TypeScript Example Presentation

1. Initial JavaScript site
   1. Create new Project | ASP.NET Web Application (.NET Framework)
      1. Not specific to .Net can be .Net core and does not even have to be Visual Studio or .Net. It’s a separate command line compiler and can be integrated into any web development build process. I’m sticking with VS because that’s the environment we are familiar with.
      2. Create content, content/cdn and content/scripts
      3. Copy files from snippets/cdn to project content/cdn
      4. Move favicon.ico to top Project folder.
      5. Create content/index.html from snippets
      6. Set Start Action | Specific Page in Project Properties to “index.html”
      7. Run
      8. Leave browser up and shutdown server
   2. Create initial JavaScript functionality
      1. Create scripts\greeting.js and fill in from snippets
         1. Point out the EcmaScript 6 back quotes
      2. Add script tags to Index.html
      3. Run
      4. Compare the first tab and the second in the browser
      5. Open dev tools (F12)
      6. Run in InternetExplorer Go to dev mode and show errors in the console.
      7. Shutdown server
2. Migrate to TypeScript
   1. Rename greetings.js to greeting.ts and Rebuild the Solution.
      1. Show all files in Solution to show generated greeting.ts
      2. Show them greetings.js. Point out source map comment.
      3. Run the app.
      4. Show dev mode and switch to Sources
      5. Refresh IE
   2. TypeScript Options
      1. Close Solution and Reopen
      2. Show TypeScript configuration page of the Project
         1. ECMAScript version = Target version of JavaScript, Allow implicit any types, ComplileOnSave,NoEmitOnErrorError
      3. Easy configuration for common options.
      4. Only available in VisualStudio with MSBuild.
   3. tsconfig.json
      1. Many examples online use tsconfig.
      2. VS2017 will automatically switch to tsconfig if available.
      3. Each folder can be configured separately and tsconfig settings can be inherited from parent.
      4. Create tsconfig.json
      5. Default does not allow implicit “any”, does not include compileOnSave, all files go to same folder as TS files.
      6. Show project properties page is disabled.
3. Refactor 1 – Types
   1. Create “src” folder and move greeting.ts there.
   2. Put in include “src”, compileOnSave and outDir = “./scripts”
   3. Run
   4. Show dev mode and sources
   5. Change greeting.ts : `Hello $(person}!’ to ‘Hello ${user}!’ – show the error
   6. Change greeting.js : “Hello “ + person + “!” to “Hello “ + user + “!” – show NO error
   7. Change parameter person to a “: number”
   8. Show that index.html scripts do not give you good intellisense.
   9. Refactor index.html script into **M**ain.ts – DO NOT ADD script tag for main.js
   10. Show error in main.ts
   11. Change parameter to a string show that error goes away in main.ts
   12. Change value in main.ts to 42 and show new error
   13. Change parameter type to “string | number”
   14. Show error goes away again.
   15. Change 42 back to the user variable.
   16. Hard Refresh the browser
   17. Show that the name was not updated it still shows “You”
       1. May need to do a hard refresh
   18. Need to include script “main.js” in the page.
       1. This will become cumbersome if we have to do this for each script we write.
4. Refactor 2 – simple bundling and JQuery
   1. Shut down app
   2. Replace outDir with outFile=“./scripts/application.js”
   3. Add blank line and comments to the top of both greeting.ts and main.ts and show application.js
   4. Update index.html to load application.js
   5. Look at application.js and application.js.map
      1. Comments but no blank lines
      2. removeComments in tsconfig.json
   6. Run app
   7. Show Dev mode sources – one script in scripts but src has ts files.
   8. Shut down
   9. Convert main.ts to use JQuery, $
   10. Notice TypeScript doesn’t know what $ is.
   11. We will use npm to install @types
   12. Open command window in the solution folder
   13. Create package.json
       1. “npm init”
       2. Create Solution Folder “config” and add “package.json”
   14. From Command line run “npm install @types/jquery –save-dev”
       1. There are many Type packages for common libraries.
       2. Old way is “typings” where types are stored in scripts/types - VS does this way
   15. Update tsconfig with lib to fix Iterable error
   16. Now it knows what $ is and it knows that innerText is not correct.
   17. Add the jquery script tag to index.html, run, show and shutdown
5. Refactor 3 – Using interfaces and function signatures
   1. Create src/models folder
   2. Create Model/Person interface
   3. Change parameter in greeting to be type Person and update the code
      1. Now there is an error in main.ts
   4. Change var user to be a json object.
   5. NOTICE: no explicit declaration of implementing Person.
   6. Look at application.js.
      1. Notice that Person interface does not show up. It’s important in TypeScript for type checking but has no real value in the JavaScript
   7. Run the app.
   8. Show Dev Mode Source Person.ts is there even though it’s not in application.js
   9. What if we want to allow greeting to be used with specific string name parameters or a Person object
      1. Overloading
      2. Copy overload greeting code from snippet
      3. Last signature is not shown. It is only the implementation
   10. Refresh the page
       1. Missing lastName.
   11. Add restOfName parameter to implementation signature and code.
   12. Show application.js and how restOfName is implemented
       1. Notice only the implementation shows up
       2. Go back to main and bring up list of signatures
   13. Change parameter to greeting to “First Only”.
       1. Error because lastName is missing
       2. Make lastName? Optional
   14. Refresh browser and there’s a space after FirstName
       1. Add code to check length of restOfName
   15. Put back user parameter and refresh page.
6. Refactor 4 - Classes
   1. Shutdown
   2. Add the PersonObj class
   3. Change user to be assigned to a new PersonObj
      1. Show code insight for the constructor
   4. Add fullName() method to Person Interface
      1. PersonObj now has an error
      2. Add fullName() implementation to PersonObj
      3. “this” in constructor makes sense but why does typescript require “this” in the class methods.
      4. JavaScript has global variables. You could want to refer to a global variable. “this” tells TypeScript you want to refer to the class fields.
   5. Fat Arrow – make sure this is this.
      1. Functions can be passed around. This is not always the object they started out on.
      2. Fat Arrow makes sure this is this
      3. Show application.js
      4. An example of the incredible power of JavaScript. “\_this” is defined at definition inside the parent function but can be used within its scope to always refer to the original this at definition.
   6. Add middleName property
      1. Rebuild – PersonObj does not include middleName;
      2. Make middleName? optional
      3. Rebuild – no errors
   7. Create new class PersonWithMiddleNameObj()
      1. Copy code but point out super() in constructor and implementation of fullName()
   8. Change user in main.ts to be a PersonWithMiddleNameObj
      1. “.Net”, “Development”, “Team”
   9. Run the app
   10. Missing middleName
   11. Change greeting() to use fullName() and Refresh
   12. Take a look application.js to see how classes and subclasses are implemented.
   13. You don’t lose the adhoc power of JavaScript.
   14. Implement Person interface on use right in main.ts
       1. Don’t even have to declare it as a Person
       2. Change fullName() to fullNameX() and save
       3. You see there is now an error in the call to greeting
       4. Put fullName() back and save
   15. Refresh browser
   16. Shutdown
7. Unit tests
   1. Many JavaScript UnitTest libraries.
      1. Mocha/Chai, Jasmine, QUnit are popular
      2. If not already done, “npm install mocha chai ts-node -g”
      3. I’m going to use Mocha because it’s the first one I got working ☺
   2. “npm install @types/mocha @types/chai”
   3. Create src/tests folder
   4. Create src/GreetingTest.ts from Snippet
   5. Create content/tests.html from Snippet
   6. Need to split out code that starts the app so it doesn’t start during the tests
   7. Run and switch to “tests.html”
   8. Add more tests for greeting()
      1. Talk about what the tests are testing
   9. Chutzpah may be working. If it is show it to them.
8. Refactor 5 – Renaming fields, interfaces, classes and methods
   1. Want to make it clearer what’s an interface and drop the Obj from the classes.
   2. Rename Person.ts to IPerson.ts and PersonObj.ts to Person.ts
   3. Rename the actual interface Person to IPerson with Ctrl+R,Ctrl+R
   4. Rename PersonObj and PersonWithMiddleNameObj to drop the Obj
   5. Refresh test.html
   6. Show “Find All References” for IPerson.lastName.
   7. Rename “Person.lastName” to “familyName”
   8. Show that the user.lastName is not changed because TypeScript does not know it’s a Person.
   9. TypeScript does know that user can no longer be used as the parameter to greeting()
   10. Add “: IPerson” to declare it’s type,
       1. The error in main is gone but now it tells us there is no lastName property.
   11. change lastName to familyName for user so the error goes away
   12. Refactor Rename famlyName back to lastName in the main.ts in the user : IPerson variable
   13. Show that the Person interface now has it’s lastName property again.
   14. Refresh test.html
   15. Refactor Rename fullName() to FullName()
   16. Refresh test.html
   17. It didn’t catch the one in greeting because it doesn’t really know it’s an IPerson.
   18. Type cast parm “ (parm as IPerson)”
   19. Error shows up that fullName() Is not found.
   20. Change it to FullName() error goes away.
   21. You could Refactor Rename FullName() to something like CompleteName() from greeting.ts, and it would rename in IPerson and the Person classes.
   22. These types of renames in JavaScript are a scary prospect because you can never be sure if you caught all the correct places until you run EVERY line of code.
9. Refactor 6 – Interface Inheritance
   1. Shuttdown
   2. Create interface src/models/ISecurityIdentity.ts from snippets
   3. Create class src/models/SecurityIdentity.ts to
   4. We created an implementation class of the interface that extends PersonWithMiddleName
   5. We can add some “security” to main.ts by checking for authenticate()
   6. Run the app
   7. Change the login name to undefined
   8. Refresh the page.
   9. Take a look at application.js
   10. Nothing voodoo. Very similar to how extending classes would be handled in JavaScript without the learning curve.
   11. In TypeScript the syntax is familiar to anyone that has used Java or C# but it’s converted to JavaScript that will run on any browser.
   12. Add a GreetingTest for SecurityIdentity
       1. just copy the PersonWithMiddleName test and change the object
10. Refactor 7 – Namespaces
    1. Shut down
    2. Add namespace Identity to IPerson
    3. Export IPerson
    4. Add namespace in Person.ts but only export class PersonWithMiddleNameObj
    5. Fix the references in SecurityIdentity, greeting and GreetingTest
       1. Going to have to comment out the Person test because it’s no longer exposed.
    6. Add namespace Lib to greeting.ts
       1. Export all greeting() function signatures and fix references
    7. Fix the references in main, GreetingTest
    8. Look at greeting and Identity in application.js
    9. Rebuild
       1. Person is no longer visible because it’s not exported.
       2. Add export on Person and rebuild
    10. Run the app and refresh the test page
11. Refactor 8 - Modules
    1. Update tsconfig.json
       1. “module”:”amd” – It’s the typical module system for browsers
       2. CommonJS is typically used when running in NodeJS
    2. Replace src contents with Snippets/Refactor 8-bundled
       1. Remove namespaces but leave the exports
       2. Add imports for referenced objects
    3. Show Person.ts
       1. Import IPerson
       2. Person is not exported but PersonWithModdleName is
       3. set default import for single class/interface/function
    4. Show SecurityIdentity
       1. Import of PersonWithMiddleName because it’s the default
       2. Import of ISecurityIdentity
    5. Show ISecurityIdentity
       1. Interfaces take special treatment for setting defaults.
    6. Update index.html
       1. Add requires script tag
       2. Remove script tag mocha.js, chai.js
       3. Wrap call to main.document\_ready() to with require([‘Main’], …
    7. Update tests.html
       1. Add requirejs tag
       2. Wrap mocha.run with require([‘tests/GreetingTest’], …
12. Refactor 9 – Framework
    1. Create Models/IPage.ts from snippets
    2. Create Models/IApplication.ts from snippet
    3. Add “export default IPerson” to “IPerson”
    4. Create folder src/pages
    5. Create pages/MainPage.ts
       1. Show MainPage error and the button to implement the interface.
       2. Copy the switchingTo() implementation from snippet
    6. Create tests/MainPageTests and copy from snippets
    7. Add tests/MainPageTests to require in tests.html
    8. Replace main.ts with all WebApplication code
    9. Remove JQuery from index.html and tests.html
    10. Rebuild
    11. Run the app
    12. Refresh the Tests.html page
    13. Error because of empty MiddleName.
        1. Fix it in PersonWithMiddleName.FullName() by checking for an empty middleName.