

Week 3: Presentation layer - View

L3.1: Overview of MVC

Model-View-Controller

- Originates from *Smalltalk-80* - Object-oriented programming language
- It is developed by *Xerox PARC* - Research and development company

Design pattern

- Common software patterns
 - **Model**: Application object
 - **View**: Screen representation
 - **Controller**: How user interface reacts to user input
-

L3.2: Views

User Interface (UI)

- It can be:
 - Screen
 - Audio
 - Vibration (haptic)

User Interaction

- Determined by hardware constraints
- Different target devices possible
- User-Agent information useful to identify context
- It can be:
 - Keyboard / Mouse
 - Touchscreen
 - Spoken voice
 - Custom buttons

Types of Views

Fully Static

- No user interaction
- **Example:** A brochure website, A landing page

Partly Dynamic

- Some user interaction
- **Example::** Google web search, A news website

Mostly Dynamic

- Lots of user interaction
- **Example::** Social media app, E-commerce website

Output

- **HTML** - direct rendering
- Dynamic images
- JSON/XML - machine readable

View - any "representation" useful to another entity

L3:3 *User Interface Design*

- Design for interaction with user

Goals

Simple

- easy for user to understand and use

Efficient

- user achieves goal with minimal effort

Aesthetics

- Aesthetics is an important part of user interface design
- A well-designed UI should be visually appealing, but it should also be functional and easy to use.

Some key principles of UI aesthetics:

- Simplicity
- Consistency
- Visual Hierarchy
- Use of color
- Use of typography
- Use of graphics

Accessibility

- Accessibility is a critical aspect of user interface (UI) design, ensuring that people with disabilities can access and use digital products and services.


Some key considerations for designing accessible UIs include:

- Text Alternatives
- Screen Reader Compatibility
- Color Contrast

Systematic Process

- **Functionality requirements gathering** - what is needed?
- **User and Task analysis** - user preferences, task needs
- **Prototyping** - design wireframes and mockups
- **Testing** - usability testing, user feedback, accessibility

L3.4: Usability Heuristics

- Jakob Nielsen's 10 general principles for interaction design 

1. Visibility of system status

The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time.

Tips

- Communicate clearly to users what the system's state is — no action with consequences to users should be taken without informing them.
- Present feedback to the user as quickly as possible (ideally, immediately).

2. Match between system and the real world

The design should speak the users' language. Use words, phrases and concepts familiar to the user, rather than internal jargon. Follow real-world conventions, making information appear in a natural and logical order.

Tips

- Ensure that users can understand meaning without having to go look up a word's definition.
- Never assume your understanding of words or concepts will match that of your users.

3. User control and freedom

Users often perform actions by mistake. They need a clearly marked "emergency exit" to leave the unwanted action without havign to go through an extended process.

Tips

- Support Undo and Redo.
- Show a clear way to exit the current interaction, like a *Cancel button*.
- Make sure the exit is clearly labeled and discoverable.

4. Consistency and standards

Users should not have to wonder whether different words, situations or actions mean the same thing. Follow platform and industry conventions.

Tips

- Improve learnability by maintaining both types of consistency: internal and external.
- Maintain consistency within a single product or a family of products (internal consistency).
- Follow established industry conventions (external consistency).

5. Error prevention

Good error messages are important, but the best designs carefully prevent problems from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

Tips

- Prioritize your effort: Prevent high-cost errors first, then little frustrations.
- Prevent mistakes by removing memory burdens, supporting undo, and warning your users.

6. Recognition rather than recall

- Minimize the user's memory load by making elements, actions and options visible. The user should not have to remember information from one part of the interface to another. Information required to use the design should be visble.

Tips

- Let people recognize information in the interface, rather than forcing them to remember (“recall”) it.
- Reduce the information that users have to remember.

7. Flexibility and efficiency of use

Shortcuts - hidden from novice users - may speed up the interaction for the expert user so that the design can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

Tips

- Provide accelerators like keyboard shortcuts and touch gestures.
- Provide personalization by tailoring content and functionality for individual users.
- Allow for customization, so users can make selections about how they want the product to work.

8. Aesthetic and minimalist design

Interfaces should not contain information that is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility.

Tips

- Keep the content and visual design of UI focused on the essentials.
- Don't let unnecessary elements distract users from the information they really need.

9. Help users recognize, diagnose and recover from errors

Error messages should be expressed in plain language (no error codes), precisely indicate the problem and constructively suggest a solution.

Tips

- Use traditional error-message visuals, like bold, red text.
- Tell users what went wrong in language they will understand — avoid technical jargon.
- Offer users a solution, like a shortcut that can solve the error immediately.

10. Help and documentation

Its best if the system doesn't need any additional explanation. However, it may be necessary to provide documentation to help users understand how to complete their tasks.

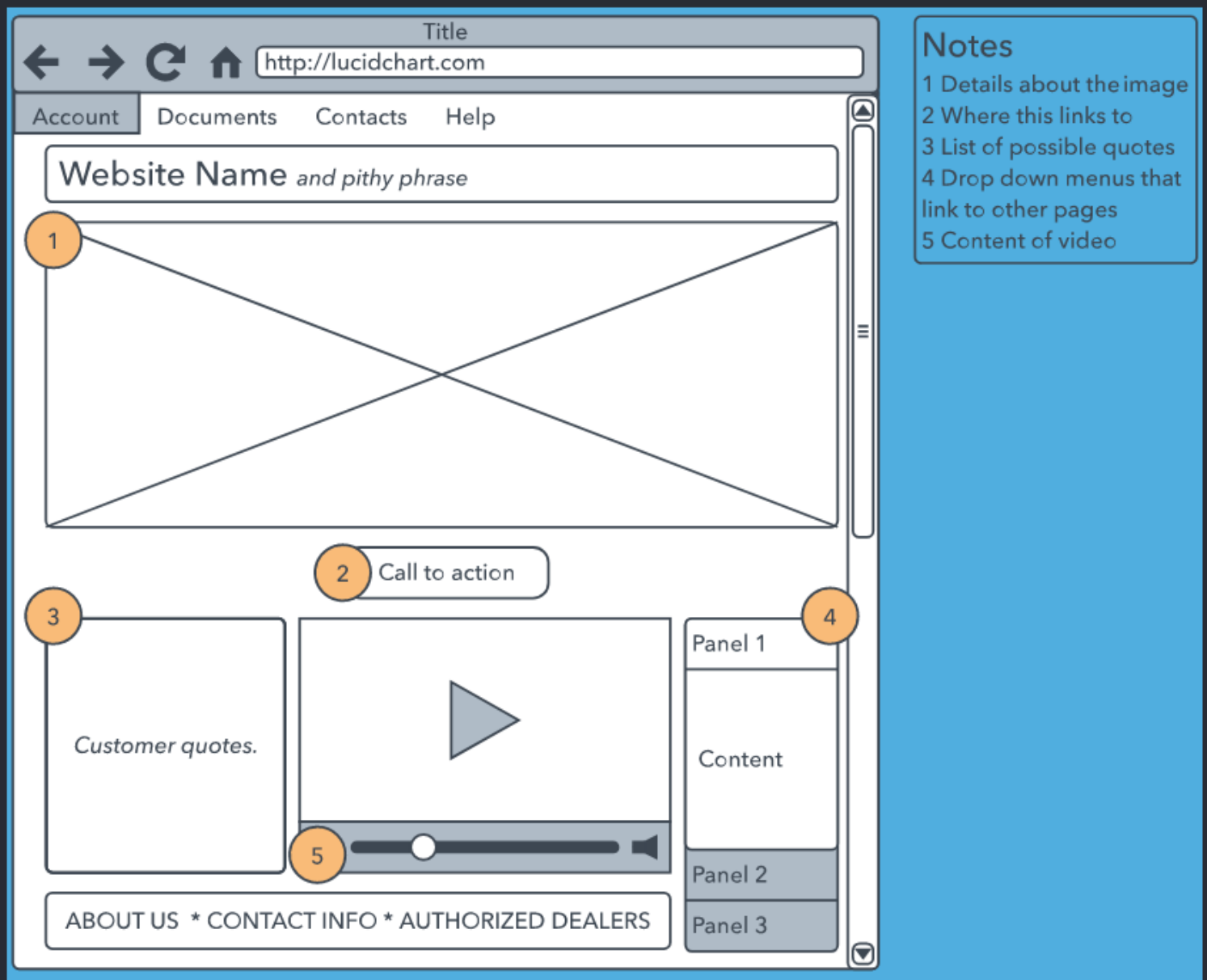
Tips

- Ensure that the help documentation is easy to search.
- Whenever possible, present the documentation in context right at the moment that the user requires it.

L3.5: Tools - Part 1

Wireframes

- A wireframe is a visual representation of a user interface, stripped of any visual design or branding elements.
- It consists:
 - Information design
 - Navigation design
 - User interface design



L3.6: Tools - Part 2

Programmatic HTML generation: PyHTML

- **PyHTML** is a Python library for generating HTML code.
- To use it, we need to download **pyhtml** package from PyPI.

```
$ pip install pyhtml
```

- A sample usage of **pyhtml**


```
import pyhtml as h

template = h.html(
    h.body(
        h.h1('Hello, World!'),
        h.p('This is a paragraph'),
        h.div(
            h.img(src='https://picsum.photos/200/300'),
            h.a('Click here', href='https://www.google.com')
        )
    )
)

print(template.render())
```

- This will generate the following HTML code:

```
<html>
  <body>
    <h1>Hello, World!</h1>
    <p>This is a paragraph</p>
    <div>
      
      <a href="https://www.google.com">Click here</a>
    </div>
  </body>
</html>
```

- We can do a lot of things, check out the [documentation](#) .

Templates

- Templates are a way to separate the HTML structure of a page from the content of the page.
- We can use placeholders / variables in the template to represent the content.
- Examples: Jinja2, Genshi, Mako, Django Templates, etc...

Jinja2

- Jinja2 is a modern and designer-friendly templating language for Python.
- It is used by Flask, Django, etc...
- To use it, we need to download **jinja2** package from PyPI.

```
$ pip install Jinja2
```

- A sample usage of `jinja2`

```
from jinja2 import Template
template = Template('Hello {{ name }}!')
print(template.render(name='Parampreet Singh'))
```

- This will generate the following code:


```
Hello Parampreet Singh!
```

- We can also use `jinja2` to render HTML code.

```
from jinja2 import Template
template = Template('''
<html>
  <body>
    <h1>Hello {{ name }}!</h1>
    <p>This is a paragraph</p>
    <div>
      
      <a href="https://www.google.com">Click here</a>
    </div>
    <ul>
      {% for i in range(1, 6) %}
        <li>List item {{ i }}</li>
      {% endfor %}
    </ul>
  </body>
</html>
''')
print(template.render(name='Parampreet Singh'))
```

- This will generate the following HTML code:

```
<html>
  <body>
    <h1>Hello Parampreet Singh!</h1>
    <p>This is a paragraph</p>
    <div>
      
      <a href="https://www.google.com">Click here</a>
    </div>
    <ul>
      <li>List item 1</li>
      <li>List item 2</li>
      <li>List item 3</li>
      <li>List item 4</li>
      <li>List item 5</li>
    </ul>
  </body>
</html>
```

- There's a lot we can do with `jinja2`, check out the [documentation](#) .

Remember

Templates can generate any output, not just `HTML`.

L3.7: *Accessibility*

- Various forms of disabilities:
 - Visual
 - Speech
 - Touch
 - Sensor-Motor

W3 Web Content Accessibility Standards

Standards

Web content

- HTML, images, scripts etc...
- Web content should be:
 - Perceivable
 - Operable
 - Understandable
 - Robust

User agents

- Browsers - desktop / mobile / speech-oriented etc...

Authoring tools

- Text editor, code editor, word processor etc...

Principles

Perceivable

- Provide **text alternatives** for non-text content.
- Provide **captions** for multimedia.
- Create content that can be **presented in different ways**.
- Make it easier for users to **see and hear** content.

Operable

- Make all functionality **keyboard accessible**.
- Give users **enough time** to read and use content.
- Do not use content that causes **seizures** or physical reactions.
- Help users **navigate and find content**.

Understandable

- Make text **readable and understandable**.
- Make content appear and operate in **predictable ways**.
- Help users **avoid and correct mistakes**.

Robust

- Maximize **compatibility** with current and future user agents, including **assistive technologies**.

Aesthetics

- Visual appearance.
- Simplicity preferred.

Evolution of google icons

