Week 1 – Web Development

Web Project Structure

Introduction to Vanilla Websites

A "vanilla" website typically refers to a standard website built using HTML, CSS, JavaScript, and potentially server-side scripting languages like PHP or Perl. This guide focuses on the foundational structure of such projects. More complex applications (e.g., Java, NodeJS) will have different project structures.

Prerequisites

Before diving into web project structure, ensure you have a solid understanding of:

- **Files and Directories (Folders):** What they are, how they are organized.
- **Basic File Management:** Creating, moving, copying, deleting, renaming, and backing up files and directories.

Several resources are recommended in the video description for these foundational skills.

Local vs. Remote Development

- **Local Machine:** You develop your website on your own computer using code editors (like VS Code) and graphics programs.
- **Remote Server:** Your website files are uploaded (published) to a remote server so users can access them via the internet.
- **User Access:** Users only see the version of the files on the remote server.
- **Updates:** Any changes made to your local files must be uploaded to the server to reflect on the live website.

Project Root Directory

- **Definition:** The main folder that contains all files and subdirectories for a single web project.
- **Uniqueness:** It should *only* contain files and directories for that specific project.
- **Naming:** Choose a descriptive name (e.g., `yarn_shop`, `pet_store`).
- **Organization:** For multiple projects, you might have a parent directory (e.g., `my_projects`) containing individual project root directories.

Essential Files and Directories within the Project Root

index.html

- **Purpose:** This is the default file that loads when a user navigates to a directory URL without specifying a filename.
- **Naming Convention:** Must be `index.html` (all lowercase).
- **Case Sensitivity:** While file and directory names are often not case-sensitive on local systems, they *are* case-sensitive on most servers. `index.html` is different from `Index.html`.
- **Automatic Loading:** If a user visits `yourwebsite.com/directory/` and an `index.html` file exists in that directory, it will load automatically.
- **Server Behavior without `index.html`:**
  - **Directory Listing:** Some servers will display a list of files and folders within the directory.

- **Forbidden/Error:** Other servers will return a "403 Forbidden" error, preventing access to the directory contents.
- **Redirect:** An `index.html` file can be programmed (e.g., with JavaScript) to automatically redirect users to another page.

- **Best Practice:** It's generally recommended to have an `index.html` file in every directory that users might access to control what they see and prevent directory listings. However, for course assignments, this might not be strictly required.

Subdirectories

These folders organize different types of assets for your project.

`images/`:

- Stores all image files (e.g., `.jpg`, `.png`, `.gif`).
- Can have sub-subdirectories for further organization if you have many images.
- An `index.html` can be placed here to prevent direct browsing of images.

`css/` (or `styles/`):

- Stores all your Cascading Style Sheets (CSS) files, which control the presentation and layout of your website.
- An `index.html` can be placed here.

`scripts/` (or `js/`):

- Stores all your JavaScript (or other client-side scripting) files.
- An `index.html` can be placed here.

Other Potential Directories

Larger or more complex websites might include additional directories for specific assets or technologies (e.g., `fonts/`, `videos/`, `data/`).

Example Project Structure (Local)

```
my_projects/
├── yarn_shop/          <-- Project Root
│   ├── index.html
│   ├── css/
│   │   └── style.css
│   ├── images/
│   │   └── logo.png
│   └── scripts/
│       └── main.js
├── pet_store/          <-- Another Project Root
│   ├── index.html
│   ├── css/
│   │   └── pet_styles.css
│   └── images/
│       └── dog_photo.jpg
└── portfolio/          <-- Another Project Root
```

- ├── index.html
└── ...

Publishing to the Server

- **Tools:** File Transfer Protocol (FTP) clients like FileZilla are recommended for uploading files. CPanel can be used but is often more tedious.
- **Server Structure:** You will mirror your local file and directory structure on the remote server.
- **Home Directory:** Your server space is organized into a "home directory," which is your personal allocated space.
- **public_html/ Directory:**
  - This is a special directory within your home directory.
  - **All publicly accessible web content must reside within public_html/.**
  - Files outside of public_html/ cannot be accessed via a web browser.
  - The URL to access your website typically points to the public_html/ directory. For example, yourusername.server.com often resolves to yourusername/public_html/.
- **Uploading:**
  - **Initial Upload:** Upload the entire project root directory (e.g., yarn_shop) into your server's public_html/ directory.
  - **Updates:** When you make changes to local files, you only need to upload the modified files.
- **Accessing Your Site:**
  - The URL to access your website is usually structured as your_login_name.your_server_domain.com.
  - Navigating to a directory within your public_html/ (e.g., your_login_name.your_server_domain.com/yarn_shop/) will automatically look for index.html if it exists.

File Synchronization and Updates

- **Local vs. Server Timestamp:** When uploading files, you'll notice timestamps. Your local file's timestamp indicates when you last saved it. The server's timestamp indicates when it was last uploaded.
- **Overwriting:** If your local file is newer than the server version, you will need to overwrite the server file. FTP clients will prompt you about this.
- **"Always use this action":** For a single FTP session, you can select "always use this action" (e.g., "overwrite") to avoid repeated prompts.
- **Automation:** Some code editors have extensions that can automatically upload files as you save them, though manual uploads offer more control over when changes go live.

Benefits of Standard Structure

- **Maintainability:** Easily locate and modify specific files and components of your project.
- **Organization:** Keeps your project tidy and understandable.
- **Scalability:** Simplifies adding new features or sections (e.g., a new tutorial directory) by following the established pattern.
- **Collaboration:** Makes it easier for multiple developers to work on the same project.

Web Development Workspace and Webspace Setup

Introduction to Workspace and Webspace

This guide covers the essential steps for setting up your development environment for web courses, focusing on project structure and publishing your work.

Key Concepts

- Directory: The technical term for what is commonly known as a "folder." It's a container for files and other directories.
- File Extension: A suffix to a filename that indicates the file type (e.g., .html, .css, .js).
- Workspace: Your local computer environment where you create and manage your project files.
- Webspace: The remote server space where your website files are hosted and made accessible via the internet.

Prerequisites

Before proceeding, ensure you understand:

- Files and directories (folders).
- File extensions.
- Basic file and directory operations: creation, moving, copying, and deletion.
- Recommended prerequisite video resources are available in the course materials and linked in the video description.

Structuring Your Project Directories

A well-organized directory structure is crucial for managing your web development projects.

General Recommendations

- Centralized Course Storage: It's recommended to store all your course-related files in a single main directory, such as Documents.
- Term-Based Subdirectories: Within your main course storage directory, create subdirectories for each academic term (e.g., Term 1, Term 2). This helps in organizing files from different semesters.
- Course-Specific Subdirectories: Inside each term directory, create a subdirectory for each course. You can use the course code (e.g., SYT10049) or a descriptive name (e.g., WebDev, Python).

Web Development Course Structure

For web development courses, a common and effective structure within the course-specific directory (e.g., WebDev) is as follows:

- Weekly Directories: Create a subdirectory for each week of the course (e.g., Week 1, Week 2, Week 3, ... Week 14).
    - Purpose: Each weekly directory will contain the projects, demonstrations, exercises, and example code from that specific week's lessons.

    - Benefit: Organizing by week prevents file name conflicts (like multiple index.html files) and allows you to easily access and review previous work. This is important for understanding progression and for potential modifications of older code.
- Assignments Directory: Create a dedicated directory named Assignments.
    - Purpose: This directory will store all your assignment submissions.
    - Sub-Assignments: Within the Assignments directory, create subdirectories for each assignment (e.g., Assign 1, Assign 2, Assign 3, Assign 4). The number of assignment directories may vary depending on the course.

Example Directory Structure:

```
Documents/
└── Term 1/
    └── WebDev/
        ├── Week 1/
        ├── Week 2/
        │   ├── index.html
        │   ├── images/
        │   │   └── example.jpg
        │   └── CSS/
        │       └── style.css
        ├── Week 3/
        ├── ...
        └── Assignments/
            ├── Assign 1/
            ├── Assign 2/
            ├── Assign 3/
            └── Assign 4/
```

Testing Your Projects Locally

Before publishing your web projects, you can test them on your own computer.

1. Locate the index.html file for the project you want to test.
2. Double-click the index.html file.
3. Your default web browser will open and display the webpage.

This allows you to see how your site looks and functions without needing an internet connection or a remote server.

Publishing Your Web Projects

Publishing your web projects makes them accessible to others via the internet. This is often required for assignments and certain exercises. Two primary methods are discussed: FileZilla and cPanel.

Using FileZilla (Recommended)

FileZilla is an FTP (File Transfer Protocol) client that provides an efficient way to upload files to your webspace.

1. Install FileZilla Client: Download and install the FileZilla client (not the server version).

2. Configure Connection: Set up a connection to your web server (e.g., dev.fast.sheridanc.on.ca). Connection details are typically provided in course materials.
3. Connect to the Server: Once configured, connect to the server.
4. Navigate to public_html: In the remote site pane of FileZilla, navigate to the public_html directory. This is the root directory for your publicly accessible website files.
5. Upload Files/Directories:
    - In the local site pane, navigate to the project directory you want to upload (e.g., WebDev/Week 2).
    - Drag and drop the desired files or the entire directory (e.g., Week 2) from the local pane to the public_html directory in the remote pane.
6. Verify Upload: Check that the files and directories have been successfully transferred.

Advantages of FileZilla: Generally faster and more user-friendly for uploading files and directories compared to cPanel.

Using cPanel (Alternative)

cPanel is a web hosting control panel that provides a graphical interface for managing your web hosting account, including file management.

1. Access cPanel: Go to the cPanel URL (e.g., cpanel.dev.fast.sheridanc.on.ca).
2. Log In: Use the "Login via Sheridan" option and your Sheridan credentials.
3. Navigate to File Manager: On the cPanel dashboard, find the "Files" section and click on "File Manager."
4. Navigate to public_html: Ensure you are in your public_html directory. If not, navigate to it.
5. Upload Files:
    - Click the "Upload" button.
    - You can select files to upload or drag and drop them.
    - Note: cPanel's file manager may have limitations on directly uploading entire folders. You might need to create directories manually within cPanel first and then upload files into them, or upload individual files. This can be more tedious than using FileZilla.
6. View Your Published Site: Access your published content via its URL. The general format is username.dev.fast.sheridanc.on.ca/directory_name/. For example, if your username is myuser and you uploaded a Week 2 directory, the URL would be myuser.dev.fast.sheridanc.on.ca/Week 2/. If index.html is present in that directory, you can often omit it from the URL

Disadvantages of cPanel: Can be less intuitive and more time-consuming for file uploads, especially if you need to upload many files or entire directory structures.

Password Protecting Directories (Assignments)

For assignments, it's crucial to protect them from public access to maintain academic integrity.

- Requirement: The Assignments directory on the server must be password protected.
- Purpose: This ensures that only authorized individuals (you and your instructor) can view your submitted assignments.
- Mechanism: This is typically achieved through cPanel's "Directory Privacy" feature.
- Password: The password used for this protection is not your Sheridan network password. It's a separate password you set up.

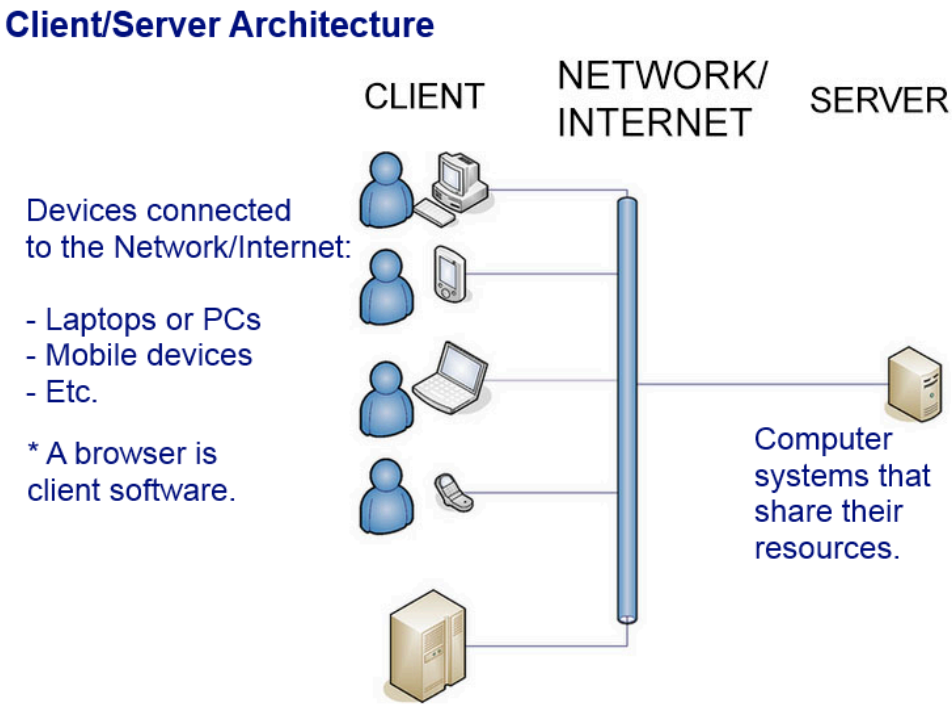
- Instructions: Detailed, step-by-step instructions for password protecting your Assignments directory using cPanel are available in the written course materials. These instructions are usually not demonstrated directly in videos to avoid complexity and potential security issues with recording.

Key Takeaways

- Organization is Key: A structured directory system (by term, course, and week) will save you time and prevent errors.
- Local Testing is Essential: Always test your work locally before publishing.
- FileZilla is Recommended: For publishing, FileZilla is generally a more efficient tool than cPanel's file manager.
- Secure Your Assignments: Always password protect your Assignments directory on the server.
- Refer to Written Materials: The accompanying written documentation provides more detailed explanations, terminology, and specific instructions.

What is a Client – server?

The client–server model is when one main computer, called the server, stores information and programs that other computers, called clients, can use through a network. The Internet works this way—servers hold the data, and our devices (clients) connect to them to access what we need.



Client–Server Example: Internet

Server – is both the computer and the software that respond to other computers' requests by sharing data, programs, or services. There are many kinds of servers—for example, web servers show websites, file servers store files, and game servers let people play games together online.

Client – can be a computer, phone, or even a program that connects to a network to request information or services from a server. When you use a web browser to visit a website or an email app to check messages, those programs act as client software communicating with servers. Basically, any device or app that asks a server for data and shows it to you is considered a client.

Web Applications

In web development, the client-server model means some work happens on the server and some happens on the user's device. Languages like PHP and MySQL run on the server, while HTML, CSS, and JavaScript run on the client's computer to display and style the webpage.

Ex. technologies used in web development:

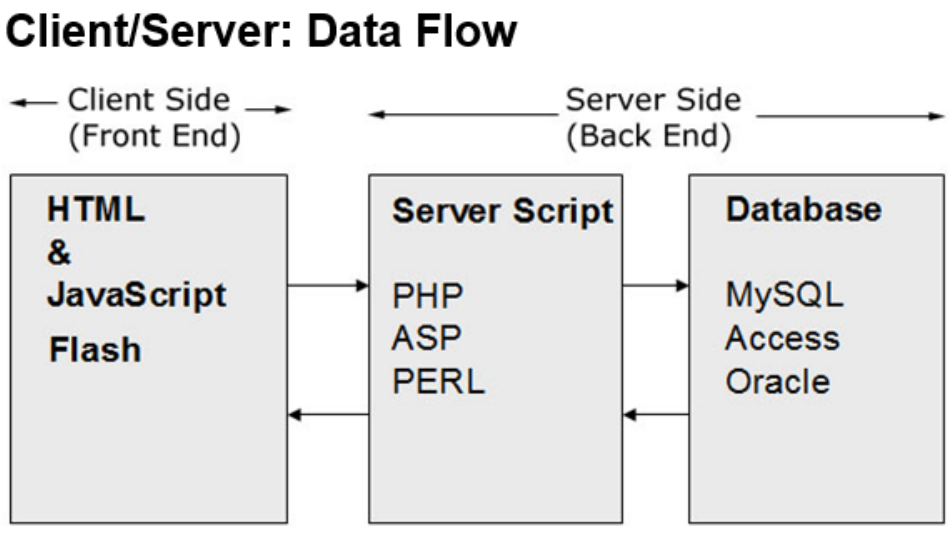
Client-side:

- **HTML**: It builds the structure and content of a webpage that your computer displays.
- **CSS**: It controls the layout, colors, and style of a webpage seen on your device.
- **JavaScript (client-side)**: It makes webpages interactive by running code directly on your browser.

Server-side:

- **JavaScript (server-side with Node.js)**: It can also run on servers using Node.js to handle data and requests.
- **PHP**: It runs on the server to process client data, manage files, and connect with databases.
- **MySQL**: It's a server program that stores and organizes large amounts of data for websites to use.

The way data moves between clients and servers can get complicated when building interactive websites or web apps. To start learning, we focus on static web pages—simple pages that don't change unless the developer edits and re-uploads them.



Example of Client-Server Data Flow

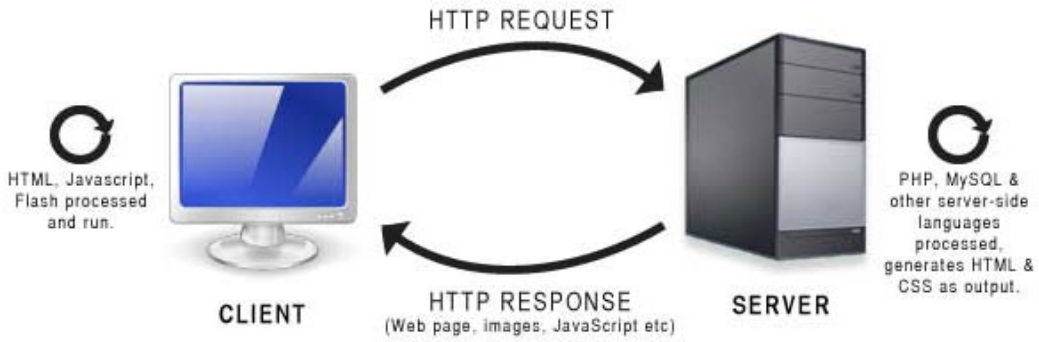
The Request/Response Model

The Request/Response Model means the client asks the server for something, the server processes it, and then sends an answer back to the client.

For example, say you visit the page Wendi's Cats:

1. When you visit the *Wendi's Cats* page, your browser creates an HTTP request asking for that webpage.
2. HTTP is a set of rules that let information, like web pages, travel across the Internet.
3. The HTTP request includes details about what file you want and where the request is coming from.
4. Your browser sends this request to the server at *www-acad.sheridanc.on.ca*.
5. The server receives the request and finds the *wendisCats.html* file in its directory.
6. The server turns the file into data that can be sent back to your computer.
7. A response object is created that contains your computer's address, the data for the file, and a success message (code 200 OK).
8. Different response codes tell you if something went wrong, like 403 (forbidden) or 404 (not found).
9. The server sends the response back to your browser.
10. Your browser opens the response, reads the data, and builds the webpage from it.
11. If the page includes other files like images, CSS, or JavaScript, your browser sends more requests to get each one.
12. The browser receives and combines all the extra files, adds styles and images, and finally shows you the finished webpage.

Ex.



Request/Response model: Shamelessly stolen from
http://demosthenes.info/blog/137/The-Client-Server-Model

Web Application Development
(Activity / Demonstration)
Web Development Process

Web development is more than just writing code with HTML, CSS, and JavaScript. It starts with understanding the website's purpose and who will use it. Developers then plan how the pages connect, what each page looks like, and how the design elements such as colors and fonts will fit together.

NOTE: A website is usually meant to share information, like articles or company details, while a web application lets users do interactive tasks, like posting messages or using calculators. Over time, the difference between websites and web applications has become smaller as technology has improved. Both can include static content that doesn't change or dynamic content that updates as users interact. In web development, the same tools are used for both, so the terms "website" and "web application" are often used interchangeably.

Different people describe the Web Development Life Cycle with different numbers of stages, but they all cover the same basic steps. What matters isn't how the stages are labeled, but understanding the main tasks and their proper order. The key is to learn what each step involves and how they fit together in building a website.

Stages of the Web Development Life Cycle

1. Definition

The definition stage is the first and most important part of web development because it ensures the website meets the client's goals and audience needs. In this stage, developers gather details about what the site is for, what content it needs, and what the client wants to achieve. This information comes from talking with the client, users, and reviewing any existing materials or old websites.

Your information gathering should answer all of the following questions:

Why does the application need to exist? What is its purpose? These should be worded as "ability" statements. Examples:

- The ability to showcase and sell Client's custom crochet patterns to the community of crocheters.
- The ability to showcase the items Client has made from their own custom crochet patterns, and to allow those who have purchased Client's patterns to show off their own work.
- The ability to allow other crocheters to create accounts so they can make purchases, upload their own photos, and rate the patterns Client has created.
- The ability for Client's customers to share and request assistance and tips for completing patterns, as well as new ideas and alternate stitches, colourways, and yarns.

What are the goals this site should achieve (there should be at least 2)? Examples:

- The goal of this site is to allow other yarn crafters to purchase Client's patterns so Client can make a profit.
- The goal of this site is to increase purchases and profits by allowing users to rate the patterns they've purchased and share pictures of the items they make from the client's patterns.
- The goal of this site is to build a close-knit community where crafters can feel comfortable sharing ideas and helping each other.

Who does this site serve, who is the intended audience? Example:

- This site is for members of the crochet community that are interested in new and unique crochet patterns. Such users tend to be 45-75 years of age with a median income of $40,000 - $75,000 per year..

How does the web site support the goal of the client? How does the site help the client with their business, how does it provide value to the client? Examples:

- *This website helps the client sell crochet patterns, gather feedback to improve them, and create a community where customers can share ideas and reviews that encourage more purchases..*

What kinds of content will the site contain and where will it come from? Will it be static and/or dynamic? Example:

- The website will include photos and written descriptions of crochet patterns, provided by the client and edited by the team.
- The patterns will be stored as secure PDF files that customers can download only after purchasing, with details saved in a database.
- The site will also feature ratings, reviews, tips, and photos shared by customers showcasing their finished creations

What kinds of interactivity will the site contain? Example:

- Users can make their own accounts on the website.
- They can buy and download pattern PDFs, then share ratings, comments, and questions like in a blog discussion.
- The client will have a private area to upload and manage patterns, handle sales and accounts, and review customer content.

After answering all the key questions, you create a report that explains the site's purpose, audience, and content for the client to review. The report is revised until the client approves it, and once finalized, you can start planning the website.

2. **Planning**

The planning stage is important because it helps avoid mistakes, makes coding faster, and must be approved by the client before moving forward.

Typically, the planning stage involves 3 smaller steps:

Project Scope: This means listing all the tasks, their order, and how long each one will take to finish.

- **Team Roles:** If you're working with others, assign who handles content, coding, and database setup.
- **Task Order:** Some tasks depend on others, like needing to choose colors and layouts before coding CSS.
- **Timelines:** Use tools like Gantt Charts to schedule tasks and ensure everything is done by the deadline.

- **Solo Work:** Even if working alone, plan tasks and timing carefully so you stay organized and meet client expectations.

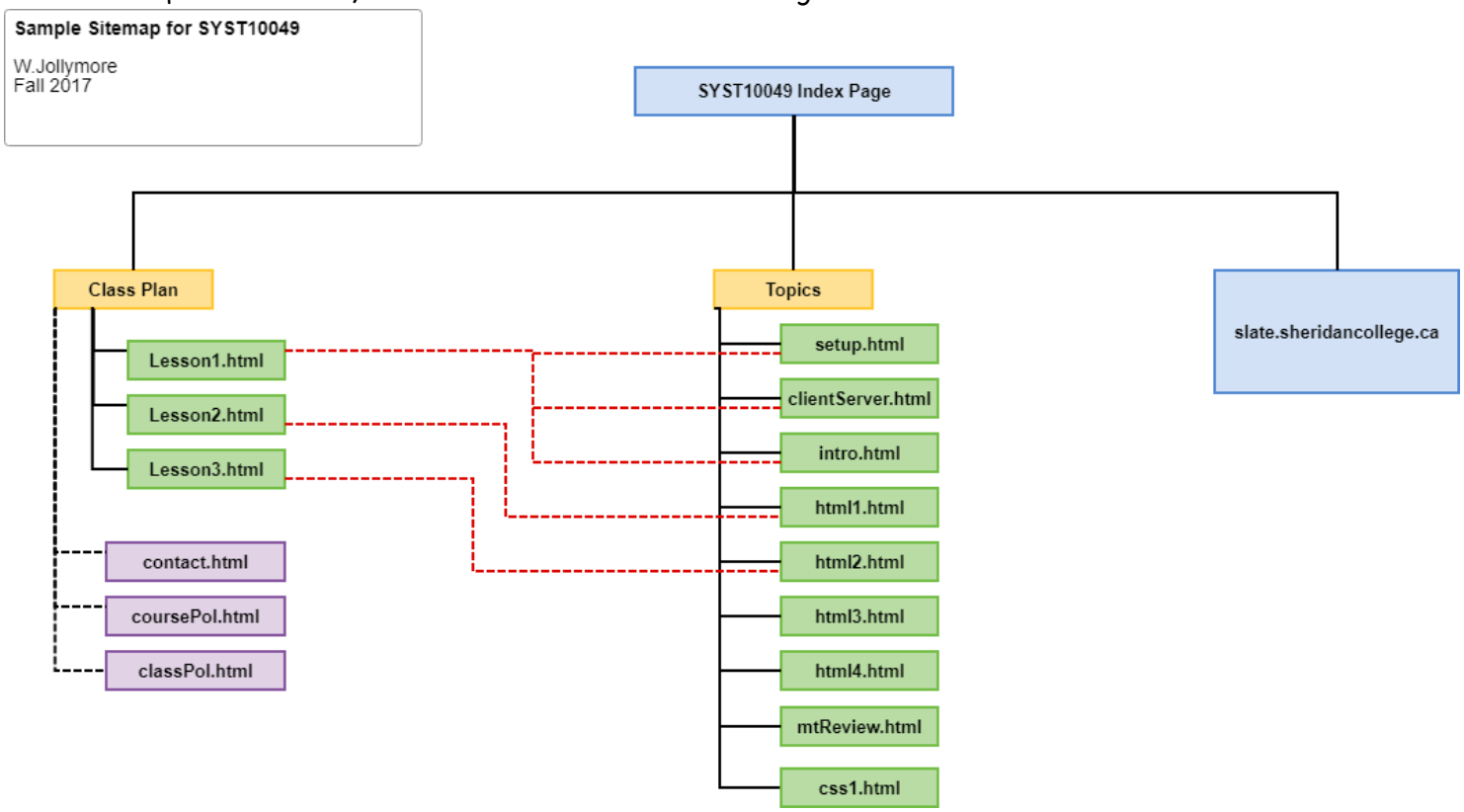
Content: This step involves gathering or creating all the website's content—like text, images, and links—and adding accessible features such as captions or descriptions for people with disabilities.

Important: *Accessibility should always be part of your design so that everyone, including people with disabilities, can use your website or application!*

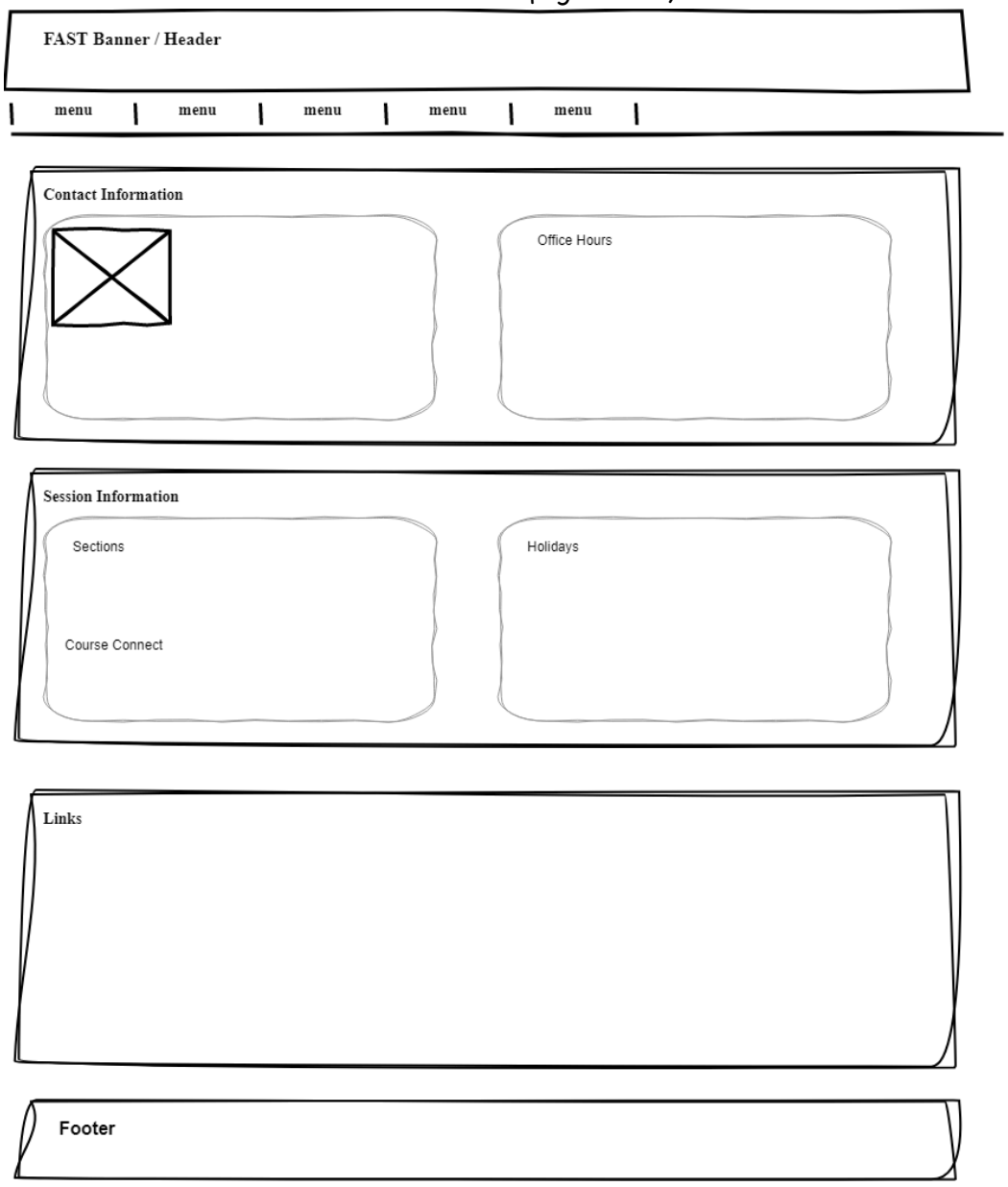
Visual Planning: This step involves creating a site map and wireframes to plan each page's layout and choosing the site's colors and overall theme.

- **Site Map**: A site map is a diagram that shows how all the pages on a website are organized and connected. You can create one using tools like Microsoft Visio or free online options such as Cacoo.
- **Wireframe:** A wireframe shows the basic layout and structure of a web page without real content, often including notes about colors or fonts. Many site map tools can also make wireframes to help plan page design.
- **Styling:** This step involves choosing the site's color scheme—usually a main, accent, and highlight color—along with fonts and accessibility features. It also includes applying consistent branding and design elements, such as logo placement and image display, across all pages.

Ex. a site map for one of my old course web sites at the college where I teach.



Ex. Wireframe – wireframe for one of the pages on my old course web site.



3. Development

Is when you actually write the code for the website. Even though most people think coding is all web developers do, it's just one part of a larger process. In this course, you'll focus on HTML and CSS now, and later learn JavaScript and PHP to make websites more interactive.

Typically, web pages will include code written in:

1. HTML: It's the language that gives a webpage its structure and content.
2. CSS: It controls the design, colors, fonts, and layout of the webpage.
3. JavaScript: It adds movement and interactivity to make webpages dynamic and engaging.

4. PHP, Perl, or other server-side languages: They handle data and create interactive features by working with information stored on the server.

Many tools and frameworks build on basic web languages to make development easier and more powerful. For instance, Bootstrap helps style webpages with CSS, and Node.js lets you run JavaScript on the server. During development, you might also need to create or add content for the site. If your website uses a database, this stage includes building and filling that database with the information the site will display.

4. Testing

Testing means showing the website to a small group of people, including the client and test users, to look for mistakes or issues.
Testers check for broken features, grammar errors, and other problems while using different browsers, devices, and screen sizes.
Accessibility testing ensures the site works for everyone, including people using screen readers or keyboard navigation.
Testing is detailed and formal, involving reports, screenshots, and communication between testers and developers.
After fixing all problems found during testing, the website is ready to launch.

5. Launch

Launching a site means uploading all its files to a web server so it's available on the Internet. If the client doesn't have their own server, you can help them choose or provide a hosting service. After the site goes live, collect feedback from real users to find and fix any issues the test phase might have missed.

6. Maintenance

A website always needs maintenance because users may find problems or request new features over time. You might also need to update content or improve features as technology and user needs change. Make sure there's a way for users to give feedback so the client can tell you what needs fixing while you move on to other projects.

Understanding the Web and the Internet

Many people think the Internet and the World Wide Web are the same, but they aren't. The Internet is a huge network of connected computers that share information worldwide. The Web is just one part of it that lets people access and share information through hypertext pages with clickable links.

Questions

1. Currently, what are the most popular browsers being used? Do you think this means you should only design your web pages/applications to target the functionality/features of those two browsers.

Web developers should make sites work on all browsers and assistive devices so that every user, no matter what they use, can access the content properly.

2. How does a page get loaded into your browser when you click a link/bookmark?

The browser asks the server for a webpage, the server sends it back with a status code like "200 OK" if successful or an error code like "404 Not Found" if something goes wrong, and then the browser displays the page.

3. What is bandwidth? How does it affect the loading of web pages and their files?

Bandwidth is the amount of data a network can send each second—not the speed of travel—so pages with large media files need more bandwidth to load properly.

4. What is the World Wide Web Consortium, what do they do?

The World Wide Web Consortium (W3C) is an organization that creates and maintains standards for how web technologies are built. It also sets guidelines to make web content accessible to everyone, including people with disabilities.

5. The word "Internet" is the shortened form of what word? What does that word mean?

The Internet, meaning "a network of networks," is a huge system made up of many smaller connected networks that share information with each other.

6. Why did Tim Berners-Lee develop a webbed system of hypertext?

Tim Berners-Lee wanted scientists worldwide to easily share information online. Before hypertext, finding a document was slow because files were stored in deep folder structures. His invention of hypertext let people click links to jump straight to related documents, creating the "web" of connected pages we use today.

7. What is ARPANET? Why was it originally developed?

ARPAnet was a project by the U.S. Department of Defense during the Cold War to create a communication system that could survive damage. Messages were split into small packets that traveled different routes to the same destination. This way, if part of the network failed, most packets would still arrive, and the message could be rebuilt.

8. Who controls the Internet?

No single person or group owns the Internet; each network is managed separately, but they all cooperate to stay connected and share information globally.

9. What is a packet? How does a packet get from one place to another? What happens if you want to send a very large file?

A packet is a small piece of a message sent across the Internet. Each packet travels through different routers and can take its own path based on network speed or traffic. When all the packets reach their destination, they're reassembled to recreate the original file or message.

10. What does a router do?

A router directs each packet toward its destination by sending it through several routers along the best available path.

11. What does TCP stand for? What does TCP do?

TCP, or Transmission Control Protocol, is a system of rules that checks and organizes packets when they reach their destination. It makes sure every packet arrives and, if any are missing, asks the sender to resend them so the full message can be rebuilt correctly.

12. What does HTTP stand for? What is HTTP?

HTTP, or Hypertext Transfer Protocol, is the system of rules computers use to send and receive web pages over the Internet. It tells the browser how to ask for a file and tells the server how to respond—whether the file exists, is missing, or access is denied.

13. What does HTML stand for? What is HTML?

HTML, or Hypertext Markup Language, is the code developers use to build the structure and content of a web page, like headers, text, and images. Older versions handled formatting, but now design elements such as colors and layouts are managed with CSS instead.

14. What might a GET REQUEST be for? What might a POST REQUEST be for?

A GET request asks the server for a file or page, while a POST request sends data—like login details or form information—to the server.

15. What is SSL and TLS? How do you know these are working?

SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are technologies that encrypt data to make web communication secure, creating HTTPS. You can tell a site uses HTTPS when you see a lock icon in your browser, and today almost all web traffic is protected this way.

What is a Wire Frame? A comprehensive guide.

What is a Wire frame?

A wireframe is a simple outline that shows how a webpage or app will be organized and how users will move through it. It focuses on structure, layout, and functionality without using colors or detailed design elements. Wireframes can be sketched by hand or made with digital tools, depending on the project's needs. UX designers use them so everyone involved can agree on the layout before developers start coding.

When does wireframing take place? (And is the prototype different?)

Wireframing happens early in the development process when the team explores ideas and defines what the product needs to do. During this stage, designers brainstorm, plan features, and gather user feedback. That feedback helps them move on to create more detailed designs like prototypes or mockups.

What is the purpose of wireframing in UX?

Wireframes help designers stay focused on the user, make site features and navigation clear, and are fast and inexpensive to create.

Wireframes keep the concept user-focused

Wireframing helps gather feedback from users, spark discussions with stakeholders, and encourage idea sharing among designers. Using placeholder text during testing lets designers ask users what they expect to see, revealing their natural instincts. This feedback helps designers make products that feel intuitive and are easy to use.

Wireframes clarify features and navigation

Clients might not understand technical design terms, so wireframes help show them how each feature works and why it's included. By visualizing the layout, everyone can see how the parts of the site fit together. This is the best time for stakeholders to give honest, even harsh, feedback before development begins.

Wireframes are quick and cheap to create

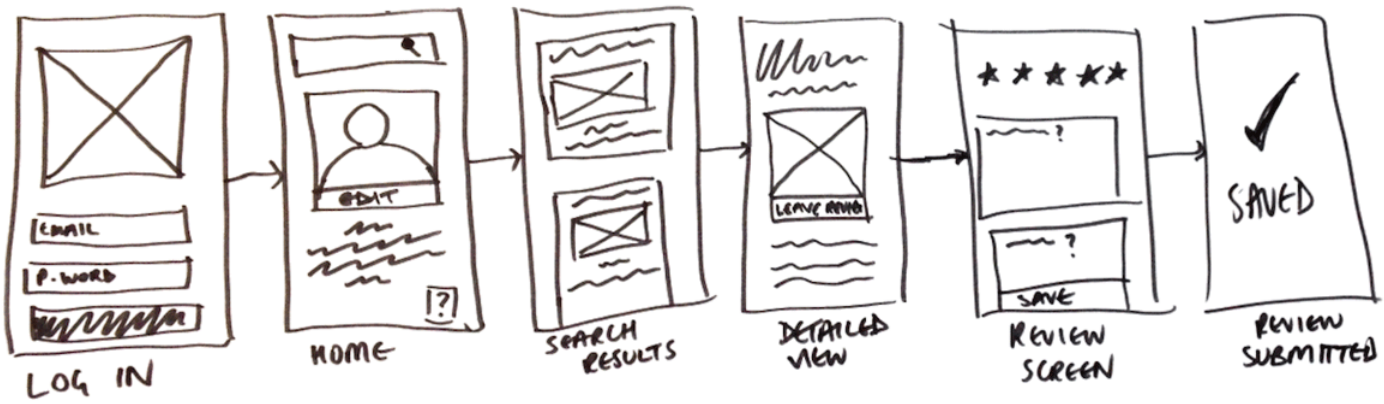
Wireframes are quick and inexpensive to make, whether you sketch them by hand or use digital tools. Keeping them simple helps users give more honest feedback because they focus on layout, not design polish. Making changes is much easier at this early stage than later in the design process.

What are the different types of wireframes?

The wireframe fidelity refers to the level of detail in each design.

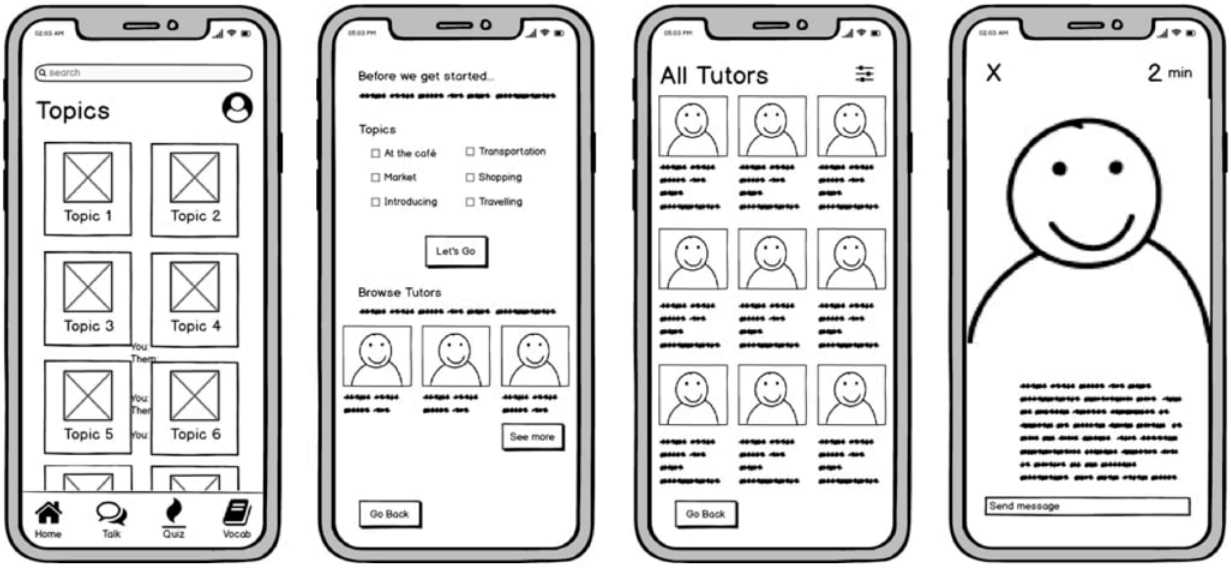
Low-fidelity wireframes

Low-fidelity wireframes are simple, rough sketches that mark the beginning of the design process. They use basic shapes, placeholder text, and minimal detail—without focusing on scale or precision. Designers and stakeholders use them to brainstorm layouts, plan navigation, and map user flows, often sketching ideas quickly to explore different product directions before refining the design.



Mid- Fidelity Wireframes

Mid-fidelity wireframes give a clearer, more detailed view of a website's layout while still avoiding distractions like images or fonts. They use different text sizes and shades of gray to show hierarchy and emphasize important elements. Designers typically create them with digital tools such as Sketch or Balsamiq to refine structure and functionality before adding visual design.



High – Fidelity Wireframes

High-fidelity wireframes are detailed, pixel-accurate designs that often include real images and text. They're used to visualize complex features like menus or interactive maps, helping designers refine how the final product will look and function.

What is included in a wireframe?

Wireframes usually include basic elements like logos, search bars, headers, share buttons, and placeholder text such as Lorem Ipsum. High-fidelity versions may also feature navigation menus, contact details, and footers for a more realistic structure. Although typography and images aren't used in low- or mid-fidelity wireframes, designers may adjust text sizes to show content hierarchy. Most wireframes are in grayscale, but high-fidelity ones might use limited color—like red for errors or dark blue for active links. However, wireframes don't display interactive elements such as drop-downs, hover effects, or expanding sections.

Desktop vs. mobile and app wireframes

When we think of wireframes, we often cast our minds to desktop site design. But mobile devices require special considerations.

Size

On a desktop site, the layout can use several columns to organize content, while mobile apps typically use one or two columns for easier viewing. Designers must also decide whether users should scroll continuously or see fewer items per page to make room for other content.

Behaviour

On a website, users typically navigate using a mouse or trackpad, clicking or hovering over elements to access menus or extra information. In contrast, mobile app users rely on tapping the screen to perform actions, which means designers must plan interactions more intentionally. When creating wireframes for mobile, it's important to think about how to guide users to tap specific buttons or features that help them reach their goals smoothly.

Interaction

Some apps connect to the internet to load data, but they also let users download content to use offline. When designing wireframes for mobile apps, it's important to include how this offline mode will look and function so the experience works smoothly without an internet connection.

What tools are used to create wireframes?

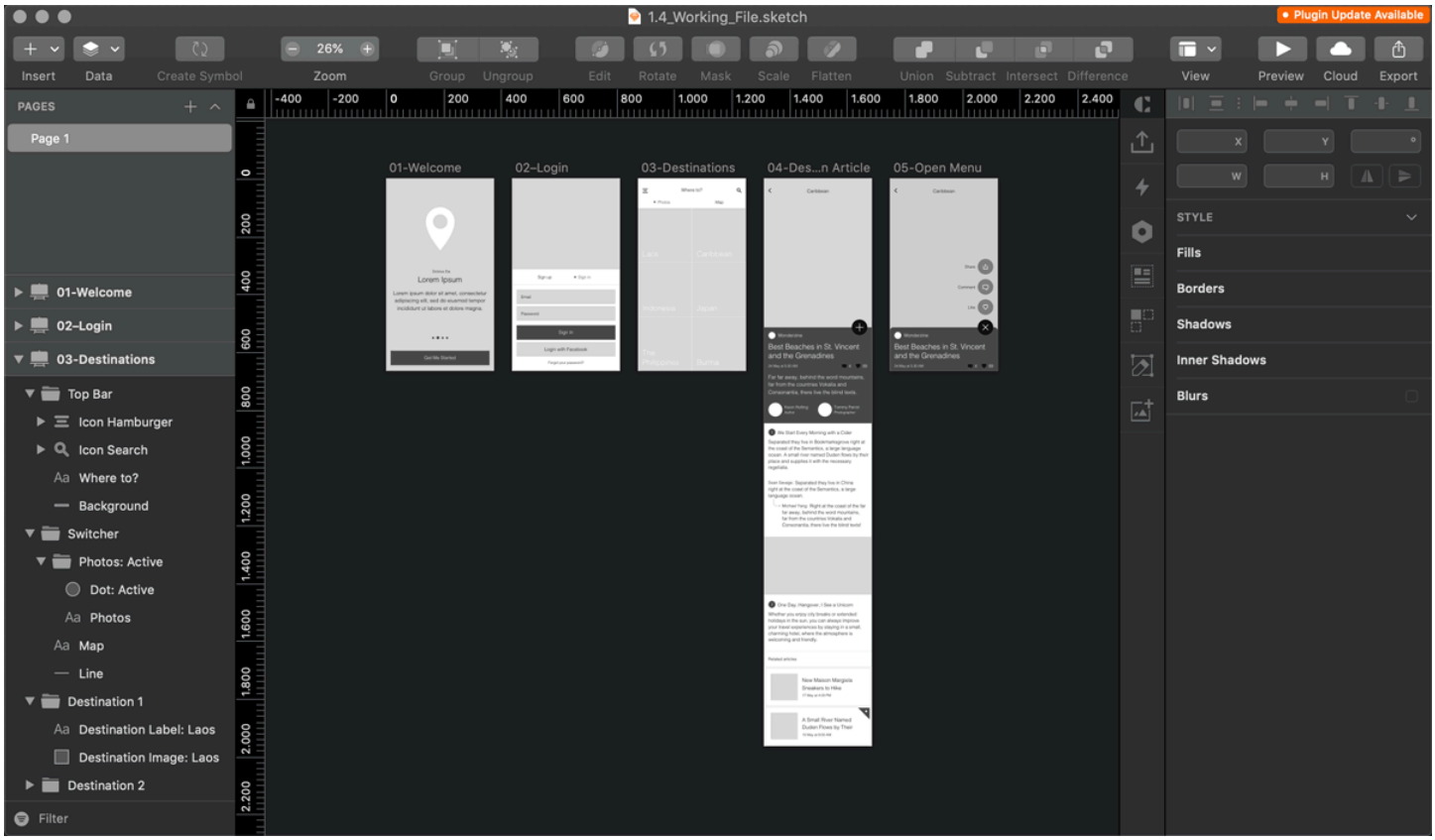
Designers now have access to powerful wireframing tools that come with built-in UI components like buttons, forms, and navigation elements. These pre-made features save time by eliminating the need to design everything from scratch, making it easier to quickly create and refine wireframes through an efficient, iterative process.

Sketch

Sketch is a popular wireframing tool that lets designers create layouts using artboards and vector shapes on a pixel-based canvas. It also includes a handy Symbols feature, which allows users to reuse UI elements they've already made, saving time and ensuring consistency across designs.

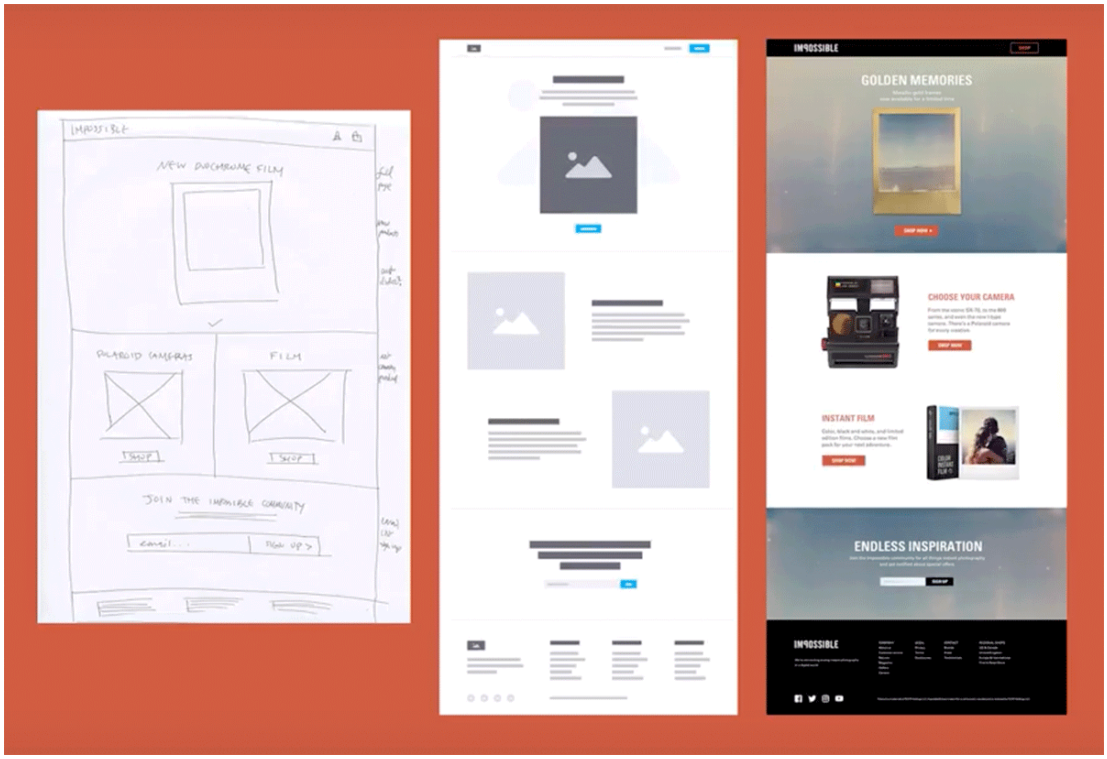
Balsalmiq

Balsamiq is a popular wireframing tool perfect for designers who want a digital option more polished than paper sketches but not focused on pixel-perfect details. It helps you concentrate on layout, user interaction, and the overall structure of your design.

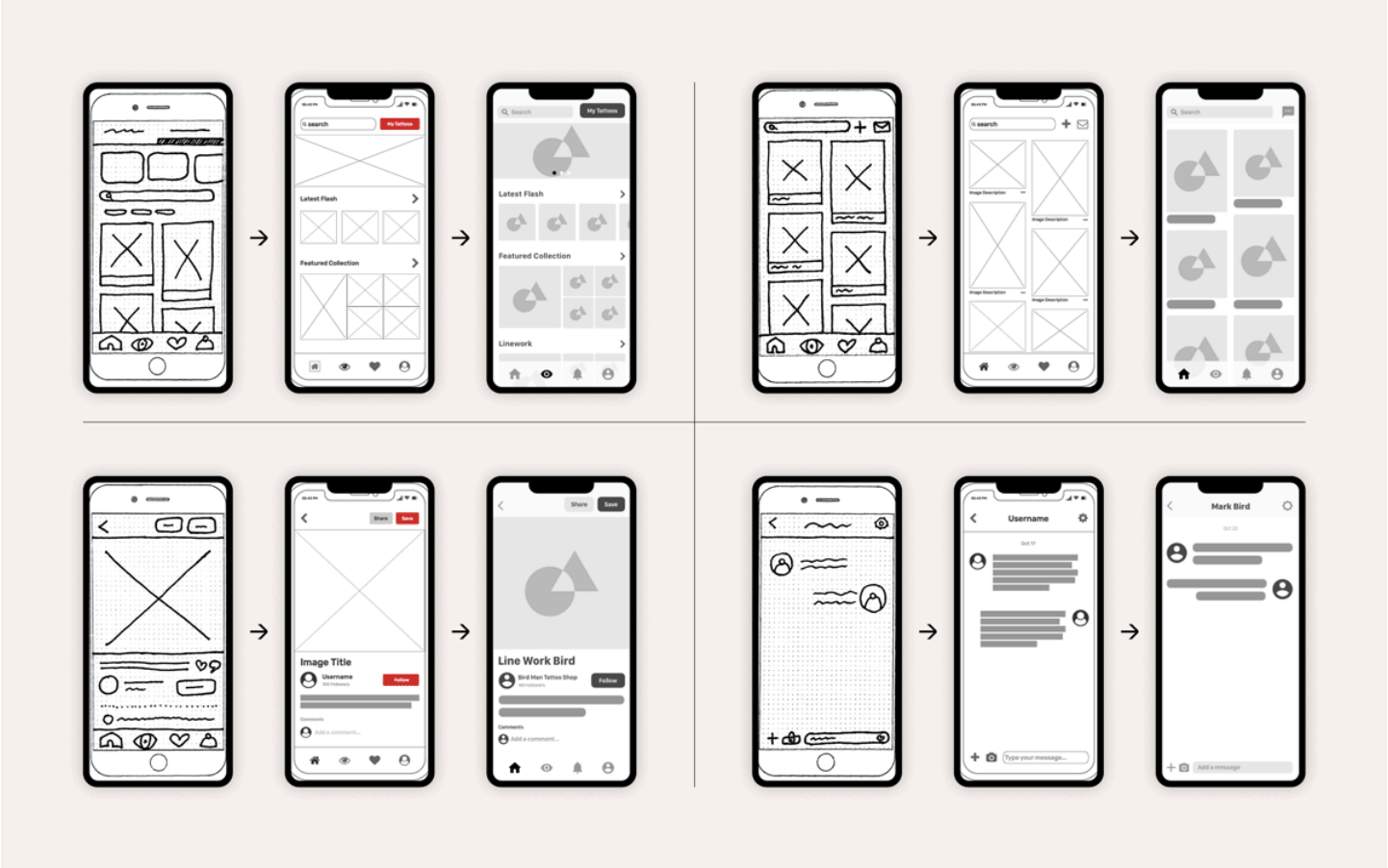


Examples of wireframe varieties

Monica Galvan shows us the transition from low-fidelity wireframe to high-fidelity wireframe, right through to the final UI for her Impossible project.



Down below, **Elvira Hellenpart's** wireframes for the VocabApp project, created as part of the CareerFoundry UX Fundamentals Program.



WebSite Structure

Standard WebSite Structure

Directories

In web development, the main folder where all a project's files are stored is called the project's root. This is different from your computer's root directory (like C:) or the main root of a web server (such as terminallearning.com). Each web project has its own root folder located within the server's root—for example, terminallearning.com/java could be the root for a Java tutorial site, while a related project could exist at terminallearning.com/java/runapp. If you use a hosting service, your website might live under something like www.hostname.com/yourUsername, with specific projects inside folders such as www.hostname.com/yourUsername/myproject. Essentially, every web project has its own root directory where all its related pages and files are kept.

Off the root we will see the following directories:
/images: where all of the image files are stored.

- *You usually don't store HTML files in the images directory because users don't visit this folder—it's only for storing image files.*
- *For websites with lots of images, you can create subfolders to stay organized, like images/patterns, images/patterns/photos, or images/userUploads.*

/css: where all of the external CSS files are stored

- Some developers name the CSS directory "/styles" instead.

- You wouldn't place HTML files there because users don't visit this folder—it's meant only for storing external CSS files.

/scripts: where all of the scripts (i.e. JavaScript) are stored

- *Some developers call the scripts folder "/js" because most scripts are written in JavaScript.*
- *You normally wouldn't store HTML files there since users don't access this folder—it's only for keeping script files.*

The "/content" folder is sometimes used to store a website's main HTML pages, except the main index page, which stays in the root folder.

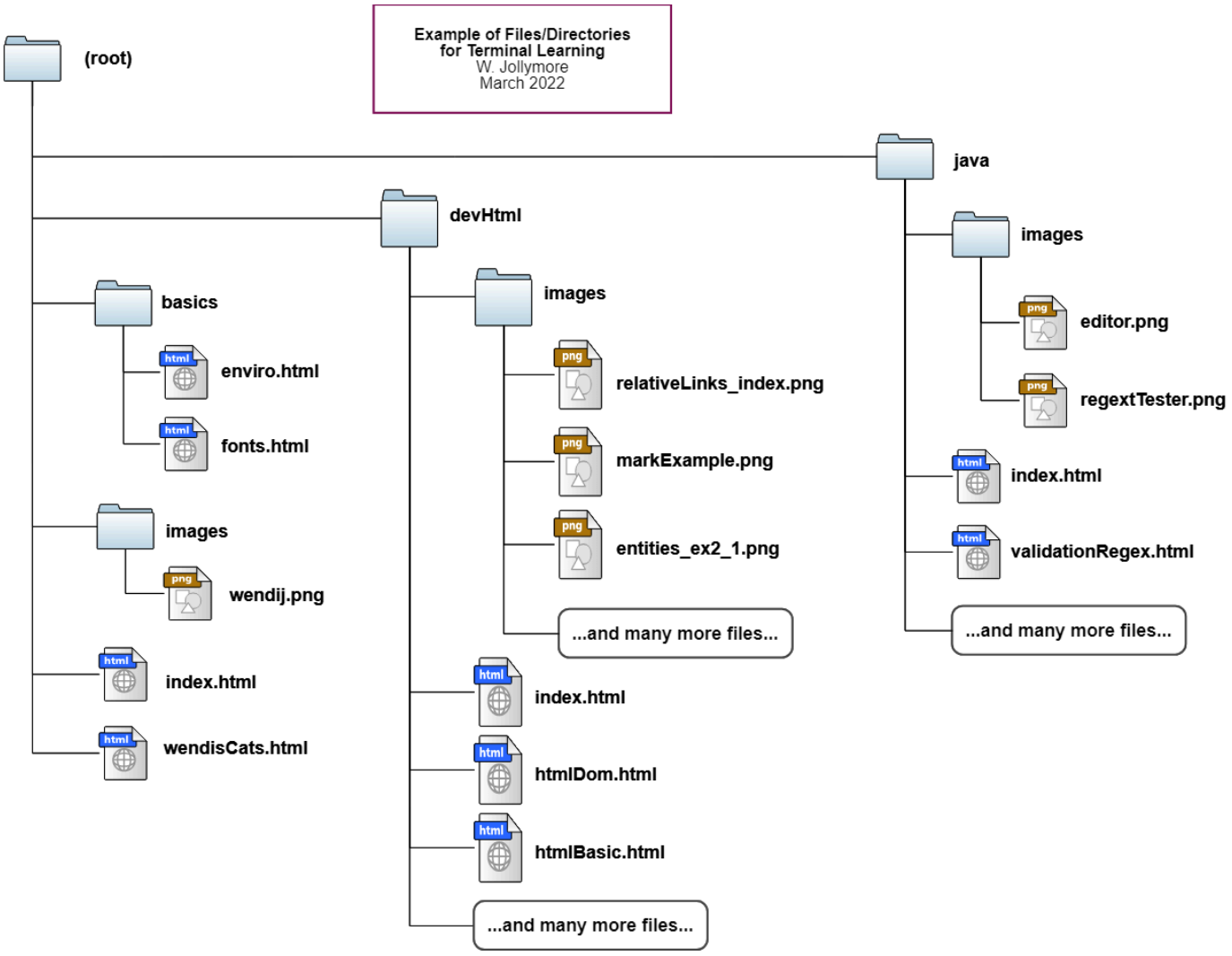
Large websites may have additional folders for different sections or topics, like /java, /devHtml, /javafx, or /devJs, to stay organized.

Files

Every directory in a web project, including the main root folder, should have an index.html file because web servers automatically look for it when a user visits a page without specifying a filename. For example, if someone goes to https://terminallearning.com/java, the server checks the /java folder for an index file like index.html, index.php, or main.html to load. If it doesn't find one, the server might show a 404 error or list the directory's contents (depending on settings). It's good practice for every folder with HTML files to include an index file so users don't get lost or face errors when exploring the site.

Large Web Sites

Large websites often have multiple folders to separate different types of content. For example, https://terminallearning.com leads to the site's main index.html page, even if you don't see "index.html" in the address bar—it's just loaded automatically. Because the tutorial site includes materials for several different courses, it's organized into subfolders to keep all the files and media well managed.



Some of the basic structure used on my tutorials web site.

The website's root folder contains several directories, but only a few are important to note here.

- The /basic folder holds shared content used across multiple tutorials, making it easy to find and update without duplicating files.
- The /css folder stores all the external stylesheets that control colors, fonts, and layouts for every tutorial page.
- The /images folder contains sitewide images, like icons used on different pages.
- The /scripts folder keeps all JavaScript files that many pages on the site use.

The website has separate folders for each tutorial, like devHtml, devCss, java, and Javafx, each containing multiple HTML lesson pages and an index.html file so visitors can automatically load the main page of that tutorial. Each tutorial folder also has its own /images subfolder to store images used only in that course, which keeps hundreds of images organized and easy to find. This structure—subfolders for content, images, scripts, and styles—is common for large websites. In smaller projects, you'll still need an index.html file in the root and in every main topic folder, plus folders for images and styles (scripts are optional for now).