Browsers and Web Applications

Browsers

Web Browsers (Short Version)

A web browser is an interpreter for three languages:

- **HTML** describes a page as a tree of elements, such as `<div>` (a box) and `<button>`.
- **CSS** describes how each element is displayed: its size, color, font, and position.
- **JavaScript** describes the behavior of the page. Gleam compiles to JavaScript.

These languages combine to create a page:

- Parsing HTML creates a Document Object Model (DOM), which is the tree of elements.
- CSS can set how elements are styled.
- JavaScript accesses the DOM and can change its structure or attributes, changing the page.

Event-Driven Programming

Web applications are event driven: code is executed based on user actions and other events

- The user clicks a button
- Some scheduled time is reached
- New information is retrieved by the app

JavaScript functions called *event handlers* are associated with elements in the DOM

When some event occurs, the browser calls its handler, which can change the DOM

```
document.addEventListener('click', (e) => {
  e.target.remove();
});
```

Web Applications

Browsers also provide a large inventory of standard application components:

- **Ways to take input:** buttons, text entry boxes, checkboxes, menus, sliders, etc.
- **Ways to display output:** text rendering, layout controls, image/video display, etc.
- **Ways to communicate:** links between pages and network communication with a server
- **Ways to use the computer/device:** local storage, camera & microphone, location, etc.

Pages are *sandboxed*: they can't read your files or use your camera without permission.

Interface components are accessible, supporting screen readers & different input methods

Web Frameworks

Model, View, Update (Short Version)

The **model** is a data structure describing the entire logical state of the application

The **view** is a function of the **model** that returns the tree of elements to display

Changes: update the **model**, then the framework calls **view**, then edits the DOM efficiently

The **update** function takes the current **model** and a **message**, and returns an updated **model**

The **view** annotates each element with the **message** that will be sent when a user interacts
(E.g., when a user clicks this *save* button, send the *UserClickedSave* message)

The update function can also return follow-on effects from a message
(E.g., for a *UserClickedSave* message, the follow-on effect would be a network request)

Effects can send additional messages, which are then passed to update

A Brief History of Functional Web Programming

2014: The Elm Architecture (TEA)

Introduced the model–view–update pattern

Purely functional

Motivation: no runtime exceptions!

2015: Redux

Introduced at Facebook to help with scaling

Adapted Elm framework to work within Javascript and React ecosystem

2023: Lustre

Modernized implementation of The Elm Architecture, in Gleam

The Lustre Framework (Short Version)

The **model** doesn't use any web-specific features; it's just a regular Gleam type

A **message** doesn't use any web-specific features; it's just a regular Gleam type

The **view** function uses Lustre-specific Gleam functions that create:

- Elements and text nodes
- Attributes that style elements
- Attributes that describe the events that trigger messages sent to the update function

The **update** function is almost just regular Gleam, but also includes Lustre-specific effects

https://lustre.hexdocs.pm/

Example: Hog GUI

Hog GUI Design

The Hog GUI uses Python for the game simulation and Gleam for the web interface

Client (Gleam)

1. Configure a game
2. Let human players take turns
3. Show outcomes

Server (Python)

1. Randomly roll dice
2. Simulate a game
3. Computer strategies

(Demo)