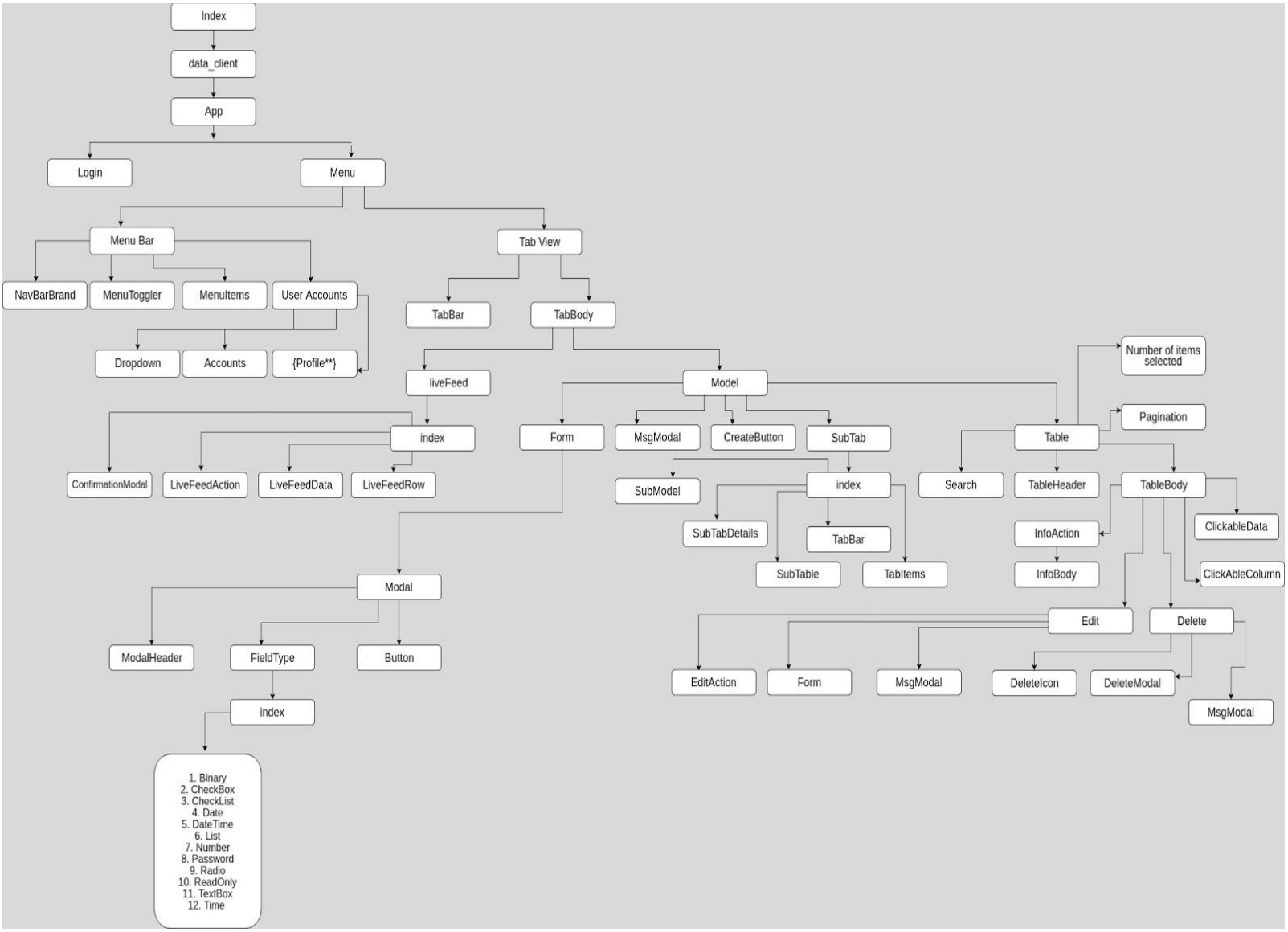
```
    );
```

App Query for selector

```
window.app_query(table_name, push, row_data, cid,
callback_method)
```

10 UI Components

The following diagram represents the complete UI component structure.



Detail diagram of Synchronous-framework

For better quality image of the diagram follow this link:

https://drive.google.com/file/d/1DtAoBr8urZ8ibh_DneulglADv90i2gZ1/view?usp=sharing

11 Conventions

This section is intended for functionality writer's conventions.

11.1 Naming Conventions

1. **Variables**: camel case
2. **Functions**: camel case
3. **Class name**: pascal case
4. **Folder name**: camel case
5. **File name**: pascal Case

Exception: **index.js** using small “i” as first letter.

6. File name and class name should be **exactly same**.

11.2 Spacing and formatting:

1. There should be a 1 line gap between any of imports, classes, functions, render.
2. Formatting is handled by **prettier**.

11.3 Component Structure

There will be two types of component used throughout the project.

1. **Container component (class component)**
2. **Presentational component (functional component)**

{Naming conventions of two components is to be decided}

Mainly, the functionality writers will be working on Container component. Along with that, they might need to add some logic to presentational component. They will also work on breaking down large presentational component.

11.4 Communication with UI team through a Prototype

We strictly separate the functionality writing and UI designing part. The least needed communication will be managed through a prototype.

We use the **high-fidelity** prototyping model.

11.4.1 Objective of using prototype

1. The UI team and functionality writers can work independently.
2. Needed testings can be done separately.

11.4.2 Workflow

Initially, a prototype will be designed by the UI team according to the given requirements. This prototype will not have any functionality at all. Functionality writers will work on a completely different file which will be generated by following the prototype. Firstly, they will analyze that prototype and will make decisions about breaking the component into sub-components if needed. Then they will add the specified functionality.

11.4.3 Convention of providing requirements for the prototype

1. The requirement should ask for a JSX structured file from the UI team which provides a basic skeleton without any functionality.

2. The prototype file format should be in .js . A css file will also be included (if needed) in the prototype.

3. The placement/position of the new required component/element in the layout should be specifically mentioned.

4. Exact details of the type of the element that needs to be added to the prototype. For e.g. the HTML element like button, input form etc. should be mentioned in the requirements directly.

5. The requirement should ask to return only 1 component

6. It should include the requirements of completion of all testings for e.g. layout, responsiveness.

11.4.4 Prototype sharing methods(probable):

1. separate repository
2. separate branch

11.4.5 An example of using prototype

We want to build a volume slider which functions and looks exactly like the youtube volume slider.

1. Firstly, the functionality writer team will figure out the detailed requirement of the volume slider. The requirement will be passed to the UI team.

2. For the sake of simplicity of this example, we take the requirement of the volume slider to be like youtube volume slider. Other standard requirements will be applied.

3. The UI team will then develop a prototype based on those requirements. This prototype will be passed to the functionality writing team.

This image represents the UI of the worked prototype. To be clear, this slider has no functionality at all. The slider thumb can not be dragged and it definitely can not change the volume.

On coding part, the prototype roughly looks like this (of course a css file will be provided):

```
import React, { Component } from "react";
const InputRange = () => {
   return (
      <div className="input-range-wrapper">
         <div className="input-range" />
      </div>
   );
};
export default InputRange;
```

4. The functionality writing team will take that prototype and see if the prototype can be broken into multiple components. In this case, the prototype can not be broken into multiple components. So proceeding to next step.

5. Then they will add needed functionality to complete the task.

   In this example, they will add the following functionalities-

   - Thumb sliding changes volume
   - Thumb sliding causes visual change.

Now, the volume slider is fully functional and visually meets the requirements.

In the coding part, after the completion of the work will look roughly like this:

```
import React, { Component } from 'react';
import {videoPlayerSlider} from "../VideoPlayerUtil";
class InputRange extends React.Component{

       componentDidMount(){
videoPlayerSlider.initDragElement(document.getElementsByClassName("inpu
t-range")[0],document.getElementsByClassName("input-range-wrapper")[0])
;
       }
    render(){
        return (
            <div className="input-range-wrapper">
                <div className="input-range"></div>
            </div>
        );
    }
}
export default InputRange;
```

12 New features/functionality addition

Features/functionality are divided in 2 parts.

1. Core features
2. Sub features

Any new core features should be added into this directory:

`src> components > “respective_feature_name” > files`

For sub features, a new folder representing feature name will be created into this directory:

`src> “respective_feature_name” > files`

All of the related files should go inside these directories.

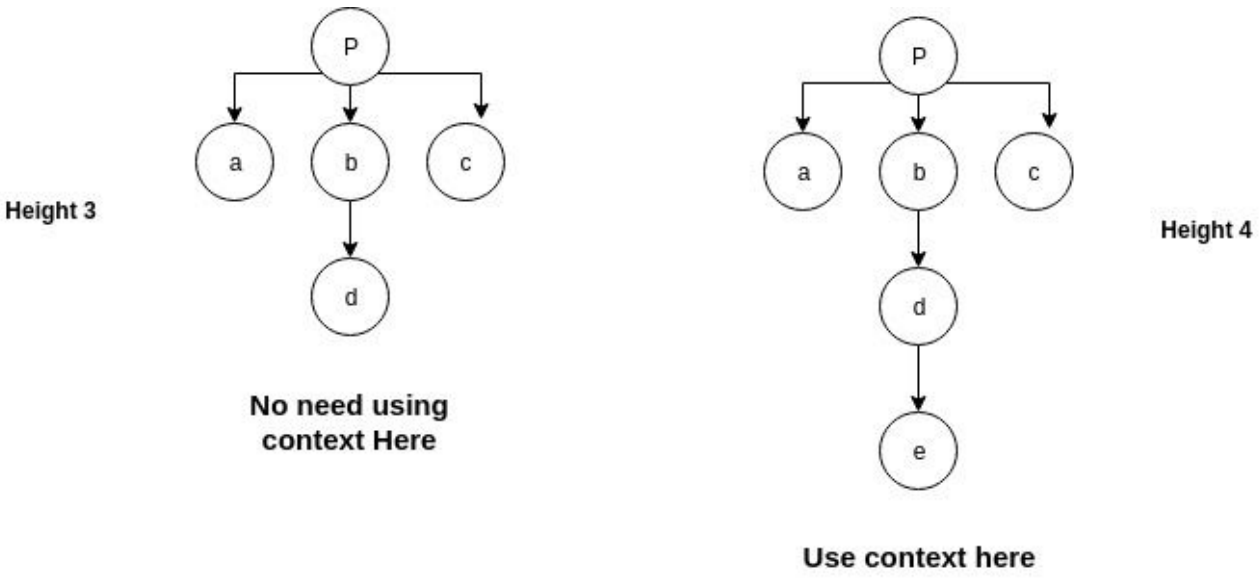
13 Using Context API

13.1 Objective

Reduce props passing.

13.2 When to use

If a parent component has less than a height/depth of 3, context api should not be used. In case of the height or depth is greater than 3, props passing should be done using context API.



14 Function writing sequence

Functions should be written in the following orders:

1. Constructor
2. Lifecycle methods with their orderly sequence
3. Event handlers like keypress event or click handlers methods
4. render
5. proptypes, default props

15 Util Functions

All the utility functions should be placed inside a single file. In case of using util functions, one needs to import it and use it afterwards. The directory of the file is:

`src > util.js`

If a function is being used multiple times or by multiple components and the function is not related to changing any state, then it should be included inside the util file.

16 UI Layout

Following tools / packages have been used to build the UI layout of this framework.

1. MUI CSS
2. Re-resizable
3. React-draggable
4. React-tabs
5. Flexbox
6. Bootstrap

17 UI Frameworks

17.1 Responsive

1. Bootstrap
2. Material design

17.2 Mobile

1. React native

18 Device Input

18.1 Keyboard Events

1. Arrows
2. Space Bar
3. Enter
4. Escape
5. Tab

18.1.1 Arrows

Up arrow / down arrow can be used to open dropdown / date / time pickers

18.1.2 Space Bar

- Space bar can be used to submit or cancel a form.
- Open dropdown / date / time pickers
- Toggle between check and uncheck checkbox / radio inputs.

18.1.3 Enter

- Submit a form
- select input from a dropdown / date / time pickers

18.1.4 Escape

- Closes a modal instantly

- If an input like dropdown / date / time pickers are being selected, escape can be used to close it down. In that case the modal will not be closed instantly.

18.1.5 Tab

Initially the first input type is focused automatically. Tab can be used to focus next element.

18.2 Mouse Events

1. Left click, Right click, Double Click
2. Hover over

18.3 Touch Events

This events are meant for mobile or touch enabled device implementation.

1. Touch event
2. Swipe

19 F.A.Q

1. How to get a build directory of the project?

   Answer: `npm run build / yarn build` generates an optimized build of the directory inside the build folder

2. Showing Module not found error.
   Answer: `npm install / yarn` will install all the necessary packages to your local directory to fix the issue.

3. Disconnected, reload pop up.

   Answer: Sometimes, the web socket gets disconnected for testing purpose. Changing the web socket address might resolve this issue. If that does not, then waiting for some time and trying to reconnect again will fix this issue.