**JS Framework**

**Project Name :** TREKK-TOUR

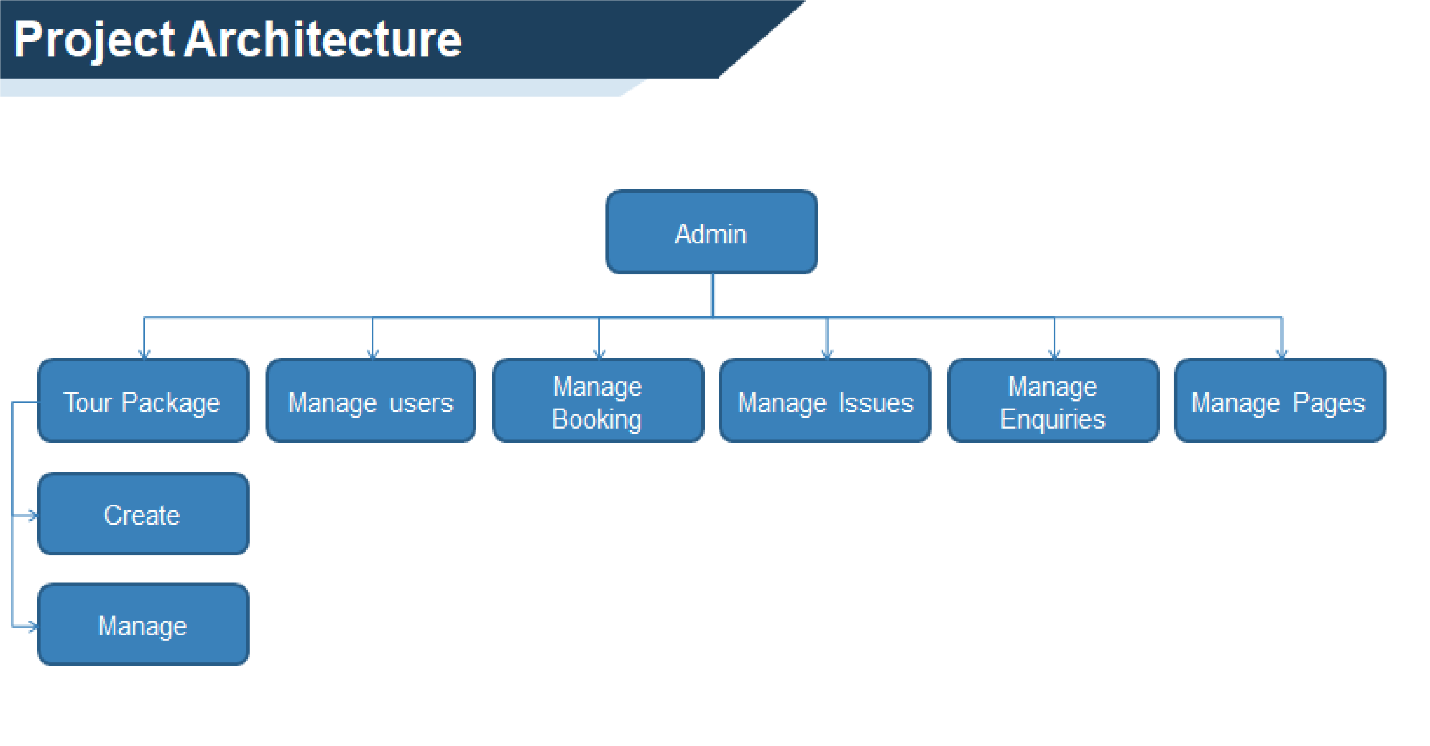
Duration : 1 year 2 months

Team size : 4

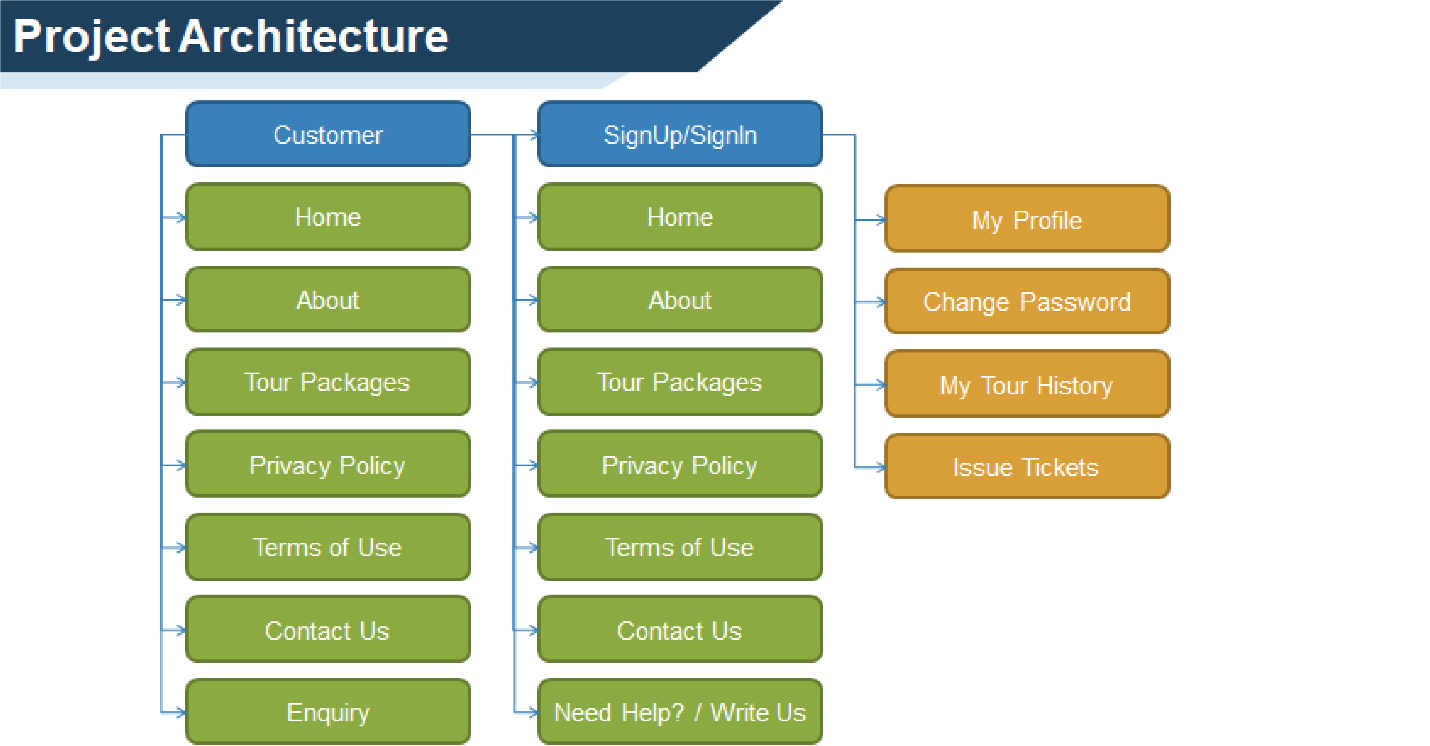
Environment : JavaScript, WebDriverIO, Mocha-Chai Framework, Visual Studio Code Editor, GitHub, Jira

Project Description :

* TREKK-TOUR is mainly focused on managing the tour packages and their customer data. TREKK-TOUR represents tourist locations visually through Images and Videos. Using this project, customer can globalize and run their business 24\*7.
* In TREKK-TOUR, Administrator will have the authority to create the tourist package. Administrator can manage the enquires and issues raised by the end-users. Administrator can manage the bookings done by the end-users. This project will reduce the paper-work and increase the customer relationship with the client.



My project is based on JavaScript WebDriver-IO Mocha Framework, wherein validations are done with CHAI Assertions.



**Why MOCHA framework ?**

Mocha is a feature-rich JavaScript-based testing framework that simplifies asynchronous testing. This framework’s additional ability is to run tests serially with greater accuracy without sacrificing the speed.

**Advantages of MOCHA framework**

* Can be installed globally, as a Dependency for the project.
* Supports all the Major Browsers.
* Different Reporter options like Spec, Allure, JUnit, JSON, HTML and so on.
* JS Assertion libraries support like chai.js, express.js, should.js and so on.
* Supports both BDD and TDD environments.
* Supports both Synchronous and Asynchronous Testing.

**Page Object Pattern:**

In this Framework, Page Object Pattern has been implemented. The concept of Page Object Pattern is simple i.e., for **1** **webpage**, **1 page object** must be created.

By the automation standards, web element/elements should not be hard coded in the test spec. Hence all the elements in the webpage are being put into a page object and Business Libraries are developed as functions according to the requirement, in their respective Page Objects.

**package.json :**

package.json is a JavaScript Object Notation file where it consists of all the dependencies of the Project along with the versions.

By default, during creation of framework mocha-framework, chromedriver, spec-reporter dependencies are added.

External Dependencies like chai, selenium-standalone, mocha-framework, allure-reporter are added after configurations are done or it can be added manually.

**cmd 🡪 npm install** – used to install all the dependencies with the versions defined in **package.json**

**wdio.conf.js :**

By this wdio.conf.js file, we can achieve Parallel and Compatibility Testing

* wdio.conf.js is a configuration file which consist of all the configurations necessary to run the Test Spec.
* **Spec** array consist of the relative path of the files or a particular test spec in the form of array. The execution will happen based on the index values.
* **exclude** array is used to exclude a particular or a set of test spec
* **suites** Block is used to create suites and clubs all the related test specs under a single suite file and . **npx wdio run ./wdio.conf.js - -suite <suiteName>** is the command used to run a particular suite file defined in the Suites block.
* **capabilities** array in configuration file is used to define the properties of the browser. Each property for the browser can be defined with no. of instances.
* **maxInstances** is used to define total number of browsers that can run parallelly. It depends on the instances given for each browser in the Capabilities array. Maximum instances that can be defined and created is **10.**
* **bail** defines the count in which, if certain number of Test Spec is failing, it will stop the execution of the Test Specs.
  + If **bail : 1** ,the spec execution will terminate if **1 Test Spec fails**. This is mostly used to perform Smoke Suite Execution.
* **connectionRetryTimeout** will define time to wait if the connection is lost or any of the network issue/server issues occurs.
* **connectionRetryCount** will define the number of retry attempts that has to be executed if any connection issue occurs
* **services** array will define the execution i.e., if the Test Spec must run in a specific browser or Compatibility Testing (Cross-Browser Testing) should be done.
* **reporters** array consists of the report type used in the framework. Here we are using
  + SPEC reports
    - SPEC report consists of results in the form of logs. It consists of TICK (✓) symbol if Test Spec Passes and CROSS (x) symbol if the Test Spec Fails.
  + Allure Reports
    - Allure reports are generated in customer’s perspective. As Test Engineers, we can understand by reading the logs generated in the SPEC report. But customers can’t. For this reason, we go for Allure Reports. This report has an UI where customers can understand the report.
      * **disableWebdriverScreenshotsReporting**: In Reporter array, by default, this property will be defined as “**true**”. This will take screenshot wherever a test spec passes. This must be changed to **“false”.** By this, only when a test spec fails, screenshot will be taken.
    - This report is generated at the end of a Release for the customer.
* **hooks** are WebDriver IO annotations used to define certain conditions that must be defined before execution of Test Specs.
  + **before – executes before a webdriver io command is executed -** used to declare any global variables.
  + **afterTest –** executes after a test run. Used to take screenshots whenever Test Spec fails.

**Commands used to run tests:**

* **npm run <keyword> –** runs all the specs defined in the **scripts** block in **package.json**
* **npx wdio run ./wdio.conf.js - - spec <relative path of the test spec>** - executes a particular test spec
* **npx wdio run ./wdio.conf.js - - mochaOpts.grep <keyword> -** executes describe/it block associated with certain keyword
* **npx wdio run ./wdio.conf.js - - suite <suiteName> -** executes a suite file consisting of certain spec files included/defined.
* **npm install -g allure-commandline --save-dev 🡪** Installs the Allure Reporter in the framework.
* **allure generate <relative-path-of the-folder> -** generates allure report to a specific folder.
* **allure open –** Opens the Generated Allure Report.

**CHAI Assertions :**

In this framework, CHAI assertions have been implemented. Unlike WebDriver IO Assertions, there are wide variety of assert conditions present in CHAI.

There are different Assertion Styles like

* **Expect/Should** API with BDD Assertion Styles.
* **Assert** API with TDD Assertion Styles.