SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

User Interface Design & Implementation

User Interface Prototyping

Week 5 – Lecture 11

January – May Term, 2020

Today's Topics

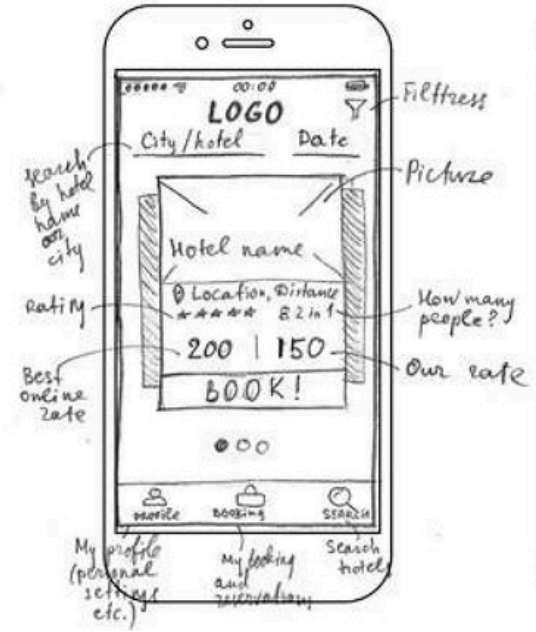
- User Interface Prototyping
- Prototyping Methods
  1. Paper Prototype
  2. Computer Prototype
  3. Interface Programming
  4. Web Development

Today's Topics

- **User Interface Prototyping**
- Prototyping Methods
  1. Paper Prototype
  2. Computer Prototype
  3. Interface Programming
  4. Web Development

What is a Prototype

- Prototype is a simulation of the final product. It's like an interactive mockup that can have any degree of fidelity.



Low-fidelity prototype

High-fidelity prototype

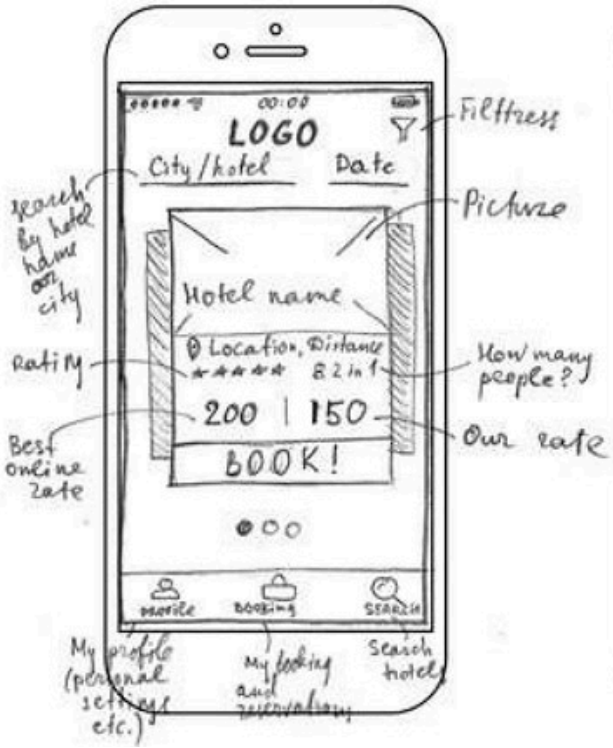
Real interface

Why Protype?

- Get feedback earlier, cheaper
- Experiment with alternatives
- Easier to change or throw away

Prototype Fidelity

- Low fidelity: omit details
- High fidelity: more like finished product



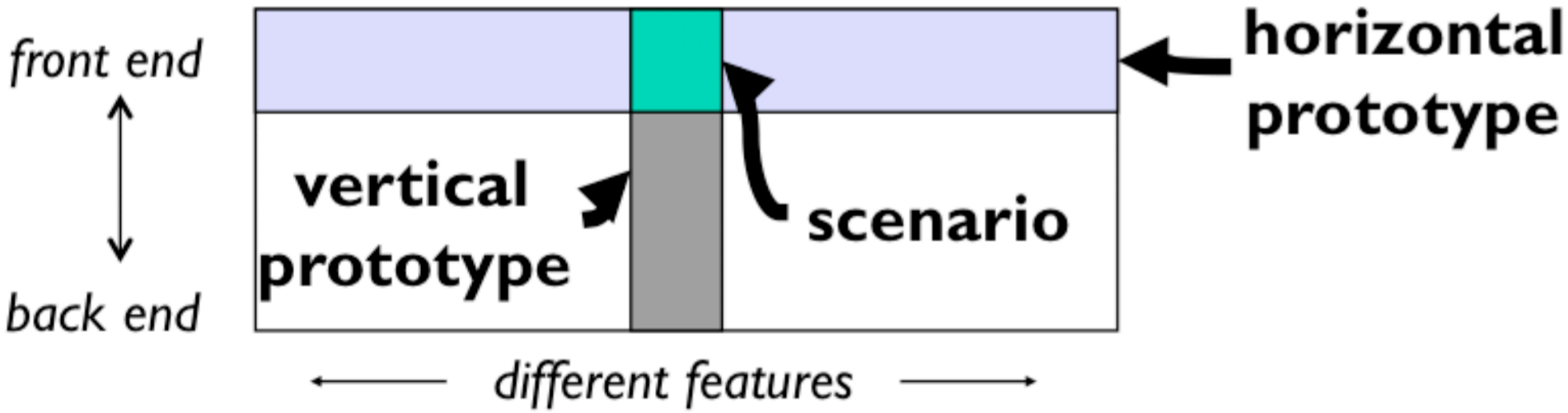
Low-fidelity prototype

High-fidelity prototype

Real interface

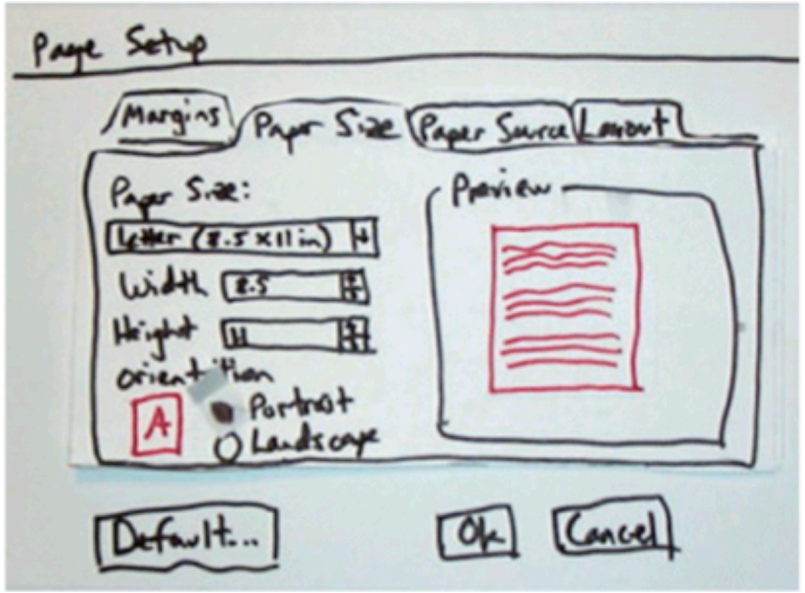
Fidelity is Multidimensional

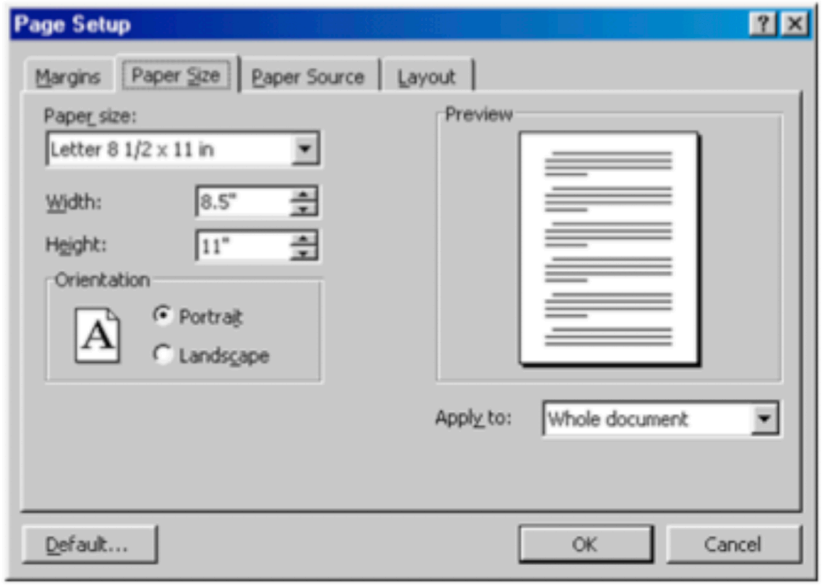
- Breadth: % of features covered
  - Only enough features for certain tasks
- Depth: degree of functionality
  - Limited choices, canned responses, no error handling



More Dimensions of Fidelity

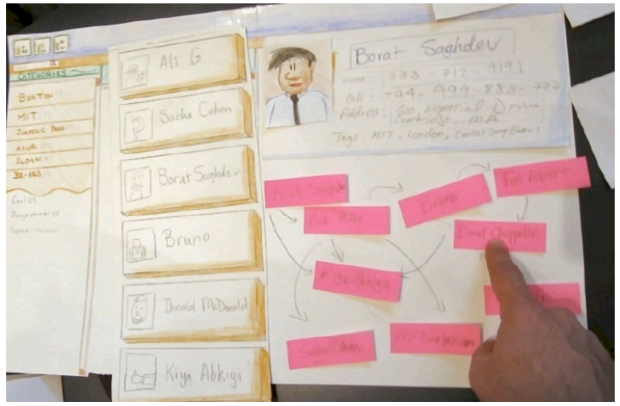
- Look: appearance, graphic design
  - Sketchy, hand-drawn
- Feel: input method
  - Pointing & writing feels very different from mouse & keyboard



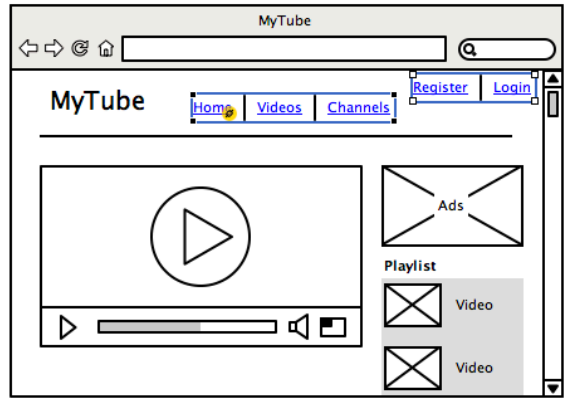


Today's Topics

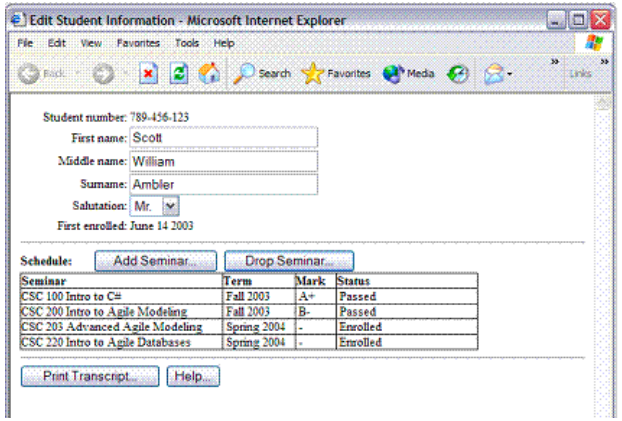
- User Interface Prototyping
- **Prototyping Methods**
  1. Paper Prototype
  2. Computer Prototype
  3. Interface Programming
  4. Web Development



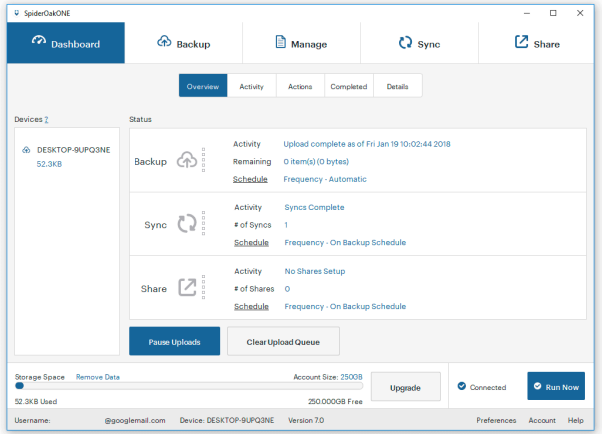
Paper prototype



Computer prototype



Web development



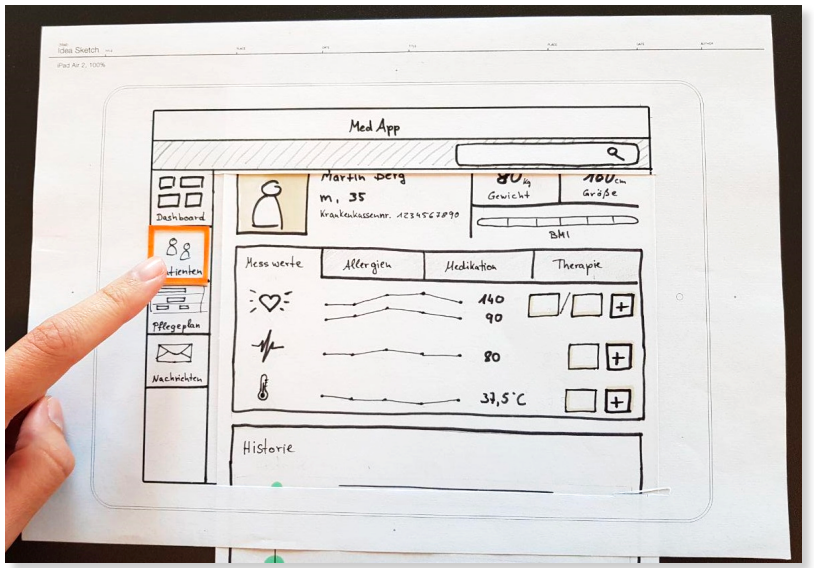
Interface programming

Today's Topics

- User Interface Prototyping
- Prototyping Methods
  1. **Paper Prototype**
  2. Computer Prototype
  3. Interface Programming
  4. Web Development

Paper Prototype

- Interactive paper mockup
  - Sketches of screen appearance
  - Paper pieces show windows, menus, dialog boxes
- Interaction is natural
  - Pointing with a finger = mouse click
  - Writing = typing
- A person simulates the computer’s operation
  - Putting down & picking up pieces
  - Writing responses on the ”screen”
  - Describing effects that are hard to show on paper



Why Paper Prototyping

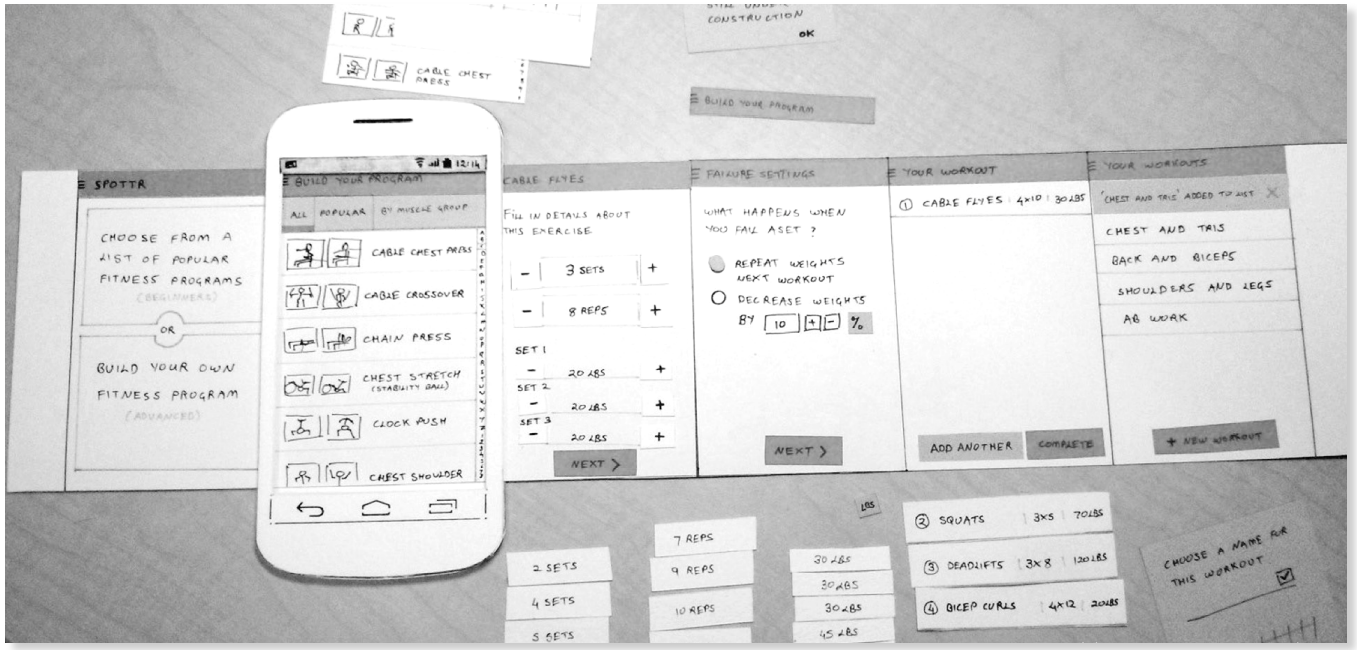
- Faster to build
  - Sketching is faster than programming
- Easier to change
  - Easy to make changes between user tests, or even during a user test
  - No code investment – everything will be thrown away (except the design)
- Focuses attention on big picture
  - Designer doesn’t waste time on details
  - Customer makes more creative suggestions, not nitpicking
- Non-programmers can help
  - Only kindergarten skills are required

Tools for Paper Prototyping

- White poster board
  - For background, window frame
- Big (unlined) index cards
  - For menus, windows, and dialog boxes
- Resticktable glue
  - For keeping pieces fixed
- White correction tape
  - For text fields, checkboxes, short messages
- Pen & markers, scissors, tape

Creating a Paper Prototype

- Gather materials and tools
- Identify the screens in your UI
  – consider use cases, inputs and outputs to the user
- Think about how to get from one screen to next

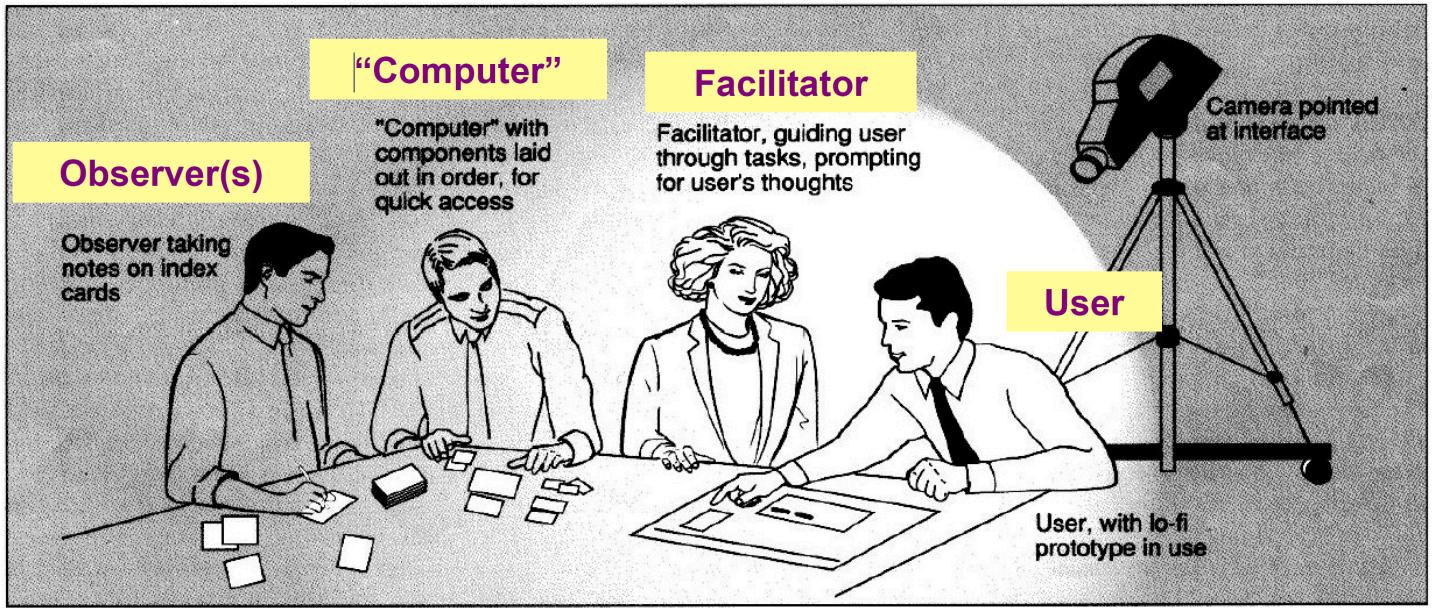


Tips for Good Paper Prototypes

- Make it larger than life
- Make it monochrome
- Replace tricky visual feedback with audible descriptions
- Tooltips, drag & drop, animation, progress bar
- Keep pieces organized
- Use folders & open envelopes
- Hand-sketched or computer-drawn

How to Test a Paper Prototype

- **“Computer”**: simulates prototype; doesn’t give any feedback that the computer wouldn’t
- **Facilitator**: presents interface and tasks to the user; encourages user to “think aloud” by asking questions; keeps user test from getting off track
- **Observer**: keeps mouth shut; takes copious notes



How to Test a Paper Prototype

Youtube Link: https://www.youtube.com/watch?v=y20E3qBmHpg

What You Can Learn from a Paper Prototype

- Conceptual model
  - Do users understand it?
- Functionality
  - Does it do what's needed? Missing features?
- Navigation & task flow
  - Can users find their way around?
  - Are information preconditions met?
- Terminology
  - Do users understand labels?
- Screen contents
  - What needs to go on the screen?

What You Can’t Learn

- Look: color, font, whitespace, etc.
- Feel: efficiency issues
- Response time
- Are small changes noticed
  - Even the tiniest change to a paper prototype is clearly visible to users
- Exploration vs. deliberation
  - Users are more deliberate with a paper prototype; they don’t explore or thrash as much

Today's Topics

- User Interface Prototyping
- Prototyping Methods
  1. Paper Prototype
  2. **Computer Prototype**
  3. Interface Programming
  4. Web Development

Computer Prototype

- Interactive software simulation
- High-fidelity in look & feel
- Low-fidelity in depth
  - Paper prototype had a human simulating the backend; computer prototype doesn't
  - Computer prototype may be horizontal: covers most features, but no backend

Computer Prototype

Real Interface

What You Can Learn from Computer Prototypes

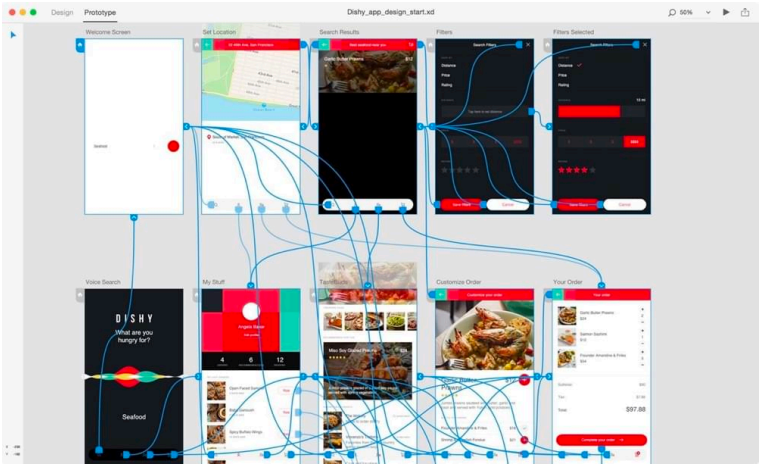
- Everything you learn from a paper prototype, plus:
- Screen layout
  - Is it clear, overwhelming, distracting, complicated?
  - Can users find important elements?
- Colors, fonts, icons, other elements
  - Well-chosen?
- Interactive feedback
  - Do users notice & respond to status bar messages, cursor changes, other feedback
- Efficiency issues
  - Controls big enough? Too close together? Scrolling list is too long?

Why Computer Prototyping?

- Faster than coding
- No debugging
- Easier to change or throw away
- Don’t let your UI toolkit do your graphic design

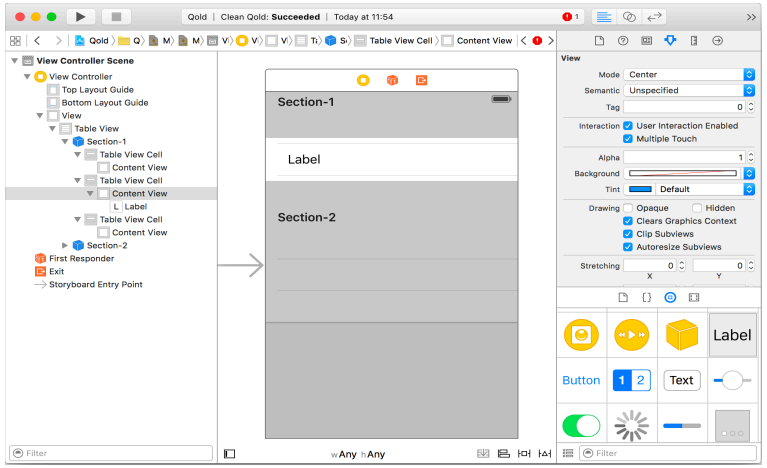
Computer Prototyping Techniques

1. Storyboard
   - Sequence of painted screenshots
   - Sometimes connected by hyperlinks ("hotspots")



Storyboard: Adobe XD

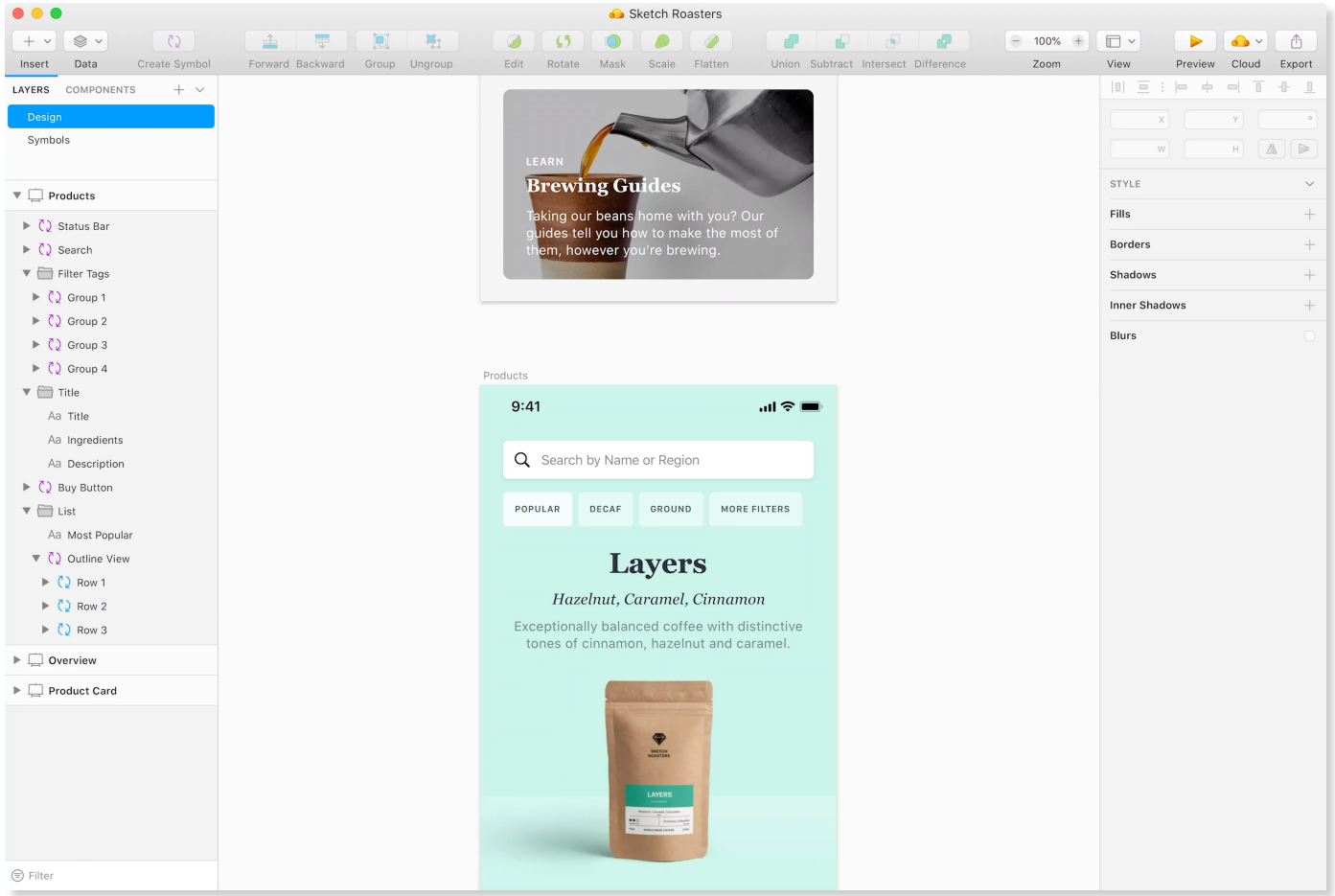
2. Form builder
   - Real windows assembled from a palette of widgets (e.g, buttons, text fields, labels)



Form builder: Xcode interface builder

#1 Storyboard: Tools

- Adobe XD
- Balsamiq
- Mockplus
- Sketch
- etc.



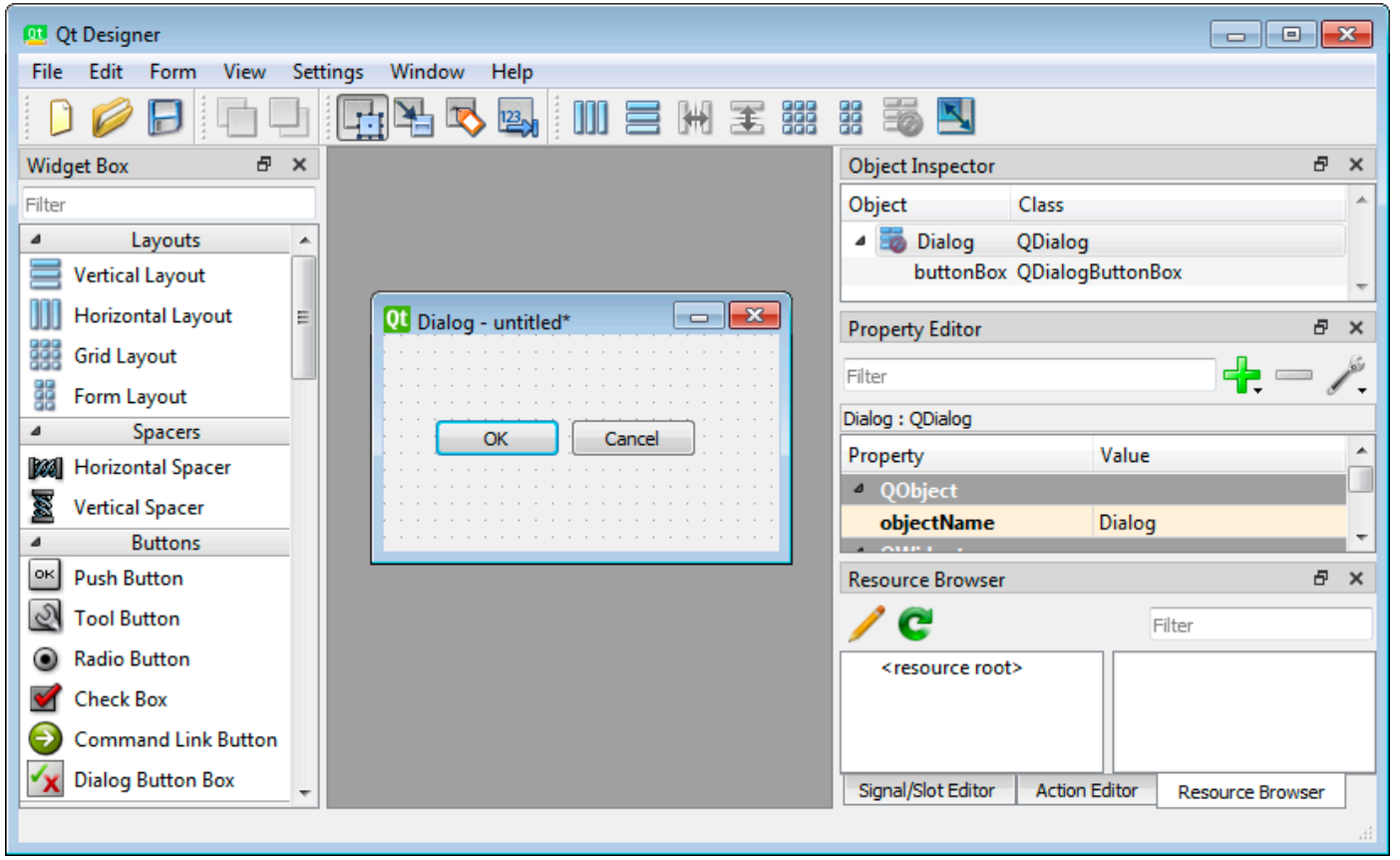
[Sketch demo]

#1 Storyboard: Pros & Cons

- Pros
  - You can draw anything
- Cons
  - No text entry
  - Widgets aren’t active
  - “Hunt for the hotspot”

#2 Form Builder: Tools

- Xcode Interface Builder
- Qt Designer
- Flex Builder
- Silverlight
- etc.



Qt Designer demo

#2 Form Builder: Pros & Cons

- Pros
  - Actual controls, not just pictures of them
  - Can hook in some backend if you need it
    - But then you won't want to throw it away
- Cons
  - Limits thinking to standard widgets
  - Less helpful for rich graphical interfaces

Today's Topics

- User Interface Prototyping
- Prototyping Methods
  1. Paper Prototype
  2. Computer Prototype
  3. **Interface Programming**
  4. Web Development

UI Software Programming

In software engineering, front end and back end are terms to describe the layers that make up a computer program or a website which are delineated based on how accessible they are to a user:

Front end: convert the underlying component into an interface (presentation layer)

Back end: handles business logic and data storage (data access layer)

The course project should focus on **Front End** design and implementation!
The **back end** should be **simulated** or **simplified** as much as possible.

Model-view-controller Pattern

Model–view–controller (MVC) is a software design pattern commonly used for developing user interfaces which divides the related program logic into three interconnected elements.

Model: manage the data, logic and rules

View: represent information such as a chart, or diagram.

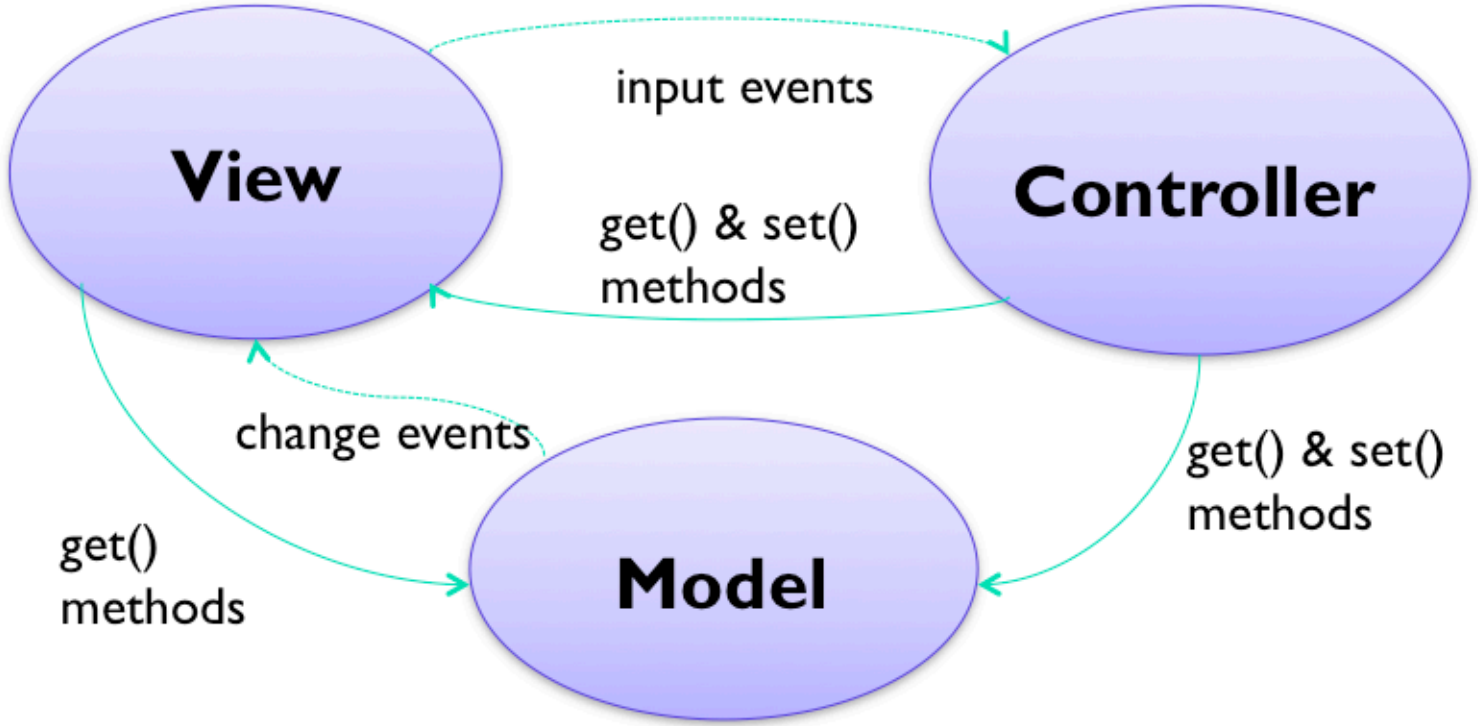
Controller: accept input; convert it to commands to the model or view

The purpose of MVC pattern is to
separate front end (View and Controller) **from the back end** (model).

Model-view-controller Pattern

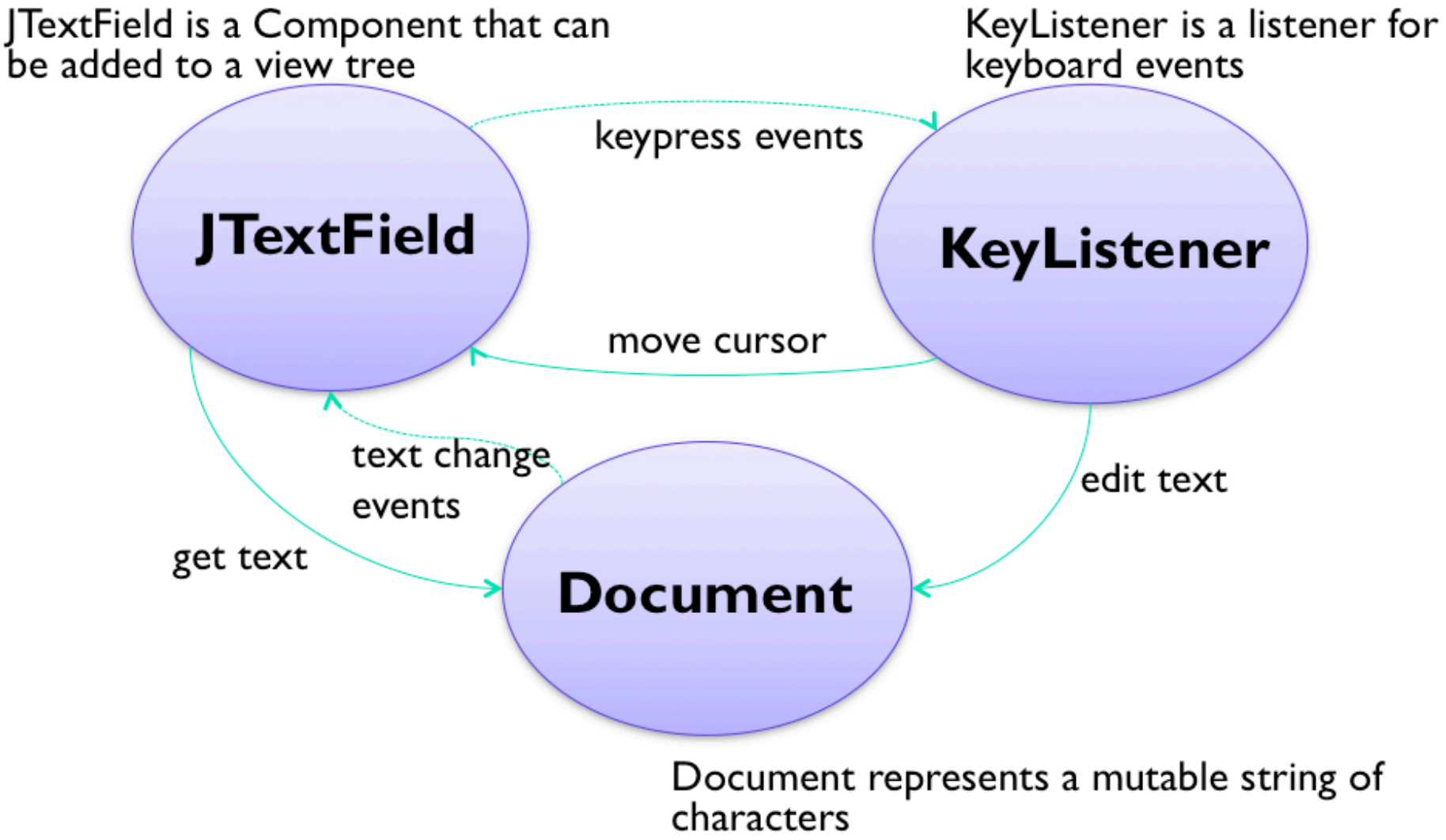
View handles output
- gets data from the model to display it
- listens for model changes and updates display

Controller handles input
- listens for input events on the view tree
- calls mutators on model or view



Model maintains application state
- implements state-changing behavior
- sends change events to views

A Small MVC Example: Textbox

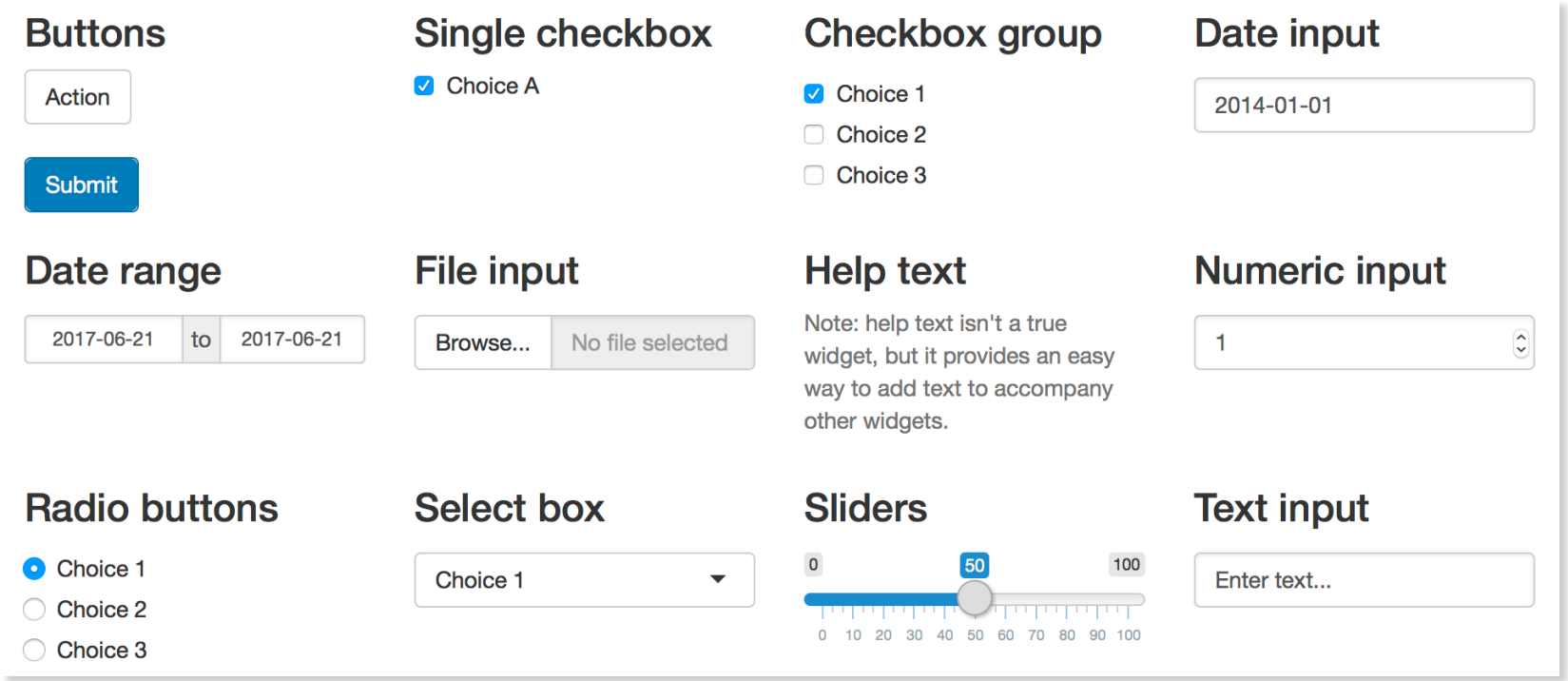


Advantages of Model-View-Controller

- Separation of responsibilities
  - Each module is responsible for just one feature
    - Model: data
    - View: output
    - Controller: input
- Decoupling
  - View and model are decoupled from each other, so they can be changed independently
  - Model can be reused with other views
  - Multiple views can simultaneously share the same model
  - Views can be reused for other models, as long as the model implements an interface

Widget: Tightly Coupled View & Controller

- MVC pattern has largely been superseded by an MV (Model-View) idea
- A widget is a reusable view object that manages both its output and input
  - Widgets are sometimes called components (Java, Flex) or controls (Windows)



Interface Programming

A graphical user interface (GUI) can be implemented by using a list of platform-independent **GUI library packages**.

In C, C++ [edit]

Name	Owner	Platforms	License
Chromium Embedded Framework	CEF Project Page	Linux, macOS, Microsoft Windows	Free: BSD
CEGUI	CEGUI team	Linux, macOS, Microsoft Windows	Free: MIT
Enlightenment Foundation Libraries (EFL)	Enlightenment.org	X11, Wayland, Microsoft Windows, macOS, DirectFB, Tizen	Free: BSD, LGPL, GPL
Fast Light Toolkit (FLTK)	Bill Spitzak, et al.	X11, Microsoft Windows, macOS	Free: LGPL
GTK+ formerly GIMP Toolkit	GNOME Foundation	Linux (X11, Wayland), Microsoft Windows, macOS, HTML5	Free: LGPL
IUP	Tecgraf, PUC-Rio	X11, Microsoft Windows	Free: MIT
JUCE	Roli Ltd.	X11, Linux[*clarification needed*], macOS, iOS, Android, Microsoft Windows	Mixed: GPL, proprietary
LiveCode	LiveCode, Ltd.	X11, macOS, Microsoft Windows	Proprietary
MKS Toolkit for Enterprise Developers formerly NuTCRACKER	DataFocus, Inc.	Microsoft Windows from X11 code[*clarification needed*]	Proprietary
Nana	Jinhao	Linux, Microsoft Windows	Free: Boost
Qt	Qt Project	Linux (X11, Wayland), OS/2, macOS, iOS, Android, Microsoft Windows	Mixed: LGPL, GPL, or proprietary
Ultimate++	Ultimate++	X11, PocketPC, WindowsCE, Microsoft Windows	Free: BSD-like
wxWidgets formerly wxWindows	wxWidgets team	X11, Wayland, OpenLook,[*clarification needed*] macOS, iOS, Microsoft Windows, OS/2	Free: wxWindows

Wikipedia: https://en.wikipedia.org/wiki/List_of_platform-independent_GUI_libraries

Interface Programming

A graphical user interface (GUI) can be implemented by using a list of platform-independent **GUI library packages**.

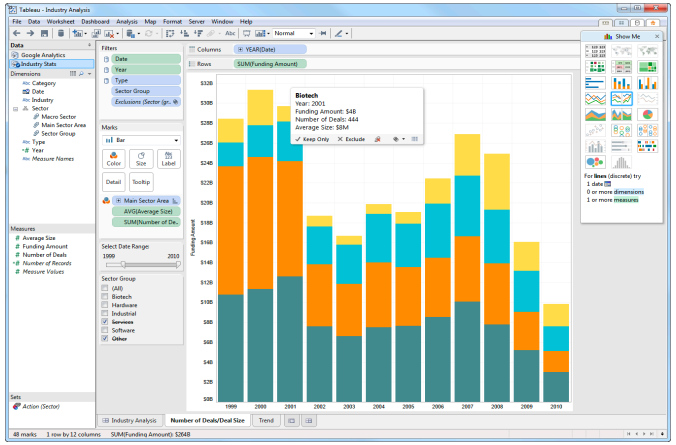
In other languages [edit]

Name	Owner	Programming Language	Platforms	License
Swing	Oracle Corporation	Java	Windows, Linux X11, macOS	Free: CDDL, GPL with linking exception
JavaFX	Oracle Corporation	Java	Windows, Linux X11, macOS, Android, iOS	Free: CDDL, GPL with linking exception
SWT	Eclipse Foundation	Java	Windows (Win32), Linux (GTK+), macOS (Cocoa)	Free: Eclipse
Apache Pivot	Apache Software Foundation	Java	Windows, macOS, Linux	Free: Apache
Xojo	Xojo, Inc.	Xojo	Windows, macOS, Linux (X11), iOS, web	Proprietary
Tcl/Tk	Open source	Tcl	Windows, OS/2, X11, OpenLook,[*clarification needed*] Mac, Android	Free: BSD-style
LCL, Lazarus	Open source	Free Pascal	Windows (Win32, Qt), Linux (GTK+, Qt), macOS (Qt, Carbon, Cocoa)	Free: GPL, LGPL
Delphi, FireMonkey	Embarcadero Technologies	Object Pascal	Windows, macOS, iOS, Android	Proprietary
VisualWorks	Cincom	Smalltalk	Windows, OS/2, Linux (X11), OpenLook,[*clarification needed*] Mac	Proprietary
Pharo	Pharo community	Smalltalk	Windows, Linux (X11), macOS	Free: MIT, part Apache 2.0
Mono, GTK#	Xamarin	C#	Windows, Linux (X11, Wayland), macOS	Free: MIT, LGPLv2, GPLv2 (dual license)
Kivy	Kivy	Python	Linux, Windows, macOS, Android, iOS	Free: MIT
WxPython		Python	Linux, Windows, macOS	Free: wxWindows
Unity	Unity Technologies	C#, JavaScript, Boo	Windows, X11, macOS, Android, iOS also features cross-platform Web player	Proprietary, based on open-source
Apache Flex Formerly Adobe Flex	Apache Software Foundation	ActionScript, Flash, Adobe AIR	Windows (x86, x64), macOS, Android (ARM, x86), iOS, Web (SWF)	Free: Apache
Flutter	Google	C, C++, Dart	Android, iOS (experimental: Web, Linux, Windows, macOS)	Free: New BSD License

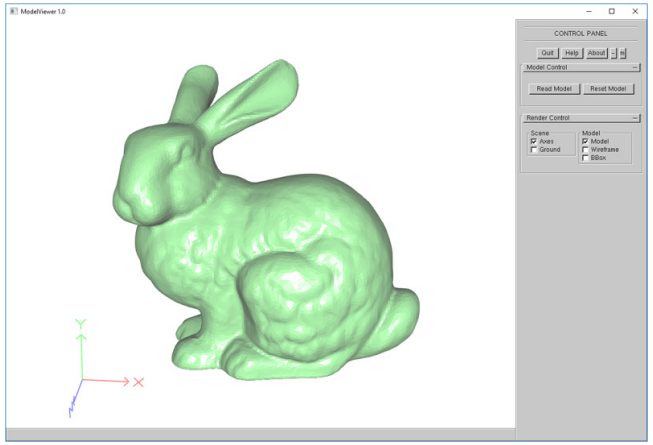
Wikipedia: https://en.wikipedia.org/wiki/List_of_platform-independent_GUI_libraries

Interface Programming Examples

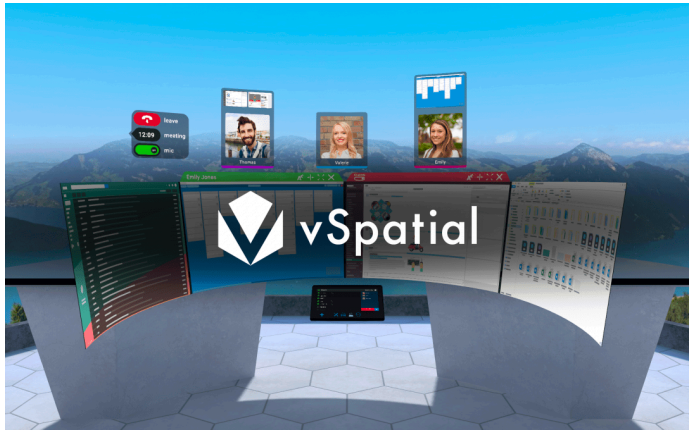
- 2D interface
- 3D interface
- VR interface
- AR interface
- etc.



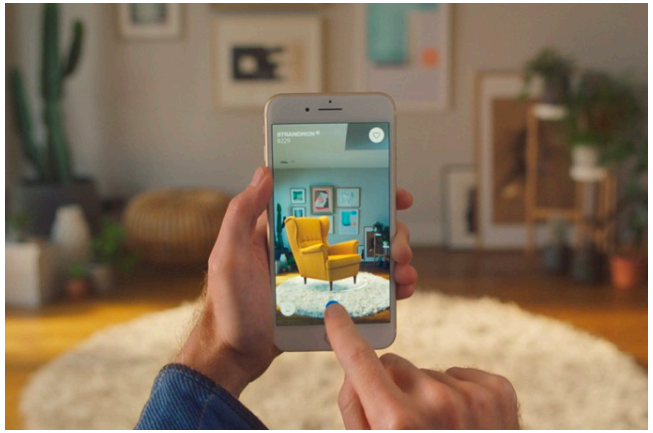
2D interface



3D interface



VR interface



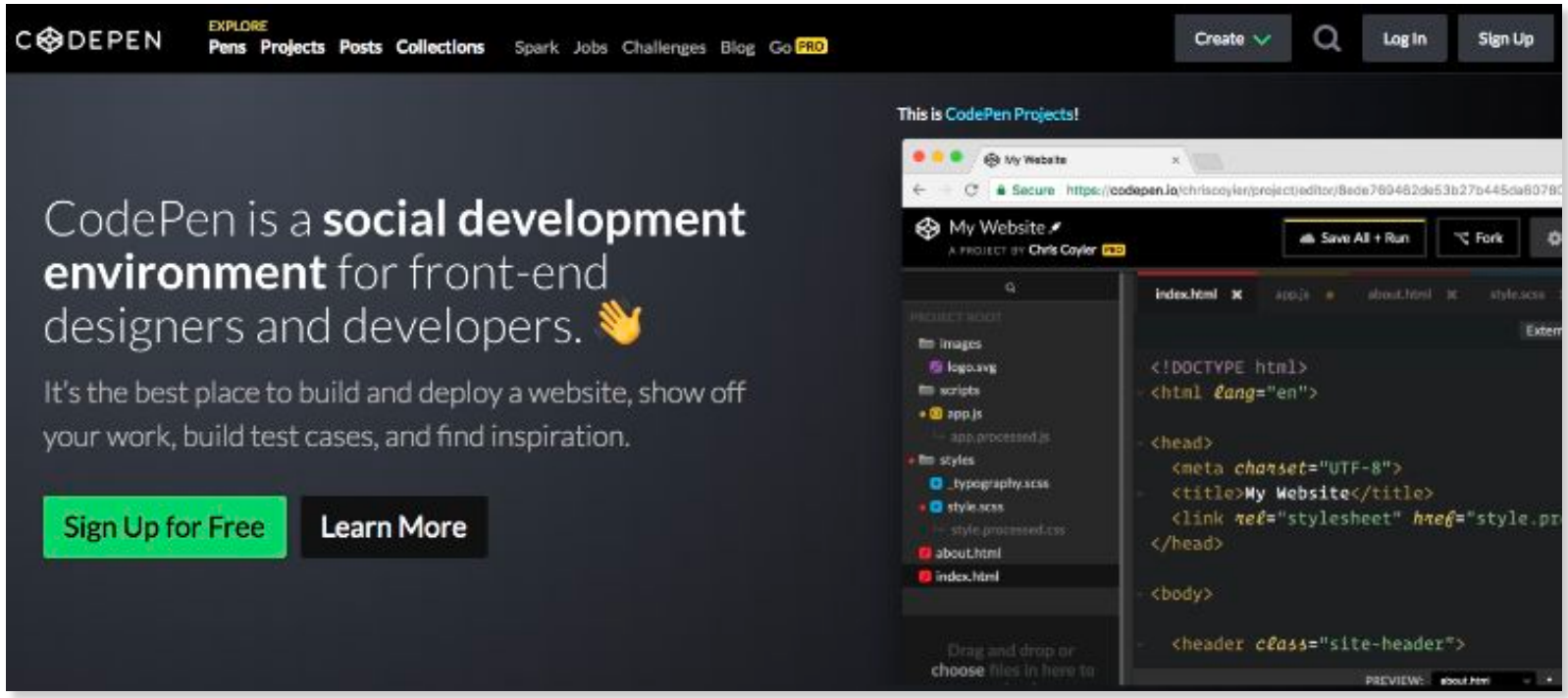
AR interface

Today's Topics

- User Interface Prototyping
- Prototyping Methods
  1. Paper Prototype
  2. Computer Prototype
  3. Interface Programming
  4. **Web Development**

Web Development

Web development is the work involved in **developing a website,** usually by **coding,** for the Internet (World Wide Web) or an intranet (a private network).



Web Development

Website	vs	**Software**
run on web servers		run on computers
require web browser		require installation
coding language (e.g., HTML)		coding language (e.g., C/C++)
require less expertise for development		need to understand various aspects of computers

Note: Similar to software, a website also has a front end and a back end.

Web Development Languages

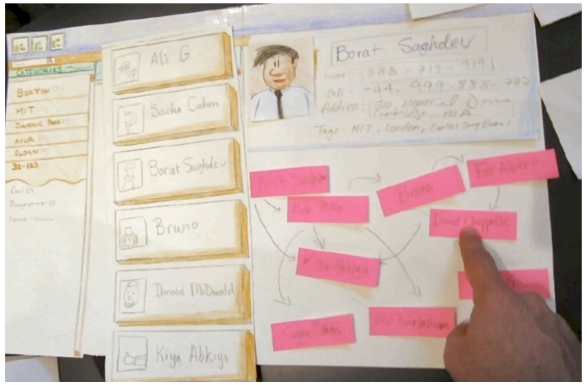
- HTML
- CSS
- Java
- JavaScript
- Python
- PHP
- etc.

Web Development Examples

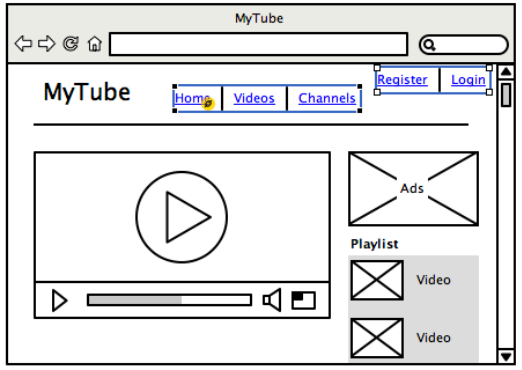
- Project website
- Web game
- News website
- Online booking
- Online exhibits
- Discussion board/forum
- Submission system
- etc.

Summary

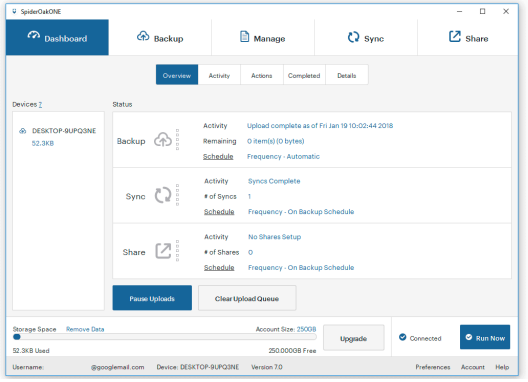
- Prototyping essentially involves building a small-scale version of a complex user interface in order to acquire critical knowledge required to build the interface.
- Several typical ways to prototype a user interface
  - Don't get attached to a prototype because it may need to be thrown away



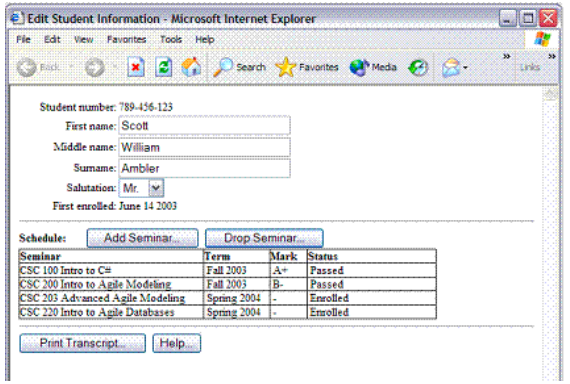
Paper prototype



Computer prototype



Interface programming



Web development

Coming Up

This Tuesday: **Visual Structure**

– Instructed by *Peng Song*

– 18:00 - 19:30 by e-learning