Build Angular UI Components

By

Donda Jenis Nareshbhai (ID No. 20CEUOS070)

A project submitted

In

partial fulfillment of the requirements for

degree of

BACHELOR OF TECHNOLOGY

In

Computer Engineering

Internal Guide

Prof. Prashant M. Jadav

Associate Professor

Dept. of Comp. Engg.

External Guide

Mr. Parth Shah

Technical Lead

Crest Data Systems

Faculty of Technology

Department of Computer Engineering

Dharmsinh Desai University

April 2024

CERTIFICATE

This is to certify that the project work titled

Build Angular UI Components
is the bonafide work of
Donda Jenis Nareshbhai (ID : 20CEUOS070)

carried out in the partial fulfillment of the degree of Bachelor of Technology in Computer Engineering at Dharmsinh Desai University in the academic session

December 2023 to April 2024.

Prof. P. M. Jadav Associate Professor Dept. of Computer Engg.	Dr. C. K. Bhensdadia Prof. and Head of the Department Dept. of Computer Engg.

Faculty of Technology

Department of Computer Engineering

Dharmsinh Desai University

April 2024

Table of contents

Chapter		Page
I.	Introduction	**1**
	1. Build Angular UI Components	1
	2. Existing Components	2
II.	About the system	**3**
	1. Purpose	3
	2. Scope	3
	3. Tools and Technologies	4
III.	About different methodologies of testing	**7**
	1. Types of Testing	7
	2. Harness	9
IV.	Harness Class Design	**10**
	1. Components	10
	2. Harness Classes	11
V.	Test Case Design	**19**
	1. Testcases	19
VI.	Analysis	**29**
	1. Screen shots	29
VII.	Conclusion	**32**
	1. Conclusion	32
	2. Future Extension	32
VIII.	Bibliography	**33**

List of Table

Table 4.2.1 Button Harness class 11
Table 4.2.2 Navbar Harness class 11
Table 4.2.3 Radio Button Harness class 12
Table 4.2.4 Menu Harness class 12
Table 4.2.5 Button-Toggle Harness class 13
Table 4.2.6 Input Harness class 13
Table 4.2.7 Card Harness class 14
Table 4.2.8 Auto Complete Harness class 14
Table 4.2.9 Tabs Harness class 15
Table 4.2.10 Progress bar Harness class 15
Table 4.2.11 Dialog Harness class 16
Table 4.2.12 Date-Time Kit Harness class 16
Table 4.2.13 Paginator Harness class 17
Table 4.2.14 Form Harness class 18

Table 5.1.1 Button Testcases 19
Table 5.1.2 Navbar Testcases 19
Table 5.1.3 Radio Button Testcases 20
Table 5.1.4 Button-Toggle Testcases 21
Table 5.1.5 Input Testcases 21
Table 5.1.6 Menu Testcases 22
Table 5.1.7 Card Testcases 22
Table 5.1.8 Auto Complete Testcases 23
Table 5.1.9 Tabs Testcases 24
Table 5.1.10 Progress bar Testcases 25
Table 5.1.11 Date-Time Kit Testcases 25
Table 5.1.12 Dialog Testcases 26
Table 5.1.13 Paginator Testcases 27
Table 5.1.14 Form Testcases 28

Chapter 1
Introduction

1.1 Build Angular UI Components :

In the realm of modern software development, the user interface (UI) stands as a critical element, shaping the overall user experience. Recognizing this importance, our project revolves around the development of UI components within the Angular framework, with a primary focus on implementing a robust testing infrastructure centered around harnesses. Harnesses serve as vital tools, providing an abstraction layer that streamlines the testing process by enabling structured interactions with UI components programmatically.

Our objectives extend beyond mere component development; we strive to create a suite of reusable UI components such as paginator, autocomplete, and buttons, meticulously crafted to adhere to design standards and accessibility guidelines. Leveraging the Jasmine testing framework, we are dedicated to developing a comprehensive suite of tests for each component, with a strong emphasis on harness-centric testing methodologies. Through this project, we aim to deliver high-quality UI components that not only enhance user experiences but also inspire confidence through testing and reliability..

1.2 Existing Components

Material Design components, commonly referred to as Mat components, are a set of UI elements developed and maintained by the Angular Material team. These components adhere to the Material Design guidelines provided by Google, offering a unified and visually appealing design language for web applications built with Angular.

The Mat components encompass a wide range of UI elements, including buttons, input fields, checkboxes, radio buttons, sliders, tabs, menus, and many more. Each component is designed with consistency, accessibility, and usability in mind, ensuring a seamless user experience across different devices and platforms.

In the context of testing Angular applications, harnesses provide a structured interface for interacting with UI components programmatically. For Mat components, custom harnesses are developed alongside the components themselves to facilitate efficient and comprehensive testing. These harnesses abstract the complexities of component interactions, enabling developers to simulate user actions, query component properties, and verify expected behaviours during testing scenarios.

The integration of Mat components with custom harnesses enhances the testing process, promoting code maintainability, scalability, and reliability. By leveraging the reliability and visual appeal of Mat components and harnesses, developers can ensure the quality and stability of their Angular applications, delivering a seamless user experience to their audience.

Chapter 2
About the System

2.1 Purpose

The primary purpose of our project is to ensure the reliability, functionality, and quality of user interface (UI) components developed within the Angular framework. By focusing on harness-centric testing methodologies, our goal is to streamline the testing process and enhance code maintainability while delivering robust and thoroughly tested UI components. We aim to provide developers with a suite of reusable Angular UI components, meticulously crafted to adhere to design standards, accessibility guidelines, and best practices in component development.

Through the integration of custom harnesses alongside component development, we seek to empower developers with efficient tools for programmatically interacting with UI components during testing scenarios. Ultimately, our project strives to elevate the standard of UI development within the Angular ecosystem, fostering a culture of reliability, scalability, and user-centric design in web application development.

2.2 Scope

The scope of our project entails the development, testing, and implementation of a range of custom Angular UI components tailored to specific project requirements. This includes the design and creation of various components such as buttons, input fields, pagination, autocomplete, and more, each crafted with a focus on modularity, reusability, and adherence to coding standards. Additionally, the project involves the implementation of custom harnesses for each UI component to facilitate efficient testing procedures.

These harnesses will provide structured interfaces for programmatically interacting with the components during testing, ensuring thorough coverage of use cases and scenarios. Harness-centric testing methodologies will be employed to streamline testing processes, enhance code maintainability, and ensure the reliability and stability of the components across different environments and usage scenarios. The project will also encompass the construction of comprehensive test suites using the Jasmine testing framework, covering a wide range of scenarios including input validation, user interactions, edge cases, and asynchronous operations.

2.3 Tools and Technologies

Technologies:

- Angular
- TypeScript
- Jasmine Framework

Tools:

- Visual studio code
- Git

In Details:

Angular :

- Angular is a popular open-source web application framework maintained by Google and a community of developers.
- It is used for building dynamic and interactive web applications, particularly single-page applications (SPAs) where content is dynamically loaded without needing to refresh the entire page.
- Angular provides a comprehensive set of tools and features that simplify the development process, including a powerful templating system, two-way data binding, dependency injection, and modular architecture.

TypeScript :

- TypeScript is an open-source programming language developed and maintained by Microsoft.
- It is a superset of JavaScript, meaning that any valid JavaScript code is also valid TypeScript code.
- TypeScript is an open-source programming language developed and maintained by Microsoft. It is a superset of JavaScript, meaning that any valid JavaScript code is also valid TypeScript code. TypeScript extends JavaScript by adding optional static typing, interfaces, classes, and other features that make it easier to build large-scale, complex applications.

Jasmine :

- Jasmine is a popular open-source testing framework for JavaScript. It is used primarily for testing JavaScript code in web applications, including both front-end and back-end code.
- Jasmine provides a behavior-driven development (BDD) syntax that makes writing tests more descriptive and readable.

Git + CLI (Command Line Interface) :

- Git (version control) is another must-have skill a developer should have to store their project on GitHub. It helps developers to work and collaborate with each other and it allows them to track and host different versions of project files. You should have good knowledge of how Git and these code hosting platforms work. Developers use the command of Git to track the version of your files, so learn how to use all the commands such as push, pull, add, commit, etc. Also learn about merging, branching, handling merging conflicts, etc.

- Everything in React you will be doing with the help of CLI (Command-line interface). Installing packages, using NPM, creating a react app, running react application, and a lot of things so you really need to make a habit of using CLI. Below is an example of running a react application using CLI.

Visual Studio Code :

- Visual Studio Code (VSCode) is a free and open-source code editor developed by Microsoft.
- It is widely used by developers for various programming languages and platforms, including but not limited to JavaScript, TypeScript, Python, Java, and C#

Chapter 3
About different methodologies of testing

3.1 Types Of Testing

There are several methodologies of testing, each designed to address specific aspects of the software development and testing process. Here are some of the most common methodologies:

- **Unit Testing:**
  - Unit testing involves testing individual components or units of code in isolation. It focuses on verifying that each unit of code functions correctly in isolation from other parts of the system.
  - Unit tests are typically automated and written by developers to ensure that their code behaves as expected under different conditions.

- **Integration Testing**:
  - Integration testing involves testing how individual units of code work together when integrated into larger modules or systems.
  - It focuses on verifying that different components interact with each other correctly and that the system as a whole behaves as expected. Integration tests help identify issues that may arise due to interactions between different parts of the system.

- **Functional Testing:**
  - Functional testing involves testing the functionality of the software from an end-user perspective. It focuses on verifying that the software meets the specified requirements and performs the intended functions correctly.
  - Functional tests are typically black-box tests that do not require knowledge of the internal implementation of the software.

- **Regression Testing:**
  - Regression testing involves re-running previously executed tests to ensure that recent changes to the software have not introduced new bugs or caused existing functionality to break.
  - It helps ensure that the software remains stable and reliable over time, even as new features are added or existing features are modified.

- **Acceptance Testing:**
  - Acceptance testing involves testing the software to ensure that it meets the acceptance criteria defined by stakeholders.
  - It focuses on verifying that the software meets the business requirements and is ready for deployment to production. Acceptance tests are typically performed by stakeholders or end-users in a real-world environment.

- **Performance Testing:**
  - Performance testing involves testing the performance characteristics of the software, such as responsiveness, scalability, and reliability, under various conditions.
  - It helps identify performance bottlenecks and optimize the software to ensure that it can handle expected levels of load and usage.

- **Security Testing:**
  - Security testing involves testing the software for vulnerabilities and weaknesses that could be exploited by attackers.
  - It helps ensure that the software is secure and resistant to malicious attacks, such as unauthorized access, data breaches, and denial-of-service attacks.

These are just a few examples of the different methodologies of testing. Depending on the nature of the software being tested and the specific requirements of the project, different combinations of these methodologies may be used to ensure the quality and reliability of the software.

For our project we use unit testing for testing particular component.

3.2 Harness :

A component harness is a class that lets a test interact with a component via a supported API. Each harness's API interacts with a component the same way a user would. By using the harness API, a test insulates itself against updates to the internals of a component, such as changing its DOM structure. The idea for component harnesses comes from the PageObject pattern commonly used for integration testing.

@angular/cdk/testing contains infrastructure for creating and using component test harnesses. You can create test harnesses for any component, ranging from small reusable widgets to full application pages.

The component harness system supports multiple testing environments. You can use the same harness implementation in both unit and end-to-end tests. This means that users only need to learn one API, and component authors don't have to maintain separate unit and end-to-end test implementations.

- ComponentHarness is the abstract base class for all component harnesses. Every harness extends this class. All ComponentHarness subclasses have a static property, hostSelector, that matches the harness class to instances of the component in the DOM. Beyond that, the API of any given harness is specific to its corresponding component; refer to the component's documentation to learn how to use a specific harness.

Chapter 4
Harness Class Design

4.1 Components :

- ❖ Button
- ❖ Navbar
- ❖ Radio Button
- ❖ Button-Toggle
- ❖ Input
- ❖ Menu
- ❖ Card
- ❖ Tabs
- ❖ Auto-Complete
- ❖ Progress Bar
- ❖ Dialog Component
- ❖ Date-Time Kit
- ❖ Paginator
- ❖ Form

➢ For this components we design the harness classes which contains asynchronous functions. Which is used in jasmine testing of framework for developers to easily testing the component. So basically harness is a medium between component and testing which is useful to give dynamic testing and reducing code in testing.

4.2 Harness Classes :

❖ Button

Table 4.2.1 Button Harness class

Asynchronous Function	Parameter	Description
click()	None	Click on button
hover()	None	Hover on button
getText()	None	Get the text button
getColor()	None	Get the color of the button
getHoverColor()	None	Get the color of button when it is hovered
getTextColor()	None	Get the color of button text

❖ Navbar

Table 4.2.2 Navbar Harness class

Asynchronous Function	Parameter	Description
getLogoText()	None	Get the text of the navbar logo
getMenuItems()	None	Get the the navbar items
clickItem()	(index:number)	Click on particular item
getMenuTextColor()	None	Get the color of menu items
hoverOverMenuItem()	(index:number)	Hover on particular menu item
getMenuItemColor()	(index:number)	Get the color of menu item
openDropDown()	None	Open dropdown of menu item
isDropdownOpen()	None	Check if dropdown is open or not
closeDropdown()	None	Close the dropdown

getNavbarColor	None	Get the color of navbar color

❖ Radio Button

Table 4.2.3 Radio Button Harness class

Asynchronous Function	Parameter	Description
isChecked()	(index:number)	Checks whether particular label is checked or not
click()	(index:number)	Click on radio label
noOfButtons()	None	Get the no. of label that radio button have

❖ Menu

Table 4.2.4 Menu Harness class

Asynchronous Function	Parameter	Description
getButtonColor()	None	Get the color of menu button
getButtonTextColor()	None	Get the text color of menu button
click()	None	Click on menu button
isClicked()	None	Check whether button is clicked or not
isMenuOpen()	None	Checks whether menu is open or not
closeMenu()	None	Close the menu
isMenuClosed()	None	Checks whether menu is closed or not
getMenuItems()	None	Get the menu items
clickMenuItem()	(index:number)	Click on menu item
isItemClicked()	(index:number)	Checks whether item is clicked or not
isSubMenuOpen()	(index:number)	Checks whether particular submenu is open or not
closeSubMenu()	None	Close the submenu
isSubmenuClosed()	None	Checks whether submenu is close or not
getSubMenuItems()	(index:number)	Get particular submenu items
clickSubMenuItem()	(index:number)	Click on particular submenu item
isSubItemClicked()	(index:number)	Check submenu items is clicked

❖ Button-Toggle

Table 4.2.5 Button-Toggle Harness class

Asynchronous Function	Parameter	Description
click()	(index:number)	Click on button
isClicked()	(index:number)	Checks whether button is clicked or not
noOfButtons()	None	Get the no. of label that toggle button have
getButtonColor()	(index:number)	Get the color of particular button
getTextColor()	(index:number)	Get the text color of particular buttton

❖ Input

Table 4.2.6 Input Harness class

Asynchronous Function	Parameter	Description
getValue()	None	Get the value of input box
setValue()	(value:string)	Set the value of input box
getPlaceholder()	None	Get placeholder of input
isEmailValid()	None	Checks whether error message is there or not
getPattern()	None	Get the pattern of input
isInputValid()	None	Checks whether pattern error message is there or not

❖ Card

Table 4.2.7 Card Harness class

Asynchronous Function	Parameter	Description
getTitle()	None	Get the title of card
getBodyContent()	None	Get the body content of card
getFooterContent()	None	Get the footer content of the card
getCardWidth()	None	Get the width of the card
getTitleSize()	None	Get the size of card title

❖ Auto-Complete

Table 4.2.8 Auto Complete Harness class

Asynchronous Function	Parameter	Description
click()	None	Click on autocomplete input
getOptions()	None	Get the options
setInput()	(value:string)	Set the autocomplete input
selectOption()	(index:number)	Select an option
getInputValue()	None	Get the input box value
getBgColor()	None	Get the input box background color
getPlaceholder()	None	Get the placeholder
closeOptions()	None	Close options

❖ Tabs

Table 4.2.9 Tabs Harness class

Asynchronous Function	Parameter	Description
getTabs()	None	Get the tabs
selectTab()	(index:number)	Select the tab
isTabActive()	(index:number)	Checks whether tab is active or not
isTabDisable()	(index:number)	Checks whether tab is disable or not
getBgColor()	(index:number)	Get the background color of tab
hasBottomBorder()	(index:number)	Checks whether tab has bottom color or not
getTabContent()	None	Get the active tab content
getTabContentWidth()	None	Get the width of the tab

❖ Progress bar

Table 4.2.10 Progress bar Harness class

Asynchronous Function	Parameter	Description
isDeterminate()	None	Is progressbar is determinate or not
getProgressWidth()	None	Get the width of the progress bar
getProgressColor()	None	Get the color of progress bar color
getHeight()	None	Get the height of the progress bar
getWidth()	None	Get the width of the progress bar

❖ Dialog Components

Table 4.2.11 Dialog Harness class

Asynchronous Function	Parameter	Description
click()	None	Click on dialog component
isDialogOpen()	None	Checks whether dialog is open or not
closeDialog()	None	Close the dialog
getBgColor()	None	Get background color of body
getBgColor()	None	Get the background color of tab
clickOnDialog()	None	Click on dialog
isShowContent()	None	Checks whether dialog show content or not

❖ Date-Time Kit

Table 4.2.12 Date-Time Kit Harness class

Asynchronous Function	Parameter	Description
openPicker()	None	Open date time picker
isPickerOpen()	None	Check whether picker is open or not
getMonth()	None	Get selected month
getYear()	None	Get selected year
getBgColor()	(day:number)	Get background color of particular day
decreaseMonth()	None	Decrease month
increaseMonth()	None	Increase month
selectDate()	(day:number)	Select a particular date

getInput()	None	Get input box value
clearInput()	None	Clear the input box value
setToToday()	None	Set today's date
setTime()	(hours: string, minutes: string)	Set the time
getHour()	None	Get the selected hour
getMinute()	None	Get the selected minute
closePicker()	None	Close the date time picker

❖ Paginator

Table 4.2.13 Paginator Harness class

Asynchronous Function	Parameter	Description
clickOnPage()	(pagenumber: number)	Click on page number
getCurrentPage()	None	Get the current page
currentPageOpacity ()	None	Get opacity of current page
getVisiblePages()	None	Get visible pages
goToNextPage()	None	Go to next page
goToPrevPage()	None	Go to previous page
goToFirstPage()	None	Go to first page
goToLastPage()	None	Go to last page

goToLastPage()	None	Go to last page
setItemsPerPage()	(items: string)	Set items per page
getItemsPerPage()	None	Get items per page

- Form

Table 4.2.14 Form Harness class

Asynchronous Function	**Parameter**	**Description**
getFormTitle()	None	Get form title
getFormWidth()	None	Get the form width
getFormHeight()	None	Get the form height

Chapter 5
Test Case Design

5.1 Testcases

Here are the testcases which is used the harness class of particular component to test the component.

- Button

Table 5.1.1 Button Testcases

Testcase	Description	Status
Should click on button	It is used for checking whether button is clickable or not	Pass
Should Change color	It checks whether color of button is changed or not	Pass
Should change text of button	It checks whether text of button is changed or not	Pass
Should change text color	It checks whether color of button text is changed or not	Pass
Should change color on Hover	It checks color of button after hovering	Pass

- Navbar

Table 5.1.2 Navbar Testcases

Testcase	Description	Status
Should change logo	It is used for checking whether logo of navbar is changed or not	Pass
Should get correct items	It is check correct items show on navbar	Pass
Should change item color	It is check for item color	Pass

Should click on menu item	It checks whether menu item is clickable or not	Pass
Should open dropdown menu	It opens dropdown by clicking on item	Pass
Should click on dropdown-menu items	It checks whether dropdown menu item is clickable or not	Pass
Should change background color of navbar	It checks color of navbar background color	Pass
Should open nested dropdown menu	It opens the nested dropdown menu	Pass
Should close nested dropdown menu	It closes the nested dropdown	Pass

- Radio Button

Table 5.1.3 Radio Button Testcases

Testcase	Description	Status
Should click on radio button	Checks whether radio button is clickable	Pass
Should unchecked when click on other radio label	Other label should be unchecked when click on other label	Pass
Should get correct no. of labels	For getting how many labels are there in radio buttton	Pass
Should add new label in radio button	Should add new label in radio button	
At a time only one label should be checked	Only one label is selected in radio button	Pass

- Button-Toggle

Table 5.1.4 Button-Toggle Testcases

Testcase	Description	Status
Should Click on button	Checks whether button is clickable	Pass
Should get correct no. of buttons	Getting correct no. of toggle buttons	Pass
At a time only one button is active	Only one button is toggle at a time	Pass
Should change color when active	Checking whether button is active because there only button color is changed	Pass
Should change text color when it is active	Checking whether button is active because there only button text color is changed	Pass

- Input

Table 5.1.5 Input Testcases

Testcase	Description	Status
Should get and set value of input box	Getting the value of input box	Pass
Should get and set placeholder of input box	Checking whether placeholder is set or not	Pass
Should validate email on email type	It is used to checking whether input is follow emailId type in input	Pass
Should get and set pattern in input	Checking whether pattern is set or not	Pass
Should Validate pattern	It is used to checking whether input is follow pattern regex type in input	Pass
Should show appropriate	Checking error message for invalid	Pass

error message on validation	input	

- Menu

Table 5.1.6 Menu Testcases

Testcase	Description	Status
Should click on menu	Checking menu button is clickable	Pass
Should get correct menu button color	Checking menu button color	Pass
Should get and set correct menu button text	Checking correct text of menu button	Pass
Should open menu	Check if menu is open after clicking on menu button	Pass
Should close menu	Check if menu is close when clicking outside of it	Pass
Should show correct menu items	For getting correct menu items and also checking add new items into it	Pass
Should click on menu items	Checking menu item is clickable	Pass
Should open submenu	Checking whether submenu is open or not	Pass
Should close submenu	Check if sub menu is close when clicking outside of it	Pass
Should click on submenu-item	Checking sub menu item is clickable	Pass

- Card

Table 5.1.7 Card Testcases

Testcase	Description	Status
Should get and set correct title	Checking of title in card	Pass

Should get and set body content	Checking correct body content is shown	Pass
Should get and set correct footer content	Checking correct footer content is shown	Pass
Should get and set height of the card	Checking the height of the card	Pass
Should get and set width of the card	Checking the width of the card	Pass
Should set card image	Check for correct card image is set or not	Pass
Should change background color of card	Is background color changes of card	Pass
Should change size of card title	Card size is changeable or not	Pass

- Auto Complete

Table 5.1.8 Auto Complete Testcases

Testcase	Description	Status
Should get correct items when click on input box initially	Is all items are shown to user when click on input box	Pass
Should get filter options correctly	When input text changes then items are filtered correctly or not	Pass
Should change input box value when select option	After clicking on option it will be set in input box	Pass
Should change input box background color value when select option	After selecting option background color should be changed	Pass

Should change placeholder correctly	Placeholder should be changed	Pass
Should not take input if it is not in option	User can not type manually option in input	Pass
Should close when click on outside of it	Should be close	Pass

- Tabs

Table 5.1.9 Tabs Testcases

Testcase	Description	Status
Should get correct tab names	Checking correct name of tabs	Pass
It has default tab	One tab is by default active	Pass
Should change default tab	User can change the default tab	Pass
Should select a tab	User should select one tab at a time	Pass
Should change active tab color	Active tab color have background	Pass
Selected tab have bottom border	Active tab color have bottom border	Pass
Should show correct content	Checks whether tab shows correct content or not	Pass
Should disable particular tab	User can disable any tab	Pass
Should set and get correct width of tabs	Checks whether tab width is correct or not	Pass

- Progress bar

Table 5.1.10 Progress bar Testcases

Testcase	Description	Status
Should check progress is indeterminate or not	Checking for progress is determinate or indeterminate	Pass
Should get correct width of progress	Checking the width of progress	Pass
Should get correct progress color	Checking color of progress	Pass
Should get correct height of the progress	Checking height of the progress	Pass
Should get correct width of progress	Checking width of the progress	Pass

- Date Time kit

Table 5.1.11 Date Time Kit Testcases

Testcase	Description	Status
Should open date and time picker	Check whether date time picker is open or not	Pass
Initially It opens with today's date and year	By default it is open with today's date	Pass
Initially It opens with today's date and year	Today's date have background color green	Pass
Change color when hover on date	Background color of date changed when hover on it	Pass
Should change the year	Year should be changed via selection list	Pass
Should change month	Month should be changed in calendar	Pass
Should increase and decrease month	Month is increased or decreased by clicking on left and right arrow in calendar	Pass

Should select the date	User can select the date from the picker by clicking on the date	Pass
Selected date have background color black	After selecting a date it's background color changed into black	Pass
Clear the input when click on clear button	After clicking on clear button input should be empty	Pass
Should select today's date after click on today button	After clicking on today button input should be today's date	Pass
Should select the time	User can able to select time	Pass
If user click outside of picker the it should be close	By clicking outside of picker it should be closed	Pass
Picker should open with current input box date	After select date we change year and month but not select date and then close picker after again opening picker it should open with date of input box	Pass
If time not select then default should be 00:00	By default time is set to 00:00	Pass

- Dialog Component

Table 5.1.12 Dialog Components Testcases

Testcase	Description	Status
Should open dialog when clicking on button	After clicking on button dialog box should be open	Pass
Should close when clicking outside of it	Clicking outside of dialog it should be closed	Pass
Should blur background when dialog is open	After opening dialog background should be blur	Pass
Do not close dialog when click inside of it	It is not close when click inside of it	Pass

Should show correct dialog content	It shows correct dialog content	Pass
Should change width and height of dialog	It can be set height and width by user	Pass

- Paginator

Table 5.1.13 Paginator Testcases

Testcase	Description	Status
Should click on page number	User can click on page number that is visible	Pass
Current page number should be blur	Current selected page's page number should be blur	Pass
Should get correct visible pages	After clicking on next button or previous button visible pages are shown to be correct	Pass
Should go next page when click on next button	After clicking on next button page is change to next page	Pass
Should go previous page when click on prev button	After clicking on prev button page is change to previous page	Pass
Should go first page when click on first button	After clicking on first button page is change to first page	Pass
Should go last page when click on last button	After clicking on last button page is change to last page	Pass
Should select items per page	User can changed items per page in paginator	Pass
Should show default items per page content	After changing the items per page items list also be changed	Pass
Should change content when	After changing the page content	Pass

changing page	should also be change	

- Form

Table 5.1.14 Form Testcases

Testcase	Description	Status
Should get correct form title	Form title can be changed	Pass
Should get and set width and height of the form	Form height and width can be changeable	pass
Should get and set content of the form	Form content should be correctly set and get	Pass
Should embedded different component inside form	Other components are also be embedded into form like input, radio button, auto complete etc.	Pass

Chapter 6
Analysis

6.1 Screen shots

Menu :

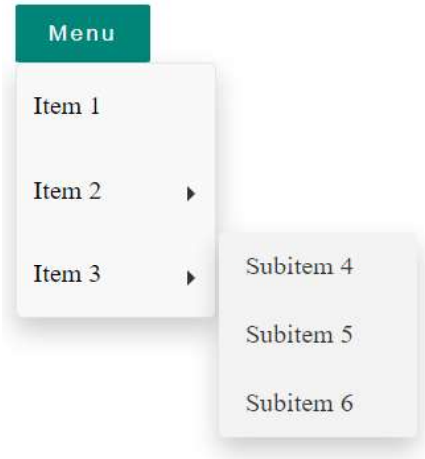

Fig 6.1.1 Menu component

Tabs :

Fig 6.1.2 Tabs component

Input :

Fig 6.1.3 Input component

Paginator :

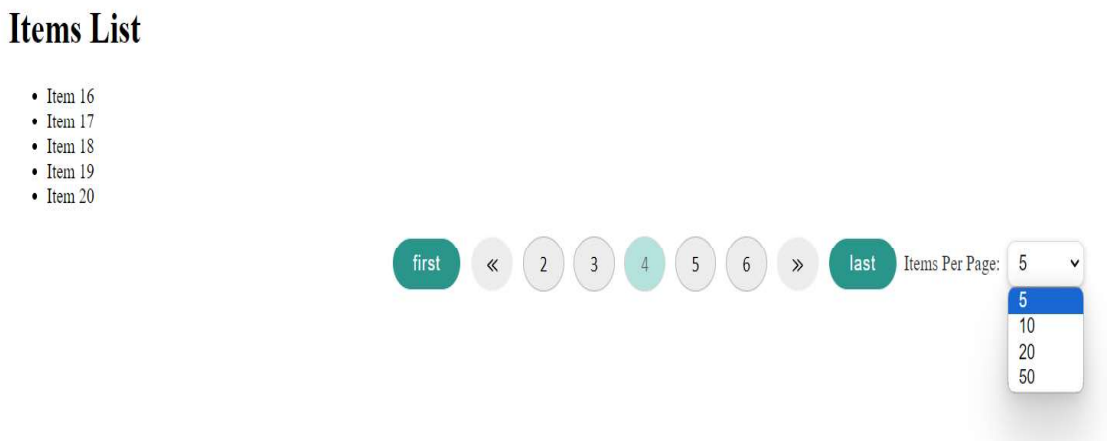

Fig 6.1.4 Paginator component

Auto-Complete:

Fig 6.1.5 Auto-Complete component

Date-Time kit :

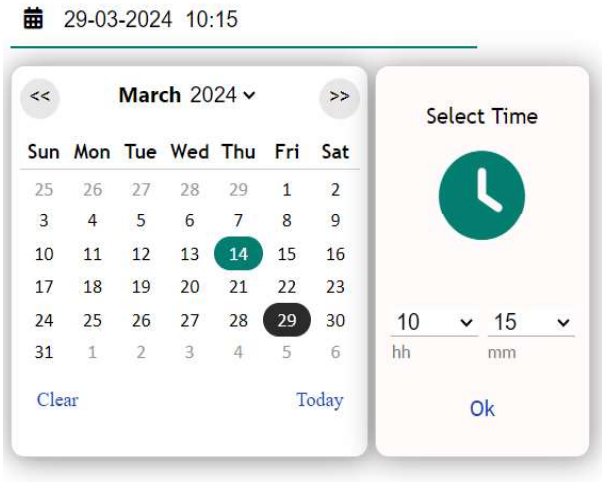

Fig 6.1.6 Date-Time Kit component

Navbar :

Fig 6.1.7 Navbar component

Chapter 7
Conclusion

7.1 Conclusion

In conclusion, harness classes play a vital role in Angular component testing, offering structured access to elements and properties for streamlined testing processes. By identifying component elements, creating corresponding harness classes, leveraging Angular testing utilities, and ensuring simplicity and maintainability, developers can effectively design harness classes. Strong typing and adherence to testing best practices further enhance the reliability and efficiency of component testing. Harness classes simplify Angular testing workflows, contributing to improved code quality and faster development cycles.

7.2 Future Extension

Looking forward, the evolution of harness classes for Angular components holds several opportunities for expansion and enhancement. One avenue involves extending testing capabilities to cover a wider range of scenarios, including complex interactions and asynchronous operations.

Additionally, integrating harness classes with emerging UI frameworks and libraries can ensure compatibility across diverse development environments. Automation and tooling development could streamline the generation and maintenance of harness classes, while enhanced IDE support could improve developer productivity.

Collaborating with the Angular community and participating in standardization efforts would foster best practices and interoperability. Moreover, optimizing harness class implementations for performance could lead to faster test execution and more efficient development workflows. By exploring these avenues, harness classes can continue to evolve, meeting the dynamic demands of Angular development and enabling robust, reliable testing practices.

Chapter 8
Bibliography

Bibliography

For completing this project all the references are mentioned below:

[1] Reference for Angular :

https://angular.io/docs

https://www.udemy.com/course/the-complete-guide-to-angular-2/

[2] Reference for TypeScript :

https://www.typescriptlang.org/docs/

https://www.udemy.com/course/understanding-typescript/

[3] Reference for Harness :

https://material.angular.io/cdk/test-harnesses/overview

[4] Reference for CSS:

https://www.w3schools.com/css/