|  |
| --- |
| React & Typescript Notes |
|  |
| September 28  COMPANY NAME  Authored by: Your Name |



Table of Contents

[React 3](#_Toc209965983)

[1.2 Typescript Improves Developer Experience in React Projects 3](#_Toc209965984)

[1.3 Comparing Prop Types vs Typescript in React 4](#_Toc209965985)

[Summary Table 4](#_Toc209965986)

[Typescript 1](#_Toc209965987)

[Subtitle Text Here 1](#_Toc209965988)

[Selenium 1](#_Toc209965989)

[Subtitle Text Here 1](#_Toc209965990)

[Playwright 1](#_Toc209965991)

[Subtitle Text Here 1](#_Toc209965992)

[Python 1](#_Toc209965993)

[Subtitle Text Here 1](#_Toc209965994)

[Java 1](#_Toc209965995)

[Subtitle Text Here 1](#_Toc209965996)

[React 2](#_Toc209965997)

[Subtitle Text Here 2](#_Toc209965998)

# React

Module 1: Introduction to React with Typescript

**1.1 Why Typescript with React?**

Typescript is widely adopted in React development for its type safety, improved tooling, and scalability benefits.

* **Type safety in components**: Typescript ensures variables, props, and state are explicitly typed, which significantly reduces runtime errors and catches bugs during the development stage. **Enhanced developer tooling**: Typescript’s robust integration with IDEs like VS Code provides intelligent code completion, auto import, and real-time error feedback, greatly improving workflow.
* **Early bug detection**: With compile-time checks, errors are usually identified before code reaches production, minimizing critical failures and speeding up debugging.
* **Self-documenting code**: Explicit type definitions make code more readable and self-explanatory, serving as living documentation for developers and easing on boarding of new team members.
* **Scalability and maintainability**: Typescript enforces structure, making large codebases easier to manage, refactor, and scale as teams grow or project complexity increases.

## **1.2 Typescript Improves Developer Experience in React Projects**

* **Automatic IntelliSense**: Typescript provides smarter autocomplete and suggestions for component props, methods, and variable names, making coding faster and less error-prone.
* **Clear and actionable error messages**: Compile-time errors come with descriptive messages, assisting developers in resolving issues quickly.
* **Code navigation and refactoring**: Typescript lets developers refactor confidently with features like “go to definition” and refactoring support in IDEs, boosting productivity and consistency.
* **Better collaboration**: In teams, explicit types and interfaces let developers understand, review, and extend each other’s code without ambiguity, reducing on boarding time.
* **Reduced maintenance**: Strong typing prevents subtle bugs and makes long-term maintenance simpler, helping developers avoid spending excessive time on troubleshooting.

## **1.3 Comparing Prop Types vs Typescript in React**

|  |  |  |
| --- | --- | --- |
| **Feature** | **PropTypes** | **TypeScript** |
| **Type Checking Stage** | Runtime | Compile-time |
| **Error Detection** | Detected when running app | Detected during development |
| **IDE Support** | Limited autocomplete | Advanced IntelliSense |
| **Documentation** | Manual, less explicit | Self-documenting types |
| **Scalability** | May require duplication | More scalable, reusable types |
| **Advanced Usage** | Basic shape/type checks | Complex conditional/union types |

* Prop Types validate only at runtime and are best for simple apps or sharing libraries with non-Typescript users.
* Typescript provides stronger static typing and developer tooling, enabling advanced type relationships, better documentation, and earlier bug detection.
* Some teams combine both, but Typescript generally replaces Prop Types in Typescript-based React apps for efficiency and type safety.

## Summary Table

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Typescript + React** | **Prop Types + React** |
| **Type Safety** | Strong and enforced | Basic, runtime-only |
| **Developer Experience** | Advanced tooling, docs | Limited, manual |
| **Error Detection** | Early, at compile time | Late, at runtime |
| **Scalability** | Excellent for large teams | Less suited |

**1.2 Setting Up Development Environment**

* Choosing the Right Boilerplate (CRA vs Vite vs Next.js)
* TypeScript Configuration (tsconfig.json)
* Essential Development Dependencies
* IDE Setup and Extensions (VS Code Recommended)
* Debugging Configuration
* **Interview Questions**:

**What's the difference between Create React App and Vite for TypeScript?**

Vite features much faster start up and build times, more flexible configuration, and a simpler plugin ecosystem, while CRA is better for stable legacy browser support but comes with slower build, less flexibility, and is reliant on Web pack.

**How do you configure TypeScript for strict mode in React?**

Add "strict": true in the tsconfig.json file, which activates the strict mode family of checks, catching implicit any types, strict null checks, and ensuring strong type safety throughout the code base.

**What are the essential @types packages needed for React?**

Primarily, @types/react and @types/react-dom ensure correct Typescript integration with core React functionality. Most other type packages (e.g., @types/react-router-dom, @types/jest) are installed per the additional packages chosen for the project.

**Single page application vs multipage application**

Single Page Applications (SPA) and Multi-Page Applications (MPA) are distinct web development architectures, each serving different needs and scenarios.

**Core Differences**

| **Aspect** | **Single Page Application (SPA)** | **Multi-Page Application (MPA)** |
| --- | --- | --- |
| Loading | Loads resources once, updates content dynamically without full reloads | Loads a new HTML page from the server for each action or view change |
| User Experience | Seamless, fast navigation and transitions, app-like feel | Traditional browsing experience, often slower as full page loads occur |
| Development Complexity | More complex client-side routing, state handling | Simpler, each page is its own endpoint |
| SEO | Challenging (requires extra setup for good SEO, e.g., SSR) | Excellent (each page can be indexed separately by search engines) |
| Performance | Fast navigation after initial load, but initial load can be heavy | Fast initial load; subsequent navigation slower due to full page reloads |
| Scalability | Suited for highly interactive, dynamic apps (e.g., dashboards) | Preferred for large, content-heavy sites (e.g., e-commerce, news) |
| Offline Support | Easier to provide with service workers and caching | Complex to implement offline for each page |

**When to Use Each**

* SPA: Ideal for dynamic apps with lots of user interaction, complex state, and desktop-like experiences (e.g., dashboards, social apps).
* MPA: Better for large, content-driven sites that prioritize SEO, analytics, and require separate entry points (e.g., blogs, online stores).

**Pros and Cons**

SPA Pros:

* Faster, smoother user experience after initial load.
* Easier to create mobile-friendly and offline-capable apps.

SPA Cons:

* Longer initial load time.
* SEO and analytics setup is more challenging.

MPA Pros:

* Great SEO and analytics.
* Easier to scale content and manage static resources.

MPA Cons:

* Full-page reloads make navigation feel slower.
* More code duplication, less dynamic interaction possible.

Choosing between SPA and MPA depends on project requirements, SEO importance, level of interactivity, and scalability needs

**1.3 Project Structure and Organization**

* TypeScript-specific Folder Structure
* Separating Types, Interfaces, and Components
* Barrel Exports for Clean Imports
* Environment Variables with Type Safety
* Configuration Files Organization
* **Interview Questions**:
  + How do you organize TypeScript types in a large React project?
  + What are barrel exports and why are they useful?
  + How do you handle environment variables with TypeScript?

**1.4 TypeScript Compiler and Build Process**

* Understanding tsconfig.json Options
* Compilation Targets and Module Systems
* Source Maps for Debugging
* Build Optimization Techniques
* Handling Type Declarations
* **Interview Questions**:
  + What are the key compiler options in tsconfig.json for React?
  + How do you handle third-party libraries without TypeScript definitions?
  + What's the difference between .ts and .tsx files?

**1.5 Tooling and Ecosystem**

* ESLint and Prettier Configuration
* Husky for Git Hooks
* Testing Setup (Jest, React Testing Library)
* Bundle Analyzers
* CI/CD Pipeline Considerations
* **Interview Questions**:
  + How do you set up ESLint for React TypeScript projects?
  + What tools do you use for code formatting and why?
  + How do you handle type checking in CI pipelines?

**Module 2: TypeScript Fundamentals for React Developers**

**2.1 Basic Types and Type Annotations**

* Primitive Types (string, number, boolean)
* Array and Tuple Types
* any, unknown, never, void Types
* Type Inference vs Explicit Annotations
* Literal Types and const assertions
* **Interview Questions**:
  + What's the difference between any and unknown types?
  + When would you use tuple types over arrays?
  + How does TypeScript type inference work with React?

**2.2 Interfaces vs Type Aliases**

* Interface Declaration and Extension
* Type Alias with Unions and Intersections
* Declaration Merging with Interfaces
* Mapped Types and Conditional Types
* Choosing Between Interface and Type
* **Interview Questions**:
  + When would you choose interface over type and vice versa?
  + What is declaration merging and how does it work?
  + Can you explain mapped types with practical examples?

**2.3 Generics in React Context**

* Generic Functions and Interfaces
* Generic Constraints with extends
* Generic Components in React
* Generic Custom Hooks
* Advanced Generic Patterns
* **Interview Questions**:
  + How do you create generic React components?
  + What are generic constraints and when are they useful?
  + Can you show an example of a generic custom hook?

**2.4 Utility Types for React**

* Partial, Required, Readonly
* Pick, Omit, Extract, Exclude
* Record, NonNullable, Parameters
* ReturnType for Hooks
* Creating Custom Utility Types
* **Interview Questions**:
  + How do you use Pick and Omit in React component props?
  + What's the practical use of ReturnType in React?
  + Can you create a utility type that makes all properties optional except ID?

**2.5 Type Guards and Narrowing**

* typeof and instanceof Guards
* User-Defined Type Guards
* Discriminated Unions
* in operator Narrowing
* Custom Type Predicates
* **Interview Questions**:
  + What are type guards and why are they important?
  + How do discriminated unions help with state management?
  + Can you write a type guard for checking if an object is a User?

**Module 3: React Components with TypeScript**

**3.1 Functional Components Typing**

* React.FC Pros and Cons
* Component Props Interfaces
* Default Props Handling
* children Prop Typing
* Component Return Types
* **Interview Questions**:
  + What are the arguments for and against using React.FC?
  + How do you type the children prop properly?
  + What's the difference between JSX.Element and React.ReactNode?

**3.2 Props Patterns and Best Practices**

* Optional vs Required Props
* Destructuring Props with Types
* Props Spread Patterns
* Component Composition Types
* Prop Drilling Solutions
* **Interview Questions**:
  + How do you handle optional props with default values?
  + What's the best way to type component composition?
  + How do you avoid prop drilling with TypeScript?

**3.3 Component Lifecycle with TypeScript**

* useEffect Dependency Array Typing
* Cleanup Functions in useEffect
* useLayoutEffect for DOM Operations
* Component Mount/Unmount Patterns
* Error Boundaries with TypeScript
* **Interview Questions**:
  + How do you properly type useEffect dependencies?
  + What's the TypeScript approach to error boundaries?
  + How do you handle async operations in useEffect with TypeScript?

**3.4 Conditional Rendering and Type Safety**

* Type Narrowing in Render Logic
* Guarded Conditional Rendering
* Render Props Pattern with Types
* Higher-Order Components Typing
* Conditional Props Patterns
* **Interview Questions**:
  + How do you ensure type safety in conditional rendering?
  + What are the TypeScript considerations for HOCs?
  + How do you type render props components?

**3.5 Component Patterns and Composition**

* Compound Components Pattern
* Container/Presenter Pattern
* Render Props with Generics
* Controlled vs Uncontrolled Components
* Forward Ref Components
* **Interview Questions**:
  + How do you type compound components in TypeScript?
  + What's the difference between controlled and uncontrolled components with TypeScript?
  + How do you use forwardRef with proper typing?

**Module 4: State Management with TypeScript**

**4.1 useState Hook with TypeScript**

* Primitive State Typing
* Object State with Interfaces
* Complex State Structures
* Functional Updates with Previous State
* State Initialization Patterns
* **Interview Questions**:
  + How do you type complex state objects with useState?
  + What are the performance implications of state typing?
  + How do you handle state that might be null or undefined?

**4.2 useReducer with Strict Typing**

* Action Discriminated Unions
* Reducer State Interface Design
* Action Creator Functions
* Immer for Immutable Updates
* Reducer Composition Patterns
* **Interview Questions**:
  + How do you create type-safe actions with discriminated unions?
  + What are the benefits of using useReducer over useState in TypeScript?
  + How do you handle async actions in useReducer?

**4.3 Custom Hooks with TypeScript**

* Hook Return Type Inference
* Parameter Typing for Custom Hooks
* Generic Custom Hooks
* Hook Dependency Arrays
* Testing Custom Hooks
* **Interview Questions**:
  + How do you create a generic custom hook?
  + What are the best practices for typing custom hook parameters?
  + How do you test custom hooks with TypeScript?

**4.4 Context API with Type Safety**

* Context Value Typing
* Context Provider Props
* useContext Hook with Type Guards
* Context Default Values
* Multiple Contexts Pattern
* **Interview Questions**:
  + How do you create a type-safe Context API?
  + What's the proper way to handle default values in context?
  + How do you avoid unnecessary re-renders with typed context?

**4.5 Advanced State Patterns**

* State Machines with XState
* Atomic State with Jotai
* Zustand for Global State
* URL State Management
* Persistent State Patterns
* **Interview Questions**:
  + How do you integrate state machines with TypeScript?
  + What are the TypeScript benefits of using Zustand over Redux?
  + How do you manage URL state with type safety?

**Module 5: Event Handling and Forms**

**5.1 Synthetic Event Types**

* Form Events (onSubmit, onChange)
* Mouse and Keyboard Events
* Focus and Blur Events
* Drag and Drop Events
* Custom Event Typing
* **Interview Questions**:
  + What are the common React event types in TypeScript?
  + How do you properly type form submission handlers?
  + What's the difference between ChangeEvent and InputEvent?

**5.2 Controlled Components**

* Input Value Typing
* Textarea and Select Handling
* Checkbox and Radio Groups
* File Input Typing
* Dynamic Form Fields
* **Interview Questions**:
  + How do you type controlled form components?
  + What's the best approach for dynamic form fields with TypeScript?
  + How do you handle file uploads with proper typing?

**5.3 Form Validation with TypeScript**

* Schema Validation with Zod/Yup
* Runtime Type Checking
* Error State Management
* Async Validation Patterns
* Form Submission Typing
* **Interview Questions**:
  + How do you integrate Zod with React forms?
  + What's the benefit of runtime validation with TypeScript?
  + How do you type form errors and validation states?

**5.4 Form Libraries Integration**

* React Hook Form with TypeScript
* Formik Typing Patterns
* Final Form Type Safety
* Custom Form Hook Creation
* Performance Optimization
* **Interview Questions**:
  + How do you use React Hook Form with TypeScript?
  + What are the TypeScript advantages of using form libraries?
  + How do you create type-safe custom form hooks?

**5.5 Advanced Form Patterns**

* Multi-step Forms
* Dynamic Field Arrays
* Form State Persistence
* Optimistic Updates
* Form Analytics and Tracking
* **Interview Questions**:
  + How do you type multi-step form state?
  + What's the best way to handle dynamic fields with TypeScript?
  + How do you implement optimistic updates with type safety?

**Module 6: Routing and Navigation**

**6.1 React Router with TypeScript**

* Route Configuration Typing
* URL Parameters Type Safety
* Query String Parsing
* Route Component Props
* Navigation State Management
* **Interview Questions**:
  + How do you type route parameters in React Router?
  + What's the best way to handle type-safe navigation?
  + How do you create a type-safe route configuration?

**6.2 Protected Routes and Authentication**

* Route Guards with Type Safety
* Authentication Context Typing
* Role-based Route Access
* Redirect Logic with Types
* Route Loading States
* **Interview Questions**:
  + How do you implement type-safe route protection?
  + What's the best pattern for authentication with TypeScript?
  + How do you handle role-based access with types?

**6.3 Dynamic Routing and Code Splitting**

* Lazy Loading Component Typing
* Dynamic Import Patterns
* Route-based Code Splitting
* Loading and Error Boundaries
* Prefetching Strategies
* **Interview Questions**:
  + How do you type lazy-loaded components?
  + What's the TypeScript approach to dynamic imports?
  + How do you handle loading states with type safety?

**6.4 URL State Management**

* Search Params Typing
* Hash Router Patterns
* History State Management
* URL Serialization/Deserialization
* Sync State with URL
* **Interview Questions**:
  + How do you manage type-safe URL state?
  + What's the best way to serialize complex state to URL?
  + How do you handle URL search params with TypeScript?

**6.5 Navigation Patterns**

* Programmatic Navigation
* Navigation State Persistence
* Breadcrumb Typing
* Active Route Detection
* Mobile Navigation Patterns
* **Interview Questions**:
  + How do you type programmatic navigation?
  + What's the best way to handle navigation state?
  + How do you create type-safe breadcrumbs?

**Module 7: Data Fetching and API Integration**

**7.1 HTTP Clients with TypeScript**

* Axios Response Typing
* Fetch API with Generics
* Request/Response Interfaces
* Error Handling Patterns
* Interceptor Typing
* **Interview Questions**:
  + How do you create type-safe API clients?
  + What's the difference between Axios and Fetch with TypeScript?
  + How do you handle API errors with proper typing?

**7.2 React Query/TanStack Query**

* Query Hook Typing
* Mutation Type Safety
* Query Key Management
* Infinite Query Patterns
* Optimistic Updates Typing
* **Interview Questions**:
  + How do you type React Query hooks?
  + What's the benefit of type-safe query keys?
  + How do you handle optimistic updates with TypeScript?

**7.3 SWR and Data Fetching Patterns**

* SWR Hook Typing
* Data Transformation Types
* Revalidation Strategies
* Global Fetch Configuration
* Cache Management
* **Interview Questions**:
  + How do you use SWR with TypeScript?
  + What are the type safety benefits of data fetching libraries?
  + How do you handle data transformation with types?

**7.4 GraphQL with TypeScript**

* Apollo Client Typing
* GraphQL Code Generator
* Query/Mutation Hooks
* Fragment Type Safety
* Cache Management Types
* **Interview Questions**:
  + How do you set up GraphQL Code Generator?
  + What's the benefit of typed GraphQL queries?
  + How do you handle GraphQL errors with TypeScript?

**7.5 WebSocket and Real-time Data**

* WebSocket Event Typing
* Real-time Hook Creation
* Connection State Management
* Message Serialization
* Error Recovery Patterns
* **Interview Questions**:
  + How do you type WebSocket connections?
  + What's the best pattern for real-time data with TypeScript?
  + How do you handle connection states with type safety?

**Module 8: Advanced Patterns and Performance**

**8.1 Render Optimization with TypeScript**

* React.memo with Props Typing
* useCallback Dependency Arrays
* useMemo Return Type Inference
* Component Re-render Prevention
* Profiling and Optimization
* **Interview Questions**:
  + How do you use React.memo with TypeScript?
  + What are the TypeScript considerations for useCallback?
  + How do you profile performance in TypeScript React apps?

**8.2 Design Patterns in TypeScript**

* Factory Pattern for Components
* Strategy Pattern for Algorithms
* Observer Pattern for State
* Decorator Pattern for Enhancement
* Compound Component Patterns
* **Interview Questions**:
  + How do you implement the factory pattern in React with TypeScript?
  + What design patterns are most useful with TypeScript?
  + How do compound components benefit from TypeScript?

**8.3 Type-Safe Styling Solutions**

* Styled Components with TypeScript
* CSS Modules Typing
* Tailwind CSS with Type Safety
* Theme Typing Patterns
* Responsive Design Types
* **Interview Questions**:
  + How do you type styled-components?
  + What's the best way to handle themes with TypeScript?
  + How do you create type-safe responsive designs?

**8.4 Internationalization (i18n)**

* i18n Library Typing
* Translation Key Safety
* Locale Switching Patterns
* Pluralization and Formatting
* RTL Layout Support
* **Interview Questions**:
  + How do you implement type-safe internationalization?
  + What's the benefit of typed translation keys?
  + How do you handle dynamic translations with TypeScript?

**8.5 Accessibility (a11y) with TypeScript**

* ARIA Attributes Typing
* Keyboard Navigation
* Focus Management
* Screen Reader Support
* Accessibility Testing
* **Interview Questions**:
  + How does TypeScript help with accessibility?
  + What are the best practices for a11y in TypeScript React?
  + How do you test accessibility with TypeScript?

**Module 9: Testing Strategies**

**9.1 Unit Testing Components**

* React Testing Library with TypeScript
* Component Render Testing
* User Event Simulation
* Async Component Testing
* Custom Render Functions
* **Interview Questions**:
  + How do you set up React Testing Library with TypeScript?
  + What are the benefits of typed testing?
  + How do you test async components with TypeScript?

**9.2 Hook Testing Patterns**

* renderHook Utility Usage
* Custom Hook Test Utilities
* Async Hook Testing
* Hook State Changes
* Hook Error Boundaries
* **Interview Questions**:
  + How do you test custom hooks with TypeScript?
  + What's the best pattern for async hook testing?
  + How do you mock hook dependencies?

**9.3 Integration Testing**

* User Flow Testing
* API Mocking with Type Safety
* Router Testing in Isolation
* Context Provider Testing
* End-to-End Type Safety
* **Interview Questions**:
  + How do you create type-safe mocks?
  + What's the approach to testing routed components?
  + How do you test context providers with TypeScript?

**9.4 Test Utilities and Patterns**

* Custom Test Utilities
* Factory Functions for Test Data
* Type-Safe Test Helpers
* Snapshot Testing with Types
* Performance Testing
* **Interview Questions**:
  + How do you create type-safe test utilities?
  + What's the benefit of factory functions in testing?
  + How do you handle snapshot testing with TypeScript?

**9.5 E2E Testing with TypeScript**

* Cypress with TypeScript
* Playwright Typing Patterns
* Test Data Management
* Visual Regression Testing
* Accessibility Testing Automation
* **Interview Questions**:
  + How do you set up Cypress with TypeScript?
  + What are the benefits of typed E2E tests?
  + How do you manage test data with TypeScript?

**Module 10: Build, Deploy, and Maintain**

**10.1 Build Optimization**

* Tree Shaking with TypeScript
* Code Splitting Strategies
* Bundle Size Analysis
* Asset Optimization
* Environment-specific Builds
* **Interview Questions**:
  + How do you optimize TypeScript builds for production?
  + What tools do you use for bundle analysis?
  + How do you handle environment-specific types?

**10.2 Deployment Strategies**

* CI/CD Pipeline with Type Checking
* Environment Variable Typing
* Deployment Configuration
* Rollback Strategies
* Monitoring and Analytics
* **Interview Questions**:
  + How do you integrate type checking in CI/CD?
  + What's the best way to handle environment configurations?
  + How do you monitor TypeScript applications in production?

**10.3 Maintenance and Refactoring**

* Type-Safe Refactoring
* Deprecation Strategies
* Version Upgrade Patterns
* Code Migration Tools
* Performance Monitoring
* **Interview Questions**:
  + How does TypeScript help with refactoring?
  + What's your strategy for deprecating types?
  + How do you handle breaking changes in type definitions?

**10.4 Documentation and Collaboration**

* Type Documentation with TSDoc
* Storybook with TypeScript
* Component Documentation
* API Type Documentation
* Team Collaboration Patterns
* **Interview Questions**:
  + How do you document TypeScript types?
  + What's the benefit of Storybook with TypeScript?
  + How do you ensure type consistency across teams?

**10.5 Advanced Tooling**

* Custom ESLint Rules
* TypeScript Compiler API
* AST Manipulation
* Code Generation Tools
* Custom Language Features
* **Interview Questions**:
  + How do you create custom ESLint rules for React TypeScript?
  + What are some advanced TypeScript compiler features?
  + How do you use AST manipulation in your workflow?

**Bonus Module: Interview Preparation**

**Common React TypeScript Interview Questions**

1. **Fundamentals**:
   * What's the difference between interface and type in TypeScript?
   * How do you handle default props in functional components?
   * What are generics and when would you use them in React?
2. **State Management**:
   * How do you type complex state with useReducer?
   * What's the benefit of discriminated unions in Redux actions?
   * How do you create type-safe context?
3. **Performance**:
   * How does TypeScript help with performance optimization?
   * What are the typing considerations for React.memo?
   * How do you type dependency arrays in hooks?
4. **Testing**:
   * How do you mock API responses with proper types?
   * What's the approach to testing generic components?
   * How do you handle type checking in tests?
5. **Advanced Patterns**:
   * How do you implement the render props pattern with TypeScript?
   * What's the type-safe approach to higher-order components?
   * How do you handle conditional rendering with type narrowing?

**Problem-Solving Exercises**

* Type a complex form with dynamic fields
* Create a type-safe data table component
* Implement a generic API hook
* Build a type-safe routing system
* Create a theme system with full type safety

**Code Review Scenarios**

* Identifying type safety issues
* Performance problems in typed components
* Refactoring JavaScript to TypeScript
* Improving type definitions
* Architecture decisions with TypeScript

|  |  |  |  |
| --- | --- | --- | --- |
| Typescript  |  | | --- | | Subtitle Text Here To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. | | *“Find even more easy-to-use tools on the Insert tab, such as to add a hyperlink or insert a comment”* | | To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. | |

# Selenium

|  |
| --- |
| Subtitle Text Here To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. |
| *“Find even more easy-to-use tools on the Insert tab, such as to add a hyperlink or insert a comment”* |
| To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. |

# Playwright

|  |
| --- |
| Subtitle Text Here To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. |
| *“Find even more easy-to-use tools on the Insert tab, such as to add a hyperlink or insert a comment”* |
| To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. |

# Python

|  |
| --- |
| Subtitle Text Here To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. |
| *“Find even more easy-to-use tools on the Insert tab, such as to add a hyperlink or insert a comment”* |
| To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. |

# Java

|  |
| --- |
| Subtitle Text Here To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. |
| *“Find even more easy-to-use tools on the Insert tab, such as to add a hyperlink or insert a comment”* |
| To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. |

# React

|  |
| --- |
| Subtitle Text Here To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. |
| *“Find even more easy-to-use tools on the Insert tab, such as to add a hyperlink or insert a comment”* |
| To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.  Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need. |