**Protractor Configuration Synatx in TypeScript**

```
import {PluginConfig} from './plugins';

export interface Config {
  [key: string]: any;

  // ------------------------------------------------------------------------
  // ----- How to connect to Browser Drivers ----------------------------------
  // ------------------------------------------------------------------------
  //
  // Protractor needs to know how to connect to Drivers for the browsers
  // it is testing on. This is usually done through a Selenium Server.
  // There are five options - specify one of the following:
  //
  // 1. seleniumServerJar - to start a standalone Selenium Server locally.
  // 2. seleniumAddress - to connect to a Selenium Server which is already
  //    running.
  // 3. sauceUser/sauceKey - to use remote Selenium Servers via Sauce Labs.
  // 4. browserstackUser/browserstackKey - to use remote Selenium Servers via
  // BrowserStack.
  // 5. directConnect - to connect directly to the browser Drivers.
  //    This option is only available for Firefox and Chrome.
  //
  // ---- 1. To start a standalone Selenium Server locally ----------

  /**
   * The location of the standalone Selenium Server jar file, relative
   * to the location of webdriver-manager. If no other method of starting
   * Selenium Server is found, this will default to
   * node_modules/protractor/node_modules/webdriver-manager/selenium/<jar file>
   */
  seleniumServerJar?: string;

  /**
   * The timeout milliseconds waiting for a local standalone Selenium Server to start.
   *
   * default: 30000ms
   */
  seleniumServerStartTimeout?: number;
```

```
 /**
   * Can be an object which will be passed to the SeleniumServer class as args.
   * See a full list of options at
   * https://github.com/SeleniumHQ/selenium/blob/master/javascript/node/selenium-
webdriver/remote/index.js
   * If you specify `args` or `port` in this object, it will overwrite the
   * values set via the deprecated config values `seleniumPort` and
   * `seleniumArgs`.
   */
  localSeleniumStandaloneOpts?: {
    /**
     * The port to start the Selenium Server on, or null if the server should
     * find its own unused port.
     */
    port?: any;
    /**
     * Additional command line options to pass to selenium. For example,
     * if you need to change the browser timeout, use
     * seleniumArgs: ['-browserTimeout=60']
     */
    args?: any;

    /**
     * Additional command line jvm options to pass to selenium. For example,
     * if you need to change the browser driver, use
     * jvmArgs: ['-Dwebdriver.ie.driver=IEDriverServer_Win32_2.53.1.exe']
     */
    jvmArgs?: string[];
  };
  /**
   * ChromeDriver location is used to help find the chromedriver binary.
   * This will be passed to the Selenium jar as the system property
   * webdriver.chrome.driver. If null, Selenium will attempt to find
   * ChromeDriver using PATH.
   *
   * example:
   * chromeDriver: './node_modules/webdriver-manager/selenium/chromedriver_2.20'
   */
  chromeDriver?: string;

  // ---- 2. To connect to a Selenium Server which is already running ----------
```

```
/**
 * The address of a running Selenium Server. If specified, Protractor will
 * connect to an already running instance of Selenium. This usually looks like
 * seleniumAddress: 'http://localhost:4444/wd/hub'
 */
seleniumAddress?: string;
/**
 * The selenium session id allows Protractor to attach to an existing selenium
 * browser session. The selenium session is maintained after the test has
 * completed. Ignored if seleniumAddress is null.
 */
seleniumSessionId?: string;
/**
 * The address of a proxy server to use for the connection to the
 * Selenium Server. If not specified no proxy is configured. Looks like
 * webDriverProxy: 'http://localhost:3128'
 */
webDriverProxy?: string;

/**
 * If specified, connect to webdriver through a proxy that manages client-side
 * synchronization. Blocking Proxy is an experimental feature and may change
 * without notice.
 */
useBlockingProxy?: boolean;

/**
 * If specified, Protractor will connect to the Blocking Proxy at the given
 * url instead of starting it's own.
 */
blockingProxyUrl?: string;

// ---- 3. To use remote browsers via Sauce Labs ----------------------------

/**
 * If the sauceUser and sauceKey are specified, seleniumServerJar will be
 * ignored. The tests will be run remotely using Sauce Labs.
 */
sauceUser?: string;
/**
 * If the sauceUser and sauceKey are specified, seleniumServerJar will be
 * ignored. The tests will be run remotely using Sauce Labs.
 */
sauceKey?: string;
```

```
/**
 * Use sauceAgent if you need custom HTTP agent to connect to saucelabs.com.
 * This is needed if your computer is behind a corporate proxy.
 *
 * To match sauce agent implementation, use
 * [HttpProxyAgent](https://github.com/TooTallNate/node-http-proxy-agent)
 * to generate the agent or use webDriverProxy as an alternative. If a
 * webDriverProxy is provided, the sauceAgent will be overridden.
 */
sauceAgent?: any;
/**
 * Use sauceBuild if you want to group test capabilites by a build ID
 */
sauceBuild?: string;
/**
 * If true, Protractor will use http:// protocol instead of https:// to
 * connect to Sauce Labs defined by sauceSeleniumAddress.
 *
 * default: false
 */
sauceSeleniumUseHttp?: boolean;
/**
 * Use sauceSeleniumAddress if you need to customize the URL Protractor
 * uses to connect to sauce labs (for example, if you are tunneling selenium
 * traffic through a sauce connect tunnel). Default is
 * ondemand.saucelabs.com:80/wd/hub
 */
sauceSeleniumAddress?: string;

// ---- 4. To use remote browsers via BrowserStack --------------------------

/**
 * If browserstackUser and browserstackKey are specified, seleniumServerJar
 * will be ignored. The tests will be run remotely using BrowserStack.
 */
browserstackUser?: string;
/**
 * If browserstackUser and browserstackKey are specified, seleniumServerJar
 * will be ignored. The tests will be run remotely using BrowserStack.
 */
browserstackKey?: string;
```

```
// ---- 5. To connect directly to Drivers ----------------------------------

/**
 * If true, Protractor will connect directly to the browser Drivers
 * at the locations specified by chromeDriver and firefoxPath. Only Chrome
 * and Firefox are supported for direct connect.
 *
 * default: false
 */
directConnect?: boolean;
/**
 * Path to the firefox application binary. If null, will attempt to find
 * firefox in the default locations.
 */
firefoxPath?: string;

// ------------------------------------------------------------------------
// ----- What tests to run ------------------------------------------------
// ------------------------------------------------------------------------

/**
 * Use default globals: 'protractor', 'browser', '$', '$$', 'element', 'by'.
 * These also exist as properties of the protractor namespace:
 * 'protractor.browser', 'protractor.$', 'protractor.$$',
 * 'protractor.element', 'protractor.by', and 'protractor.By'.
 *
 * When no globals is set to true, the only available global variable will be
 * 'protractor'.
 */
noGlobals?: boolean;

/**
 * Required. Spec patterns are relative to the location of this config.
 *
 * Example:
 * specs: [
 *   'spec/*_spec.js'
 * ]
 */
specs?: Array<string>;
```

```
/**
 * Patterns to exclude specs.
 */
exclude?: Array<string>|string;

/**
 * Alternatively, suites may be used. When run without a command line
 * parameter, all suites will run. If run with --suite=smoke or
 * --suite=smoke,full only the patterns matched by the specified suites will
 * run.
 *
 * Example:
 * suites: {
 *   smoke: 'spec/smoketests/*.js',
 *   full: 'spec/*.js'
 * }
 */
suites?: any;
/**
 * If you would like protractor to use a specific suite by default instead of
 * all suites, you can put that in the config file as well.
 */
suite?: string;

// ------------------------------------------------------------------------
// ----- How to set up browsers ---------------------------------------------
// ------------------------------------------------------------------------

/**
 * Protractor can launch your tests on one or more browsers. If you are
 * testing on a single browser, use the capabilities option. If you are
 * testing on multiple browsers, use the multiCapabilities array.
 *
 * For a list of available capabilities, see
 * https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities
 * In addition, you may specify count, shardTestFiles, and maxInstances.
 *
 * Example:
 * capabilities: {
 *   browserName: 'chrome',
 *   name: 'Unnamed Job',
 *   logName: 'Chrome - English',
 *   count: 1,
 *   shardTestFiles: false,
```

```
 *   maxInstances: 1,
 *   specs: ['spec/chromeOnlySpec.js'],
 *   exclude: ['spec/doNotRunInChromeSpec.js'],
 *   seleniumAddress: 'http://localhost:4444/wd/hub'
 * }


 */
capabilities?: {

  [key: string]: any;

  browserName?: string;

  /**
   * Name of the process executing this capability.  Not used directly by
   * protractor or the browser, but instead pass directly to third parties
   * like BrowserStack and SauceLabs as the name of the job running this
   * test
   */
  name?: string;

  /**
   * User defined name for the capability that will display in the results
   * log. Defaults to the browser name
   */
  logName?: string;

  /**
   * Number of times to run this set of capabilities (in parallel, unless
   * limited by maxSessions). Default is 1.
   */
  count?: number;

  /**
   * If this is set to be true, specs will be sharded by file (i.e. all
   * files to be run by this set of capabilities will run in parallel).
   * Default is false.
   */
  shardTestFiles?: boolean;
```

```
  /**
   * Maximum number of browser instances that can run in parallel for this
   * set of capabilities. This is only needed if shardTestFiles is true.
   * Default is 1.
   */
  maxInstances?: number;

  /**
   * Additional spec files to be run on this capability only.
   */
  specs?: string[];

  /**
   * Spec files to be excluded on this capability only.
   */
  exclude?: string[];

  /**
   * Optional: override global seleniumAddress on this capability only.
   */
  seleniumAddress?: string;

  // Optional: Additional third-party specific capabilities can be
  // specified here.
  // For a list of BrowserStack specific capabilities, visit
  // https://www.browserstack.com/automate/capabilities
};

/**
 * If you would like to run more than one instance of WebDriver on the same
 * tests, use multiCapabilities, which takes an array of capabilities.
 * If this is specified, capabilities will be ignored.
 */
multiCapabilities?: Array<any>;

/**
 * If you need to resolve multiCapabilities asynchronously (i.e. wait for
 * server/proxy, set firefox profile, etc), you can specify a function here
 * which will return either `multiCapabilities` or a promise to
 * `multiCapabilities`.
 *
 * If this returns a promise, it is resolved immediately after
 * `beforeLaunch` is run, and before any driver is set up. If this is
 * specified, both capabilities and multiCapabilities will be ignored.
 */
```

```
getMultiCapabilities?: any;

  /**
   * Maximum number of total browser sessions to run. Tests are queued in
   * sequence if number of browser sessions is limited by this parameter.
   * Use a number less than 1 to denote unlimited. Default is unlimited.
   */
  maxSessions?: number;

  /**
   * Whether or not to buffer output when running tests on multiple browsers
   * in parallel. By default, when running multiple browser sessions, the
   * results are buffered and not logged until the test run finishes. If true,
   * when running multiple sessions in parallel results will be logged when
   * each test finishes.
   */
  verboseMultiSessions?: boolean;

  // ------------------------------------------------------------------------
  // ----- Global test information
  // -----------------------------------------------
  // ------------------------------------------------------------------------

  /**
   * A base URL for your application under test. Calls to protractor.get()
   * with relative paths will be resolved against this URL (via url.resolve)
   */
  baseUrl?: string;

  /**
   * A CSS Selector for a DOM element within your Angular application.
   * Protractor will attempt to automatically find your application, but it is
   * necessary to set rootElement in certain cases.
   *
   * In Angular 1, Protractor will use the element your app bootstrapped to by
   * default.  If that doesn't work, it will then search for hooks in `body` or
   * `ng-app` elements (details here: https://git.io/v1b2r).
   *
   * In later versions of Angular, Protractor will try to hook into all angular
   * apps on the page. Use rootElement to limit the scope of which apps
   * Protractor waits for and searches within.
   */
  rootElement?: string;
```

```
/**
 * The timeout in milliseconds for each script run on the browser. This
 * should be longer than the maximum time your application needs to
 * stabilize between tasks.
 */
allScriptsTimeout?: number;

/**
 * How long to wait for a page to load.
 */
getPageTimeout?: number;

/**
 * A callback function called once configs are read but before any
 * environment setup. This will only run once, and before onPrepare.
 *
 * You can specify a file containing code to run by setting beforeLaunch to
 * the filename string.
 *
 * At this point, global variable 'protractor' object will NOT be set up,
 * and globals from the test framework will NOT be available. The main
 * purpose of this function should be to bring up test dependencies.
 */
beforeLaunch?: () => void;

/**
 * A callback function called once protractor is ready and available, and
 * before the specs are executed. If multiple capabilities are being run,
 * this will run once per capability.
 *
 * You can specify a file containing code to run by setting onPrepare to
 * the filename string. onPrepare can optionally return a promise, which
 * Protractor will wait for before continuing execution. This can be used if
 * the preparation involves any asynchronous calls, e.g. interacting with
 * the browser. Otherwise Protractor cannot guarantee order of execution
 * and may start the tests before preparation finishes.
 *
 * At this point, global variable 'protractor' object will be set up, and
 * globals from the test framework will be available. For example, if you
 * are using Jasmine, you can add a reporter with:
 *
 *    jasmine.getEnv().addReporter(new jasmine.JUnitXmlReporter(
 *      'outputdir/', true, true));
 *
 * If you need access back to the current configuration object,
```

```
 * use a pattern like the following:
 *
 *    return browser.getProcessedConfig().then(function(config) {
 *      // config.capabilities is the CURRENT capability being run, if
 *      // you are using multiCapabilities.
 *      console.log('Executing capability', config.capabilities);
 *    });
 */
onPrepare?: () => void;

/**
 * A callback function called once tests are finished. onComplete can
 * optionally return a promise, which Protractor will wait for before
 * shutting down webdriver.
 *
 * At this point, tests will be done but global objects will still be
 * available.
 */
onComplete?: () => void;


/**
 * A callback function called once the tests have finished running and
 * the WebDriver instance has been shut down. It is passed the exit code
 * (0 if the tests passed). This is called once per capability.
 */
onCleanUp?: (exitCode: number) => void;

/**
 * A callback function called once all tests have finished running and
 * the WebDriver instance has been shut down. It is passed the exit code
 * (0 if the tests passed). afterLaunch must return a promise if you want
 * asynchronous code to be executed before the program exits.
 * This is called only once before the program exits (after onCleanUp).
 */
afterLaunch?: (exitCode: number) => void;

/**
 * The params object will be passed directly to the Protractor instance,
 * and can be accessed from your test as browser.params. It is an arbitrary
 * object and can contain anything you may need in your test.
 * This can be changed via the command line as:
 *   --params.login.user "Joe"
 *
```

```
 * Example:
 * params: {
 *   login: {
 *     user: 'Jane',
 *     password: '1234'
 *   }
 * }
 */
params?: any;

/**
 * If set, protractor will save the test output in json format at this path.
 * The path is relative to the location of this config.
 */
resultJsonOutputFile?: any;

/**
 * If true, protractor will restart the browser between each test. Default
 * value is false.
 *
 * CAUTION: This will cause your tests to slow down drastically.
 */
restartBrowserBetweenTests?: boolean;

/**
 * Protractor will track outstanding $timeouts by default, and report them
 * in the error message if Protractor fails to synchronize with Angular in
 * time. In order to do this Protractor needs to decorate $timeout.
 *
 * CAUTION: If your app decorates $timeout, you must turn on this flag. This
 * is false by default.
 */
untrackOutstandingTimeouts?: boolean;

/**
 * If set, Protractor will ignore uncaught exceptions instead of exiting
 * without an error code. The exceptions will still be logged as warnings.
 */
ignoreUncaughtExceptions?: boolean;

/**
 * If set, will create a log file in the given directory with a readable log of
 * the webdriver commands it executes.
```

```
 *
 * This is an experimental feature. Enabling this will also turn on Blocking Proxy
 * synchronization, which is also experimental.
 */
webDriverLogDir?: string;

/**
 * If set, Protractor will pause the specified amount of time (in milliseconds)
 * before interactions with browser elements (ie, sending keys, clicking). It will
 * also highlight the element it's about to interact with.
 *
 * This is an experimental feature. Enabling this will also turn on Blocking Proxy
 * synchronization, which is also experimental.
 */
highlightDelay?: number;

// ----------------------------------------------------------------------
// ----- The test framework
// ----------------------------------------------------
// ----------------------------------------------------------------------

/**
 * Test framework to use. This may be one of: jasmine, mocha or custom.
 * Default value is 'jasmine'
 *
 * When the framework is set to "custom" you'll need to additionally
 * set frameworkPath with the path relative to the config file or absolute:
 *
 *   framework: 'custom',
 *   frameworkPath: './frameworks/my_custom_jasmine.js',
 *
 * See github.com/angular/protractor/blob/master/lib/frameworks/README.md
 * to comply with the interface details of your custom implementation.
 *
 * Jasmine is fully supported as test and assertion frameworks.
 * Mocha has limited support. You will need to include your
 * own assertion framework (such as Chai) if working with Mocha.
 */
framework?: string;

/**
 * Options to be passed to jasmine.
 *
 * See https://github.com/jasmine/jasmine-npm/blob/master/lib/jasmine.js
 * for the exact options available.
 */
```

```
jasmineNodeOpts?: {
    [key: string]: any;
    /**
     * If true, print colors to the terminal.
     */
    showColors?: boolean;
    /**
     * Default time to wait in ms before a test fails.
     */
    defaultTimeoutInterval?: number;
    /**
     * Function called to print jasmine results.
     */
    print?: () => void;
    /**
     * If set, only execute specs whose names match the pattern, which is
     * internally compiled to a RegExp.
     */
    grep?: string;
    /**
     * Inverts 'grep' matches
     */
    invertGrep?: boolean;
    /**
     * If true, run specs in semi-random order
     */
    random?: boolean,
    /**
     * Set the randomization seed if randomization is turned on
     */
    seed?: string,
  };

  /**
   * Options to be passed to Mocha.
   *
   * See the full list at http://mochajs.org/
   */
  mochaOpts?: {[key: string]: any; ui?: string; reporter?: string;};

  /**
   * See docs/plugins.md
   */
  plugins?: PluginConfig[];
```

```
/**
   * Turns off source map support.  Stops protractor from registering global
   * variable `source-map-support`.  Defaults to `false`
   */
  skipSourceMapSupport?: boolean;

  /**
   * Turns off WebDriver's environment variables overrides to ignore any
   * environment variable and to only use the configuration in this file.
   * Defaults to `false`
   */
  disableEnvironmentOverrides?: boolean;

  /**
   * Tells Protractor to interpret any angular apps it comes across as hybrid
   * angular1/angular2 apps (i.e. apps using ngUpgrade)
   * Defaults to `false`
   *
   * @type {boolean}
   */
  ng12Hybrid?: boolean;

  /**
   * Protractor will exit with an error if it sees any command line flags it doesn't
   * recognize. Set disableChecks true to disable this check.
   */
  disableChecks?: boolean;

  /**
   * Enable/disable the WebDriver Control Flow.
   *
   * WebDriverJS (and by extention, Protractor) uses a Control Flow to manage the
order in which
   * commands are executed and promises are resolved (see docs/control-flow.md for
details).
   * However, as syntax like `async`/`await` are being introduced, WebDriverJS has
decided to
   * deprecate the control flow, and have users manage the asynchronous activity
themselves
   * (details here: https://github.com/SeleniumHQ/selenium/issues/2969).
   *
   * At the moment, the WebDriver Control Flow is still enabled by default. You can
disable it by
   * setting the environment variable `SELENIUM_PROMISE_MANAGER` to `0`.  In a
webdriver release in
```

```
   * Q4 2017, the Control Flow will be disabled by default, but you will be able to re-
enable it by
   * setting `SELENIUM_PROMISE_MANAGER` to `1`.  At a later point, the control flow
will be removed
   * for good.
   *
   * If you don't like managing environment variables, you can set this option in your
config file,
   * and Protractor will handle enabling/disabling the control flow for you.  Setting this
option
   * is higher priority than the `SELENIUM_PROMISE_MANAGER` environment
variable.
   *
   * @type {boolean=}
   */
  SELENIUM_PROMISE_MANAGER?: boolean;

  seleniumArgs?: any[];
  jvmArgs?: string[];
  configDir?: string;
  troubleshoot?: boolean;
  seleniumPort?: number;
  mockSelenium?: boolean;
  v8Debug?: any;
  nodeDebug?: boolean;
  debuggerServerPort?: number;
  frameworkPath?: string;
  elementExplorer?: any;
  debug?: boolean;
  unknownFlags_?: string[];
}
```

**********