Building Web Applications

Mendel Rosenblum
Good web applications: Design + Implementation

Some Design Goals:

- Intuitive to use
  - Don't need to take a course or read a user manual
- Accomplish task accurately and rapidly
  - Provide needed information and functionality
- Users like the experience
  - Joy rather than pain when using the app

The hardest part of good web applications is the **design**
Outside the scope of this course (and instructor)!

Good user interface principles are encoded in the toolkits and style guides
Some guiding design principles for Web Apps

● Be consistent
  Cognitive load less for the user

● Provide context
  User shouldn't get lost in the app

● Be fast
  Don't make the user wait
Consistency: Style guides & design templates

● Web apps should have a **style guide** - Covers the look and feel of the app
  ○ Style - Color schemes, animation, icons, images, typography, writing
  ○ User interactions - Menu, buttons, pickers, dialog boxes, tables, lists, ...
  ○ Layout - Structure, toolbars, content, responsiveness

● Patterns - If you do something multiple places do it the same way
  ○ Aided by reusable implementation components
  ○ Error handling, navigation, notifications, etc.

● Design templates - Follow a familiar structure
  ○ Example: Master-detail template
Style Guide Example: Material Design from Google

● Used in Google apps (e.g. Android, web apps)
  ○ Influence by publishing (paper and ink) enhance with technology (3D look)
  ○ Focus on traditional print issues: grids, space, typography, scale, color, imagery
  ○ Heavy use of animation to convey action

● Dictates many aspect of design
  ○ Structure and layouts
  ○ User interface
  ○ Common patterns
Material Design Foundations

Environment - surfaces (e.g. paper), depth, and shadows
Layout - responsive layout grid, breakpoints, white space
Navigation - changing views: Lateral, Forward, Backward
Color - recommendations for colors that work well together
Typography - recommendations for point size, weight, spacing
Iconography - visual expressions (language independent)
Shape - use different shapes to direct attention, identify, communicate
Motion - show information (e.g. relationships), focus attention, fun
Interaction - map touch to actions
Communication - writing, formats, imagery, launch screen, onboarding
Front-end web frameworks

- **Popular example: Bootstrap**
  - CSS style sheets
    - Design templates
    - Grid layout system with responsive support (breakpoints, etc.)
    - Element styling
  - HTML components
    - Buttons, menus, toolbars, lists, table, forms, etc.
  - JavaScript
    - Modals, transitions, dropdowns, etc.
    - Originally jQuery based

- **ReactJS no-opinion. Popular: Material-UI**
  - CSS style sheets and components for implementing Material design spec
Example: Use Material Design for a Photo App

- Use a Master-Detail template layout
  - Users with Photos with Comments

- Classic layout:

<table>
<thead>
<tr>
<th>App Header (app context)</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Side Nav bar</td>
<td>User detail</td>
</tr>
<tr>
<td>User list</td>
<td>Photos with comments</td>
</tr>
</tbody>
</table>
Material-UI w/React - Use grid to layout app

```html
<Grid container spacing={24}>
  <Grid item xs={12}>
    <!-- Top bar across the top (all 12 col on xs or bigger) -->
    <TopBar />
  </Grid>
  <Grid item xs={4}>
    <!-- Row with List (4 col) & either Detail or Photo (8 col) -->
    <UserList />
  </Grid>
  <Grid item sm={8}>
    <!-- 8 columns wide

    or

    or

    8 columns wide
  </Grid>
</Grid>
```
Use grid to layout app

```html
<Grid container spacing={24}>
  <Grid item xs={12}>
    <TopBar /> ...
  </Grid>
  <Grid item xs={4}>
    <UserList ...>
  </Grid>
  <Grid item xs={4}>
    <UserDetail or <UserPhotos>
  </Grid>
</Grid>
```
Much useful functionality available for our app

Modals: Menu, Popover, Dialogs, Selects, SnackBars

Navigation: Tabs, Bottom Navigation, Drawers

Context tracking: AppBar, Stepper, Progress

Paper

Autocomplete

Tooltips

Badges
Deep linking support - React Route

To support bookmarking and sharing we can use React Route to load the views

The `content div` can be the React Route `Switch`

```
<Switch>
    <Route path="/users/:userId" component={UserDetail} />
    <Route path="/photos/:userId" component={UserPhotos}  />
    <Route path="/users" component={UserList}  />
</Switch>
```

The `UserList sidebar` can just use links to view

```
<Link to="/photos/57231f1a30e4351f4e9f4bd8">
    Photos of User Ellen Ripley
</Link>
```
Responsive Design support

- Uses CSS flexbox - Relative sizing handles changes (flex attribute)
  
  Grid ... -- Smaller widths will have smaller content area

- Use CSS breakpoints to handle big differences
Material Breakpoint sizes: xs, sm, md, lg, xl
Material UI Breakpoints

Uses a simplified model based on screen width:

- xs, extra-small: 0px or larger
- sm, small: 600px or larger
- md, medium: 960px or larger
- lg, large: 1280px or larger
- xl, extra-large: 1920px or larger
Material UI Responsive Support

- Grid Component - Grid takes xs/sm/md/lg/xl= columns properties
  
  ```jsx
  <Grid item xs={12} md={6} xl={3}> ...
  ```

- Hidden Component - Conditional rendering (xsUp, lgDown)
  
  ```jsx
  <Hidden mdUp>
    <Paper>This is a paper component except on md and bigger display</Paper>
  </Hidden>
  ```

- use MediaQuery React interface to @media
  
  ```jsx
  const theme = useTheme();
  const matches = useMediaQuery(theme.breakpoints.up('md'));
  if (matches) ... 
  ```
Accessibility

● Accessible Rich Internet Applications (ARIA)

● Provide text alternatives for any non-text content
  ○ Add text descriptions for things that need it
    `<a aria-label="Photo of user {{user.name}}" href=...`
    `<img aria-label="{{photo.description}}"`

● Provide alternatives for time-based media
  ○ Transcripts, sub-titles, etc.

● Work zoomed in, avoid quick timeouts, high contrast for foreground/background, work all keyboard and without keyboard, compatibility with assistive technologies, use simple sentences, etc.
Internationalization (I18N)

- Users want different: text, dates, numbers, currencies, and graphics
- Ultimately need a level of indirection. Consider: <h1>Getting Started</h1>
- Example: react-i18next: Look up translation by key
  - Hello <strong title="this is your name">{name}</strong>, you have {count} unread message(s).
  - <Trans i18nKey="userMessagesUnread" count={count}>
    Hello <strong title={t('nameTitle')}>{{name}}</strong>, you have {{count}} unread message.
  </Trans>
- Skip applying to user generated content
Testing the web app

- **Unit testing**
  - Each test targets a particular component and verifies it does what it claims it does
  - Requires mock components for the pieces that component interacts with
  - Example: Load an angular component (controller, directive, etc.) and run tests against it
    - Need to mock everything these touch (DOM, angular services, etc.)

- **End-to-End (e2e) testing**
  - Run tests against the real web application
  - Scripting interface into browser used to drive web application
  - Example: Fire up app in a browser and programmatically interact with it.
    - WebDriver interface in browsers useful for this

- **Metric: Test Coverage**
  - Does every line of code have a test?