PV252 Intro

Welcome!

- Hi! I’m Sam, and I’m new(-ish) here.
  - I have worked for ~10 years as a freelance Android/Web/iOS developer, now I’m an assistant professor for software engineering.
- You’ll also meet Maria (Maya) Švirikóvá.
  - Senior Interaction Designer @ RedHat, will cover UX research, prototyping and testing.
- The course is in English.
  - We can speak Czech if needed, but English is recommended both in the classroom and for the assignments/project.

Course under construction

- This is the first run of this course.
- There will be problems.
- Please communicate issues as soon as possible so that we can fix them.

Anonymous feedback:
https://forms.gle/9JCoRY2Edo21syeVA

What even is "web development"

- You should have basic knowledge of HTML/CSS/JavaScript (this is not an ”introductory” course in that regard).
  - Still, some content may be redundant for you: students come from diverse backgrounds.
- Web development is a vast and evolving topic.
  - https://www.youtube.com/watch?v=aWfYxg-Ypm4
- Challenge the status quo: Feel free to suggest approaches, architectures and technologies that you are interested in.

What even is "web development"

- The goal isn’t to teach you a specific technology or a framework (React, Angular, Vue, Svelte, Next.js, etc.).
  - “Frameworks they come, they go. Saturday through Sunday Monday…” – Eminem or something
- Learn how to think about (web) user interface architecture: recognize what tools and resources you need depending on the scope of your project.
  - We still need to teach something though…

Course evaluation

- Seminar reviews/retrospectives
- A group project (2 students)

Seminar retrospective

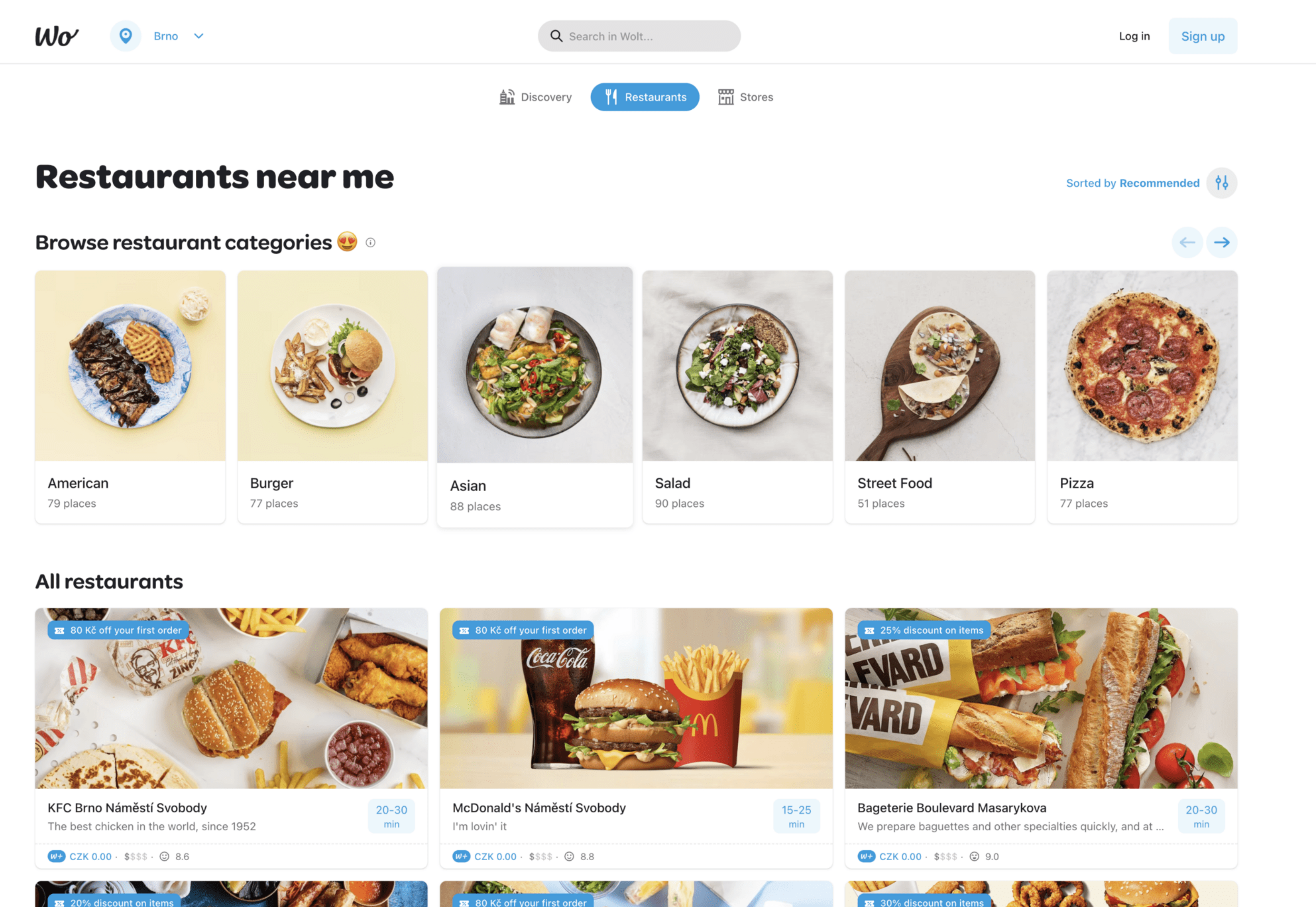
- Each seminar will have a small final assignment that you should complete until the next seminar.* *(well-justified extensions possible)
- Once finished, you'll put a link to your solution in the discussion forum, and another student* should write a review for it. *(reviewers are selected as first-come-first-serve, but it should not be your project partner)
- https://is.muni.cz/auth/discussion/predmetove/fi/podzim2024/PV252/seminar_1/
- In the end, you should have your own solution and a review of someone else's solution.
- Be polite and constructive, but thorough.
    - Virtually no one works alone: communicating clearly about the pros/cons of your design and the design of someone else is often more important than "coding skills".

Project

- Projects are completed by pairs of students.
- Projects are presented at the last lecture of the semester.
- Each team has the same core assignment, but personalized goals will be added during the semester.
- There will be room to consult your progress during the semester, but feel free to ask questions using the discussion forum as well.
- *You are free to use technologies that you prefer (i.e. no need to use X if X was shown at a seminar), but don't overcomplicate things :)*

Project

- Research, design and implement the user interface for ordering meals through a food delivery service.
- Focused on a particular demographic group (more info during the UX lectures).
- You can ignore components like user accounts, payment options, etc. The goal is to focus on product discoverability and order creation.
  - A user should be able to come to your page, somehow discover meals/restaurants (search/list/recommendations/all of the above?) and submit an order for a meal/meals.
- A backend API with fake restaurants will be published this week, but you can add extra items/dimensions if your use case needs it.



Brno

Search in Wolt...

Log in

Sign up

Discovery

Restaurants

Stores

Restaurants near me

Sorted by Recommended

Browse restaurant categories 😍

American
79 places

Burger
77 places

Asian
88 places

Salad
90 places

Street Food
51 places

Pizza
77 places

All restaurants

80 Kč off your first order

KFC Brno Náměstí Svobody
The best chicken in the world, since 1952
20-30 min
CZK 0.00 · $$$$ · 8.6

80 Kč off your first order

McDonald's Náměstí Svobody
I'm lovin' it
15-25 min
CZK 0.00 · $$$$ · 8.8

25% discount on items

Bageterie Boulevard Masarykova
We prepare baguettes and other specialties quickly, and at ...
20-30 min
CZK 0.00 · $$$$ · 9.0

20% discount on items

80 Kč off your first order

30% discount on items

PV252 Seminar 1

(Tooling)

- JavaScript, HTML and CSS are not *"compiled"*, yet most projects use some automated tooling to generate the content that is served to users. Why?
    - (For now, we are assuming the server is simply sending HTML files to users. We'll come be to server-side rendering later)
- **Compatibility:** Not all browsers support all features.
    - Your code may need to be *transpiled* to run on older devices.
- **Performance:** Simplify and minify code.
    - Smaller codebase consumes less network bandwidth and (generally) executes faster.
- **Safety:** JavaScript is very flexible, but provides a lot of room for small mistakes.
    - Type safety, linters and testing help prevent this.

npm

Package Manager (npm)

- https://docs.npmjs.com/cli/v10/commands/npm
- Stores dependencies and package metadata in the `package.json` file.
  - Many other tools support the `package.json` format.
  - https://docs.npmjs.com/cli/v10/configuring-npm/package-json
- `npm` isn't doing anything "special" beyond copying all the dependencies into `node_modules`.
  - You can use `npm` to manage dependencies for other languages.
  - Most packages are just code: you can edit your dependencies *(please don't do this outside of troubleshooting or mischief).*
- `npm` follows semantic versioning.
  - https://docs.npmjs.com/about-semantic-versioning
  - Tricky transitive dependencies: `A` requires `C="1.12"`, but `B` requires `C="2.0"`... Now what?

Package Manager (npm)

- `package-lock.json` stores exact versions of dependencies from last successful install.
  - Storing it in version control ensures reproducibility: you can revert to last known "good dependencies" if new versions don't work.
- You can define custom commands ("scripts") in `package.json`.
  - Call other tools that actually build your project.

```
# Download everything declared in package.json into node_modules
npm install
# Add dependency to package.json
npm install my_fancy_library
# Add development dependency to package.json
npm install --save-dev my_fancy_build_tool
# Find latest (compatible) versions of dependencies and save them
npm update --save
# Execute packages that support it
npx my_fancy_build_tool --fancy-option "fancy"
# Execute a custom script defined in package.json
npm run run-fancy-tool-with-my-options
```

webpack

Bundler (webpack)

- https://webpack.js.org/concepts/
- `webpack` is *bundler*: it resolves imports in your code (both to other code and to assets) and packages them into files that can be deployed on a webserver.
  - Many bundlers exist (`bun`, `vite`, ...).
- `webpack.config.js`: Exports a single object with all the settings and actions that will be applied.
  - It's actually JavaScript: you can run code and have logic in here.
- Resources are processed by loaders: image loader, css loader, html loader, ...
  - Most loaders return the content (e.g. html) or a URL in the final bundle.

```
# Build all entry points
npx webpack --mode=development
npx webpack --mode=production
# Start a live server (requires webpack-dev-server)
npx webpack serve --open --mode=development
```

Bundler (webpack)

- **Entry point**: A separate "bundle", typically one fully independent page of your application.

```
entry: {
    site_a: { import: ["./src/site_a.ts", "uikit"] },
    site_b: { import: ["./src/site_b.ts", "uikit"] },
    main: {
        dependOn: ["site_a", "site_b"],
        import: ["./src/index.ts", "uikit"],
    },
}
```

Bundler (webpack)

- **Rule**: Matches a particular content type (typically by file extension) and specifies which loader (or loaders) should be applied.

```
module: {
  rules: [
    {
      test: /\.(png|svg|jpg|jpeg|gif)$/i,
      type: "asset/resource",
    },
    {
      test: /\.html$/i,
      loader: "html-loader",
    },
    {
      test: /\.less$/i,
      // Compiles Less to CSS
      use: ["style-loader", "css-loader", "less-loader"],
    },
  ],
},
```

Bundler (webpack)

- **Plugin**: Provides extra functionality. Here, `HtmlWebpackPlugin` is used to generate a `.html` file for each entry point using a `.ejs` template.

```
plugins: [
    new HtmlWebpackPlugin({
      title: "PV252 Example project",
      chunks: ["main", "site_a", "site_b"],
      filename: "index.html",
      template: "./src/html/index.template.ejs",
    }),
    new HtmlWebpackPlugin({
      title: "Site A",
      chunks: ["site_a"],
      filename: "site_a.html",
      template: "./src/html/site_a.template.ejs",
    }),
    new HtmlWebpackPlugin({
      title: "Site B",
      chunks: ["site_b"],
      filename: "site_b.html",
      template: "./src/html/site_b.template.ejs",
    }),
],
```

Bundler (webpack)

- **Output**: Describes where and how the bundles and assets should be printed.
  - A good practice is to use content hash in the file name: if the browser has a previous version of your site saved in the cache, it will know based on the file name when the content changes.

```
output: {
    clean: true,
    filename: "[name].[hash].bundle.js",
    path: path.resolve(__dirname, "dist"),
},
```

babel

"Compiler" (babel)

- https://babeljs.io/
- Transforms "modern" JavaScript into JavaScript that is compatible with older browsers.
- Not that critical, but still useful in some cases (see later).
- With `webpack`, we can use `babel-loader`.
- Configuration in `.babelrc` (mostly declaring plugins: optimization, obfuscation, ...).

```
rules: [
  {
    test: /\.(js|ts|jsx|tsx)$/,
    use: {
      loader: "babel-loader",
      options: {
        presets: ["@babel/preset-env"],
      },
    },
    exclude: /node_modules/,
  },
]
```

typescript (tsc)

TypeScript (tsc)

- https://www.typescriptlang.org/
- Typed extension of JavaScript
  - Gradual typing: Not all parts of your application need to be typed
- Comes with its own compiler (`tsc`; `ts-loader` in `webpack`), but is also `babel` support.
  - `webpack` build does not check types by default!
- Configuration in `tsconfig.json`
  - What to include/exclude and how strict the type checking should be.
- Some more advanced stuff will come later, but going too deep into TypeScript itself is beyond the scope of this course :(

```
function sum(a: number, b: number): number {
  return a + b;
}
```

LESS

(or SASS, etc.)

Rich CSS (LESS, SASS)

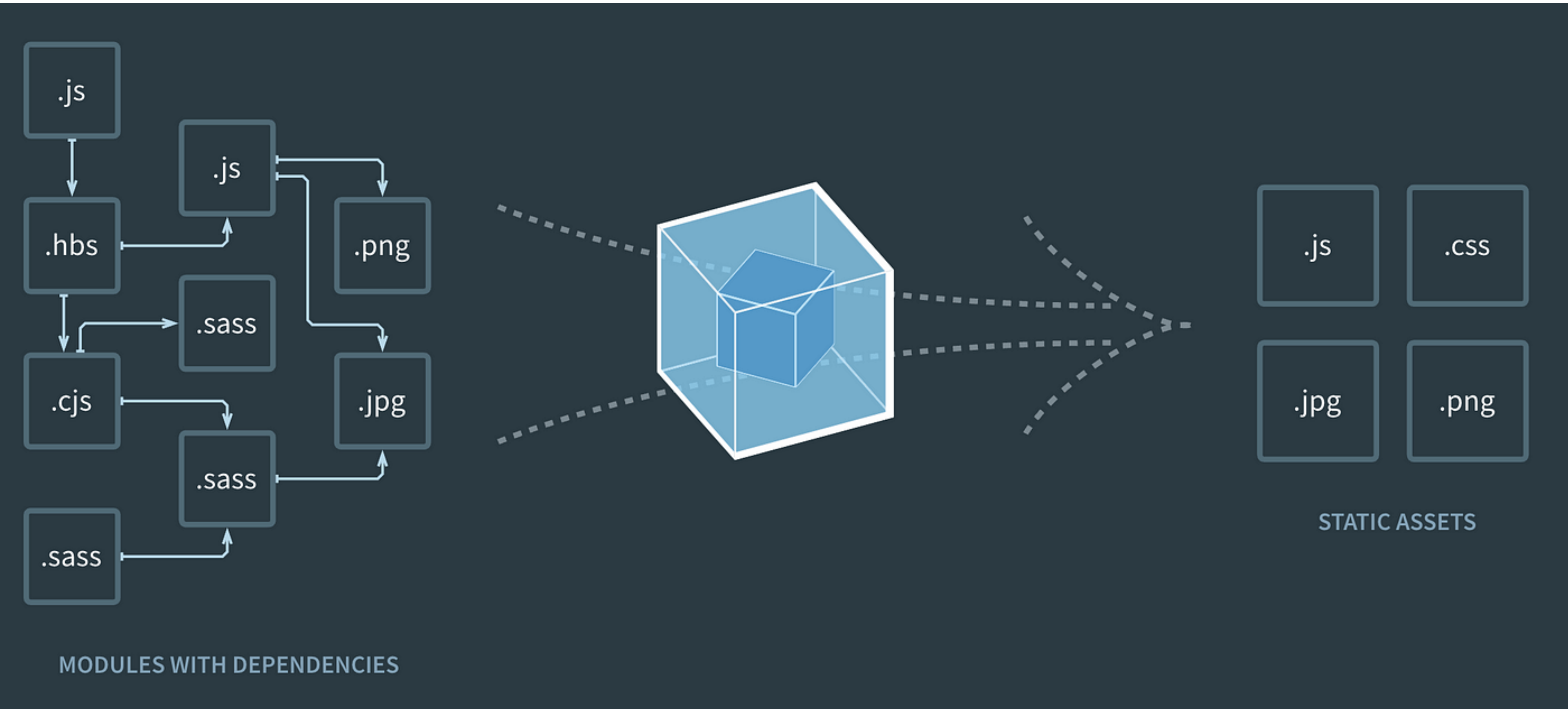
- https://lesscss.org/
- https://sass-lang.com/
- Extensions of CSS that are compiled into CSS.
- (Global) variables and basic calculations, hierarchical styles, mix-ins, inheritance, ...
    - Hierarchical styles are now part of CSS too!
- `webpack` loaders (`less-loader`, `sass-loader`)

```
@primary:  green;
@secondary: blue;

.section {
  color: @primary;

  .element {
    color: @secondary;
  }
}
```

Simple... right?



Developer Hygiene

How often do you brush your teeth?

prettier

Code formatting (prettier)

- https://prettier.io/
- Automatic code formatting: Just use it. No excuses :)
  - Essentially no downsides or overhead, and your code will be easier to read by everyone, not just you
- Configuration in `.prettierrc` (change the default styling choices)

```
# Format all files in ./src
npx prettier ./src --write
# Check that all files in ./src are formatted correctly
npx prettier ./src --check
```

eslint

Linter (eslint)

- https://eslint.org/
- Detect common issues in JavaScript (and TypeScript)
  - Also almost no reason not to use it. Might need some time investment to learn the recommended constructs and patterns, but will almost certainly save you from a lot of common bugs
- Configuration in `.eslint.config.js` (turn rules on/off, include/exclude files, etc.)

```
# Check files in the ./src folder
npx eslint ./src
```

jest

Unit testing (jest)

- https://jestjs.io/
- Tests are usually placed in *.test.js files (but other options are supported).
- Tests assert facts using `expect` + matcher (`toBe`, `toEqual`, `toBeNull`, `toBeGreaterThan`, ...).
    - Why matchers? Comparisons are generally tricky, and we also need a mechanism that generates nice error messages.

```
import { sum } from "./some_module.ts";

test("adds 1 + 2 to equal 3", () => {
  expect(sum(1, 2)).toBe(3);
});
```

```
# This one is really simple
npx jest
```

User interfaces are notoriously hard to unit test

Usually, unit testing is mostly reserved for the "business logic" in your code since they don't run in the browser playwright

End-to-end/interaction tests (playwright)

- https://playwright.dev/
- Runs in the actual browser (you can even test multiple browsers).
- Locators: `page.getByRole` (based on accessibility attributes), `page.getByText`, ...
    - `page.locator("css-selector")` exists, but is discouraged.
- Navigation: `page.goto(address)`, `el.click()`, ...
- Tests can be generated from "recordings".

```
test("profile-picture-visible", async ({ page }) => {
  await page.goto("/");
  await expect(page.getByAltText("Profile picture")).toBeInViewport();
});
```

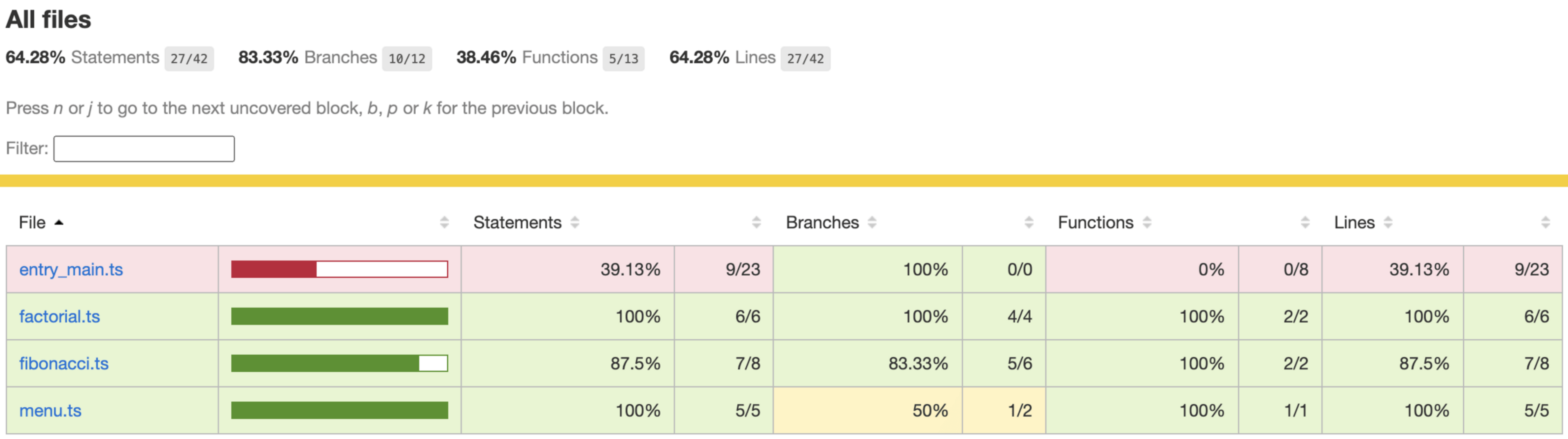
```
# Run all the tests in the background
npx playwright test
# Run tests with interactive UI
npx playwright test --ui
# Show a visual report with test results
npx playwright show-report
```

istanbul (+ nyc)

Code coverage (istanbul + nyc)

- https://istanbul.js.org/
- Who watches the watchmen?
- Code coverage isn't perfect, but gives us some insight into what is tested and what isn't
- For unit tests, code coverage is relatively easy (`jest --coverage`)
- For or E2E tests, TypeScript, etc., things get tricky...
- Babel for the rescue! (istanbul plugin)

```
# Run executable JS module with coverage collection
npx nyc some_npm_module
# Generate JSON coverage report from collected data
npx nyc report --reporter=json
# Combine coverage data from multiple runs (e.g. unit and e2e tests)
npx istanbul-merge --out combined.json partial_a.json partial_b.json
# Generage a HTML report from JSON reports
npx istanbul --include combine.json report html
```



All files

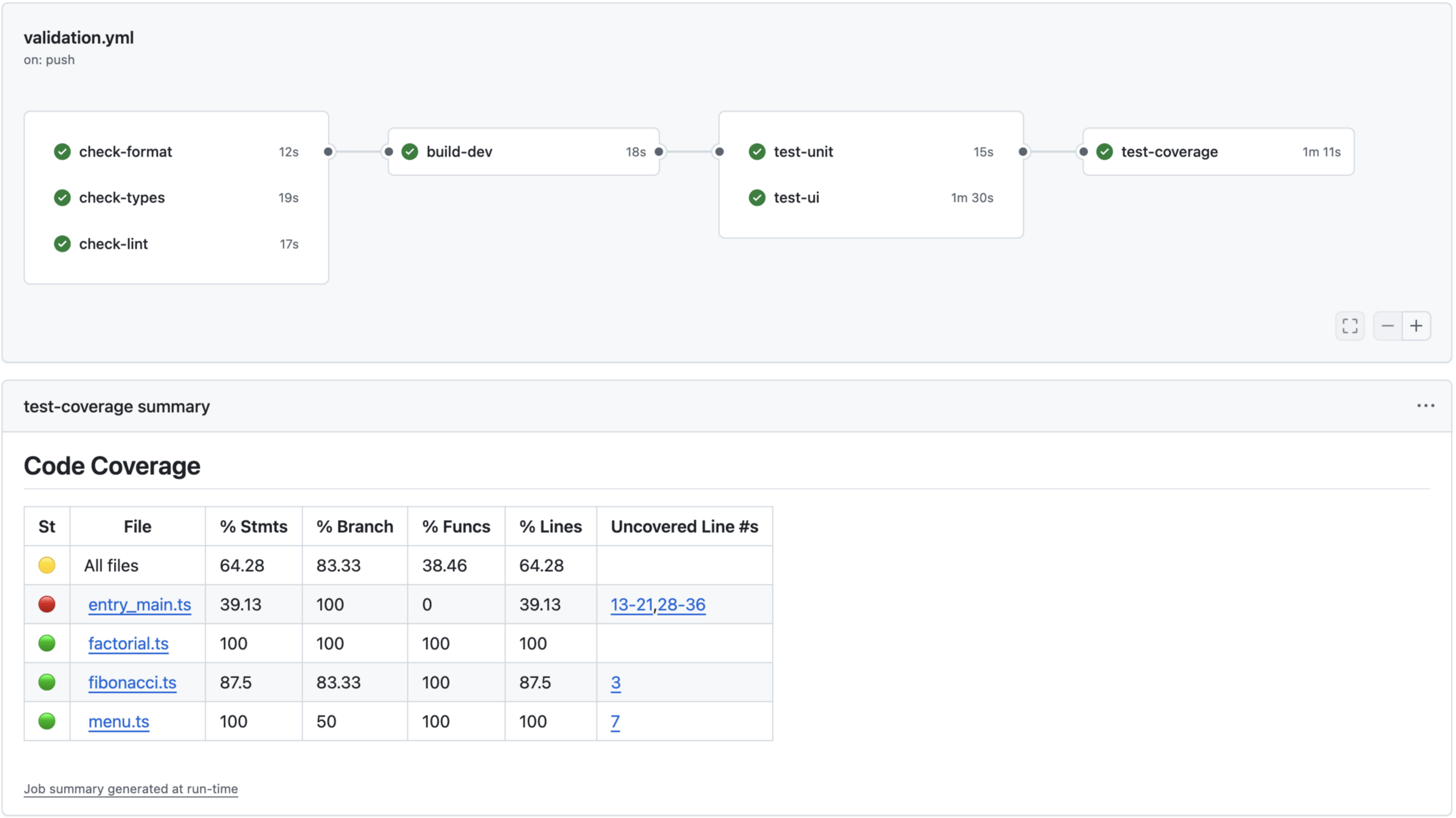
64.28% Statements `27/42` **83.33%** Branches `10/12` **38.46%** Functions `5/13` **64.28%** Lines `27/42`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File ▲		Statements		Branches		Functions		Lines	
entry_main.ts	🟥	39.13%	9/23	100%	0/0	0%	0/8	39.13%	9/23
factorial.ts	🟩	100%	6/6	100%	4/4	100%	2/2	100%	6/6
fibonacci.ts	🟩	87.5%	7/8	83.33%	5/6	100%	2/2	87.5%	7/8
menu.ts	🟩	100%	5/5	50%	1/2	100%	1/1	100%	5/5

CI…



validation.yml
on: push

test-coverage summary

Code Coverage

St	File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
🟡	All files	64.28	83.33	38.46	64.28	
🔴	entry_main.ts	39.13	100	0	39.13	13-21,28-36
🟢	factorial.ts	100	100	100	100	
🟢	fibonacci.ts	87.5	83.33	100	87.5	3
🟢	menu.ts	100	50	100	100	7

Job summary generated at run-time

Tasks for today

- Fork the example project on Github.
  - https://github.com/daemontus/pv252-project-template
- Make a commit that causes the CI *code formatting* check to fail and then fix it.
- Make a commit that causes the CI *type checking* to fail and then fix it.
- Make a commit that causes the CI *unit tests* to fail and then fix it.
- Make a commit that causes the CI *end-to-end* tests to fail and then fix it.
- Add new unit/e2e tests to improve code coverage.
- ***Read more about testing with Playwright and create 4-5 meaningful tests for a website you visit regularly.***
  - *(*These can be placed in the example project too)