Setup Example

This example application solely focuses on adding WebDriver testing to an already existing project. To have a project to test in the following two sections, we will set up an extremely minimal Tauri application for use in our testing. We will not use the Tauri CLI, any frontend dependencies or build steps, and not be bundling the application afterward. This is to showcase exactly a minimal suite to show off adding WebDriver testing to an existing application.

If you just want to see the finished example project that utilizes what will be shown in this example guide, then you can see https://github.com/chippers/hello_tauri.

Initializing a Cargo Project

We want to create a new binary Cargo project to house this example application. We can easily do this from the command line with `cargo new hello-tauri-webdriver --bin`, which will scaffold a minimal binary Cargo project for us. This directory will serve as the working directory for the rest of this guide, so make sure the commands you run are inside this new `hello-tauri-webdriver/` directory.

Creating a Minimal Frontend

We will create a minimal HTML file to act as our example application's front end. We will also be using a few things from this frontend later during our WebDriver tests.

First, let's create our Tauri `distDir` that we know we will need once building the Tauri portion of the application. `mkdir dist` should create a new directory called `dist/` in which we will be placing the following `index.html` file.

`dist/index.html`:

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Hello Tauri!</title>
    <style>
      body {
        /* Add a nice colorscheme */
        background-color: #222831;
        color: #ececec;

        /* Make the body the exact size of the window */
        margin: 0;
        height: 100vh;
        width: 100vw;

        /* Vertically and horizontally center children of the body tag */
        display: flex;
        justify-content: center;
        align-items: center;
      }
    </style>
  </head>
  <body>
    <h1>Hello, Tauri!</h1>
  </body>
</html>
```

Adding Tauri to the Cargo Project

Next, we will add the necessary items to turn our Cargo project into a Tauri project. First, is adding the dependencies to the Cargo Manifest (`Cargo.toml`) so that Cargo knows to pull in our dependencies while building.

`Cargo.toml`:

```
[package]
name = "hello-tauri-webdriver"
version = "0.1.0"
edition = "2021"
rust-version = "1.56"

# Needed to set up some things for Tauri at build time
[build-dependencies]
tauri-build = "1"

# The actual Tauri dependency, along with `custom-protocol` to serve the pages.
[dependencies]
tauri = { version = "1", features = ["custom-protocol"] }

# Make --release build a binary that is small (opt-level = "s") and fast (lto = true).
# This is completely optional, but shows that testing the application as close to the
# typical release settings is possible. Note: this will slow down compilation.
[profile.release]
```

```
incremental = false
codegen-units = 1
panic = "abort"
opt-level = "s"
lto = true
```

We added a `[build-dependency]` as you may have noticed. To use the build dependency, we must use it from a build script. We will create one now at `build.rs`.

`build.rs`:

```
fn main() {
    // Only watch the `dist/` directory for recompiling, preventing unnecessary
    // changes when we change files in other project subdirectories.
    println!("cargo:rerun-if-changed=dist");

    // Run the Tauri build-time helpers
    tauri_build::build()
}
```

Our Cargo Project now knows how to pull in and build our Tauri dependencies with all that setup. Let's finish making this minimal example a Tauri application by setting up Tauri in the actual project code. We will be editing the `src/main.rs` file to add this Tauri functionality.

`src/main.rs`:

```
fn main() {
    tauri::Builder::default()
        .run(tauri::generate_context!())
        .expect("unable to run Tauri application");
}
```

Pretty simple, right?

Tauri Configuration

We are going to need 2 things to successfully build the application. First, we need an icon file. You can use any PNG for this next part and copy it into `icon.png`. Typically, this will be provided as part of the scaffolding when you use the Tauri CLI to create a project. To get the default Tauri icon, we can download the icon used by the Hello Tauri example repository with the command `curl -L "https://github.com/chippers/hello_tauri/raw/main/icon.png" --output icon.png`.

We will need a `tauri.conf.json` to set some important configuration values for Tauri. Again, this would typically come from the `tauri init` scaffolding command, but we will be creating our own minimal config here.

`tauri.conf.json`:

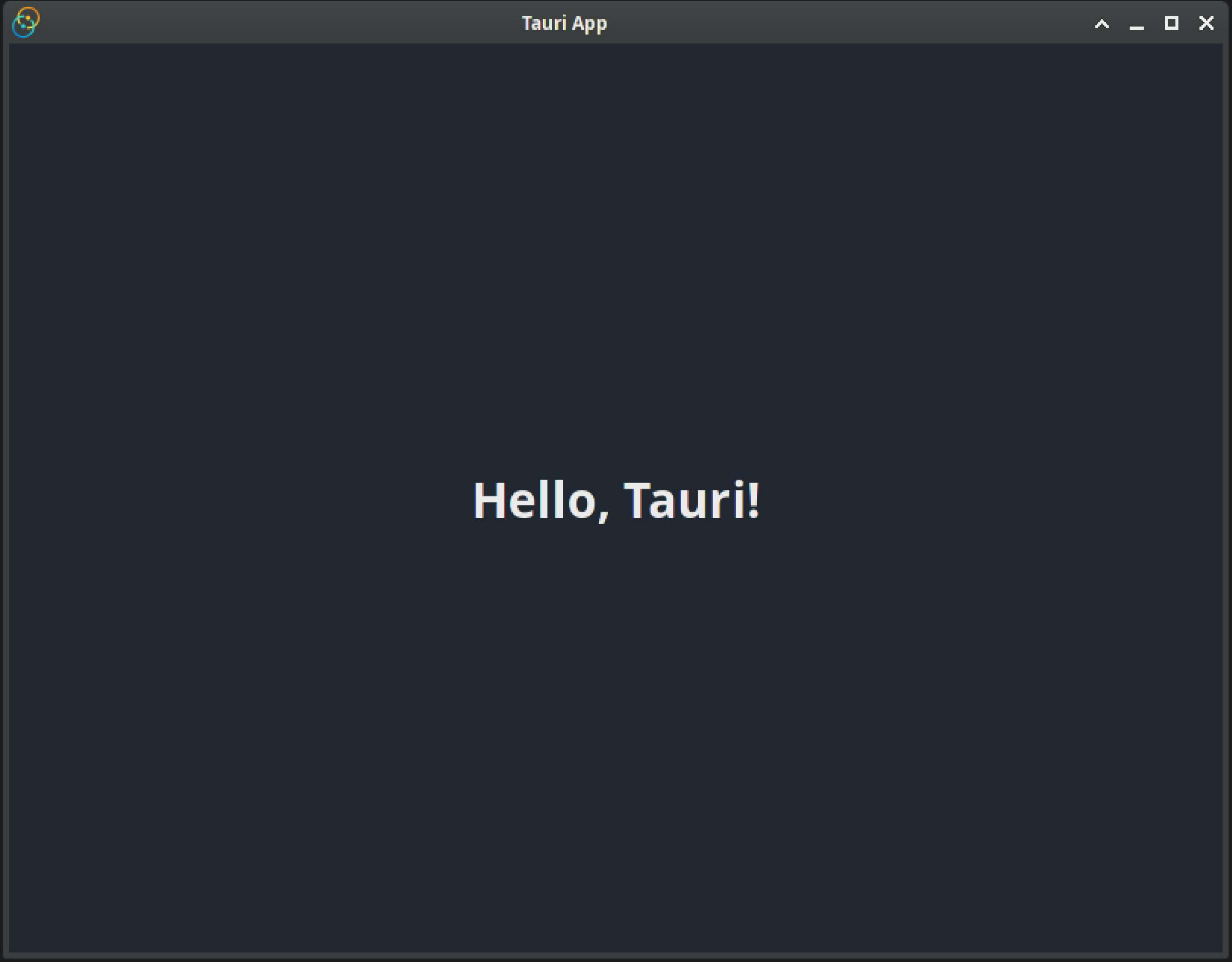
```json
{
  "build": {
    "distDir": "dist"
  },
  "tauri": {
    "bundle": {
      "identifier": "studio.tauri.hello_tauri_webdriver",
      "icon": ["icon.png"]
    },
    "allowlist": {
      "all": false
    },
    "windows": [
      {
        "width": 800,
        "height": 600,
        "resizable": true,
        "fullscreen": false
      }
    ]
  }
}
```

I'll go over some of these. You can see the `dist/` directory we created earlier specified as the `distDir` property. We set a bundle identifier so that the built application has a unique id and set the `icon.png` as the only icon. We aren't using any Tauri APIs or features, so we disable them in `allowlist` by setting `"all": false`. The window values just set a single window to be created with some reasonable default values.

At this point, we have a basic Hello World application that should display a simple greeting when run.

Running the Example Application

To make sure we did it right, let's build this application! We will run this as a `--release` application because we will also run our WebDriver tests with a release profile. Run `cargo run --release`, and after some compiling, we should see the following application pop up.



Note: If you are modifying the application and want to use the Devtools, then run it without `--release` *and "Inspect Element" should be available in the right-click menu.*

We should now be ready to start testing this application with some WebDriver frameworks. This guide will go over both WebdriverIO and Selenium in that order.

Edit this page

Last updated on ***Mar 25, 2023***