Ten usability heuristics

Jakob Nielsen

Ten usability heuristics

1. Match between system and the real world
2. Consistency and standards
3. Visibility of system status
4. User control and freedom
5. Error prevention
6. Help users recognize, diagnose, and recover from errors
7. Recognition rather than recall
8. Flexibility and efficiency of use
9. Aesthetic and minimalist design
10. Help and documentation

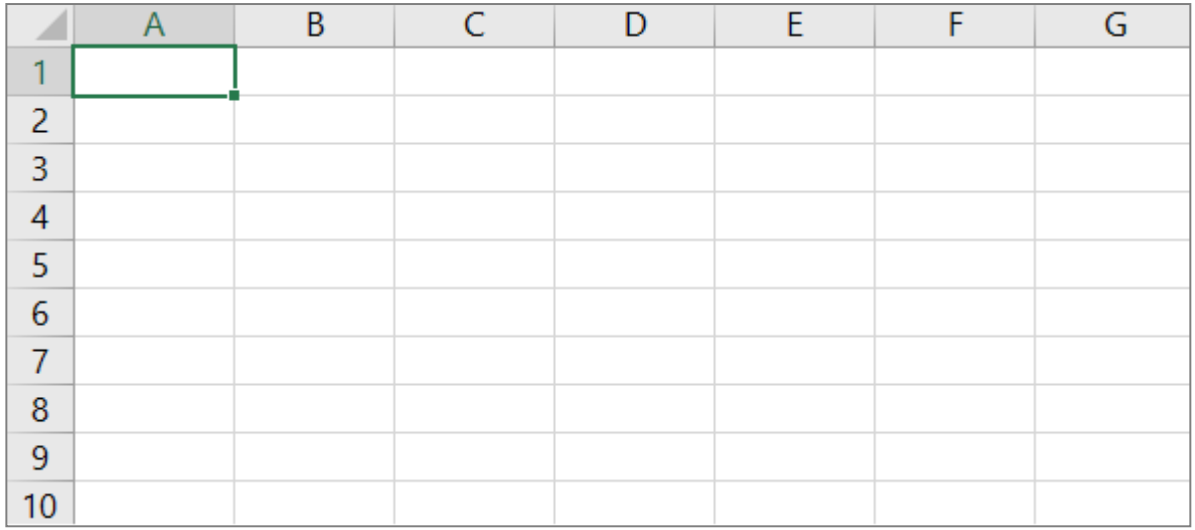
#1: Match between system and the real world

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms.

#1: Match between system and the real world

Computer science language: `[0][0], [0][2]`

Excel speaks users' language: `[A][1], [B][3]`



#1: Match between system and the real world

#2: Consistency and standards

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

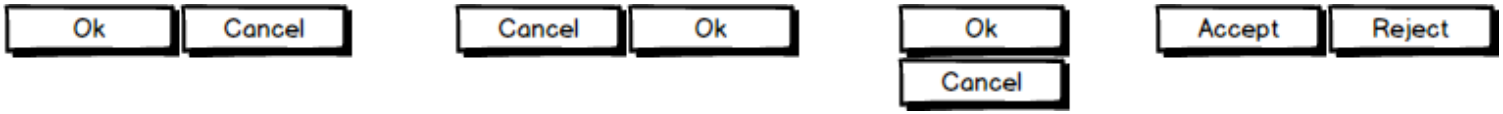
#2: Consistency and standards

Principle of least surprise

- Similar things should look and act similar
- Different things should look different
- Users should not have to wonder whether different words, situations, or actions mean the same thing (follow platform conventions)

Consistent language and graphics

- Same visual appearance across the system
- Same information/controls in same location on all windows



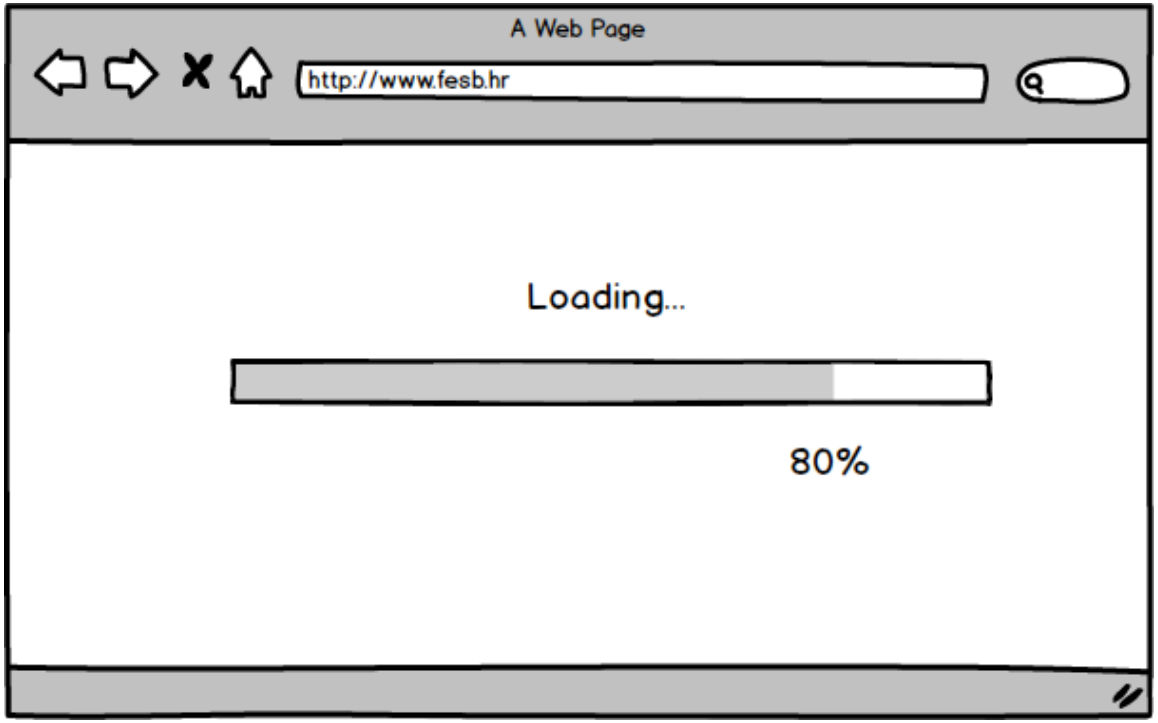
Consistent effects

- Commands, actions have same effect in equivalent situations

#3: Visibility of system status

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

#3: Visibility of system status



#3: Visibility of system status

Response times - important limits

- **< 0.1 second** - seems instantaneous
- **0.1-1 seconds -** user notices, but no feedback needed
- **1-5 seconds** - display busy cursor
- **> 1-5 seconds** - display progress bar

Response Times: The 3 Important Limits, [J. Nielsen'14]

#4: User control and freedom

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

#4: User control and freedom

Users don’t like to feel trapped by the computer

- Should offer an easy way out of as many situations as possible
- Provide clearly marked exits

Strategies:

- Cancel button
- Universal undo
- Interrupt (especially for lengthy operations)
- Quit (for leaving the program at any time)
- Defaults (for restoring default properites)

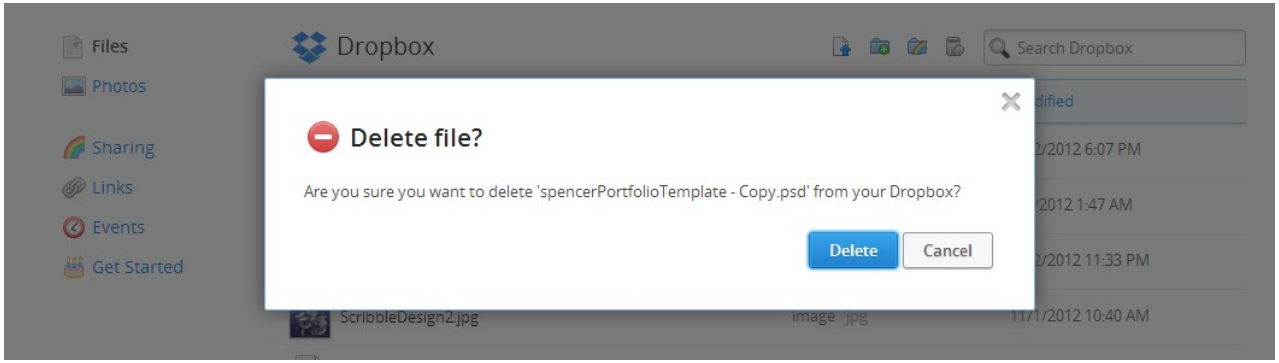
#5: Error prevention

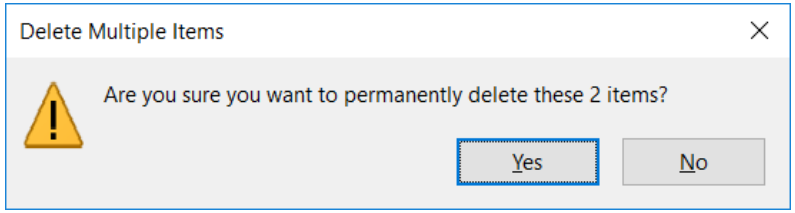
Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

#5: Error prevention

Design for errors

- Double-check with users

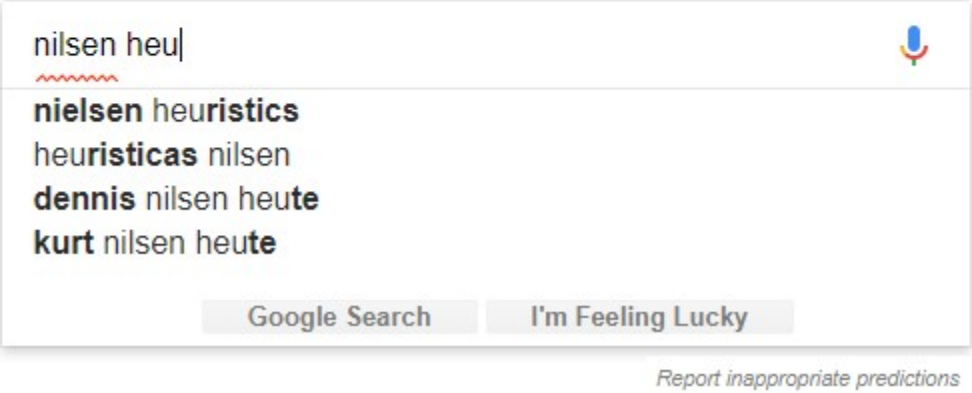




#5: Error prevention

Design for errors

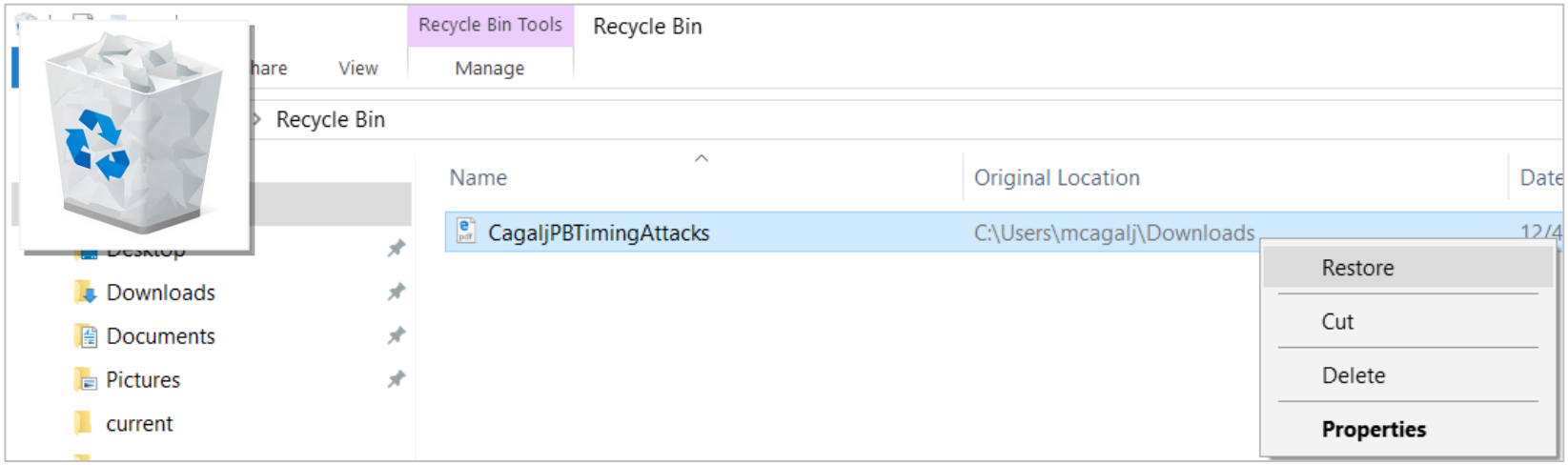
- Remove memory burdens



#5: Error prevention

Design for errors

- Support Undo and Re

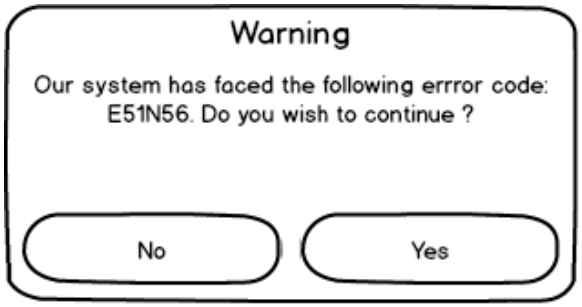


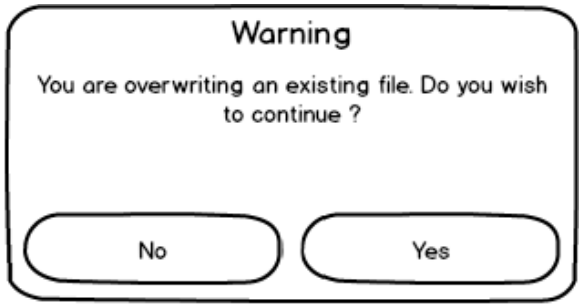
#6: Help users recognize, diagnose, and recover from errors

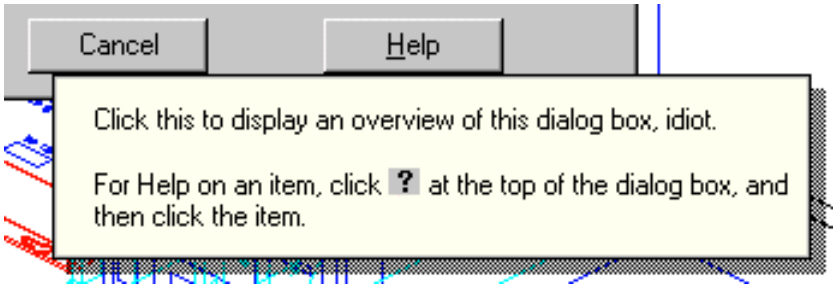
Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

#6: Help users recognize, diagnose, and recover from errors

Deal with errors in a positive manner (be polite, speak human-readable language)

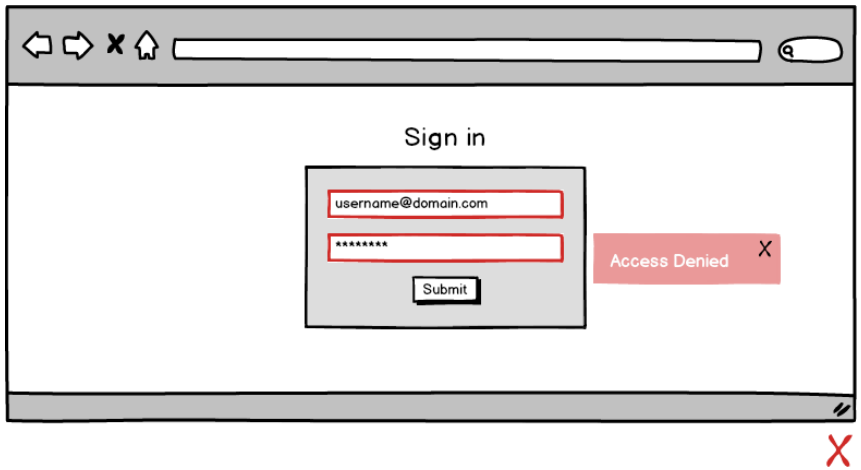


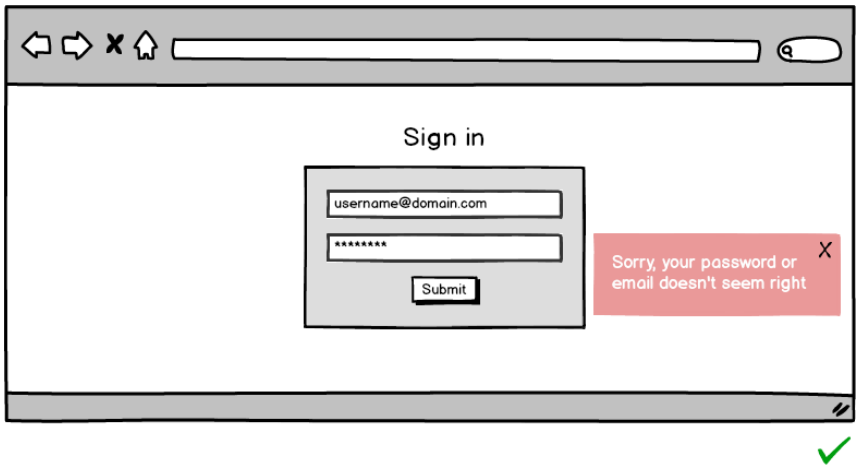




#6: Help users recognize, diagnose, and recover from errors

Deal with errors in a positive manner (be polite, speak human-readable language)





#7: Recognition rather than recall

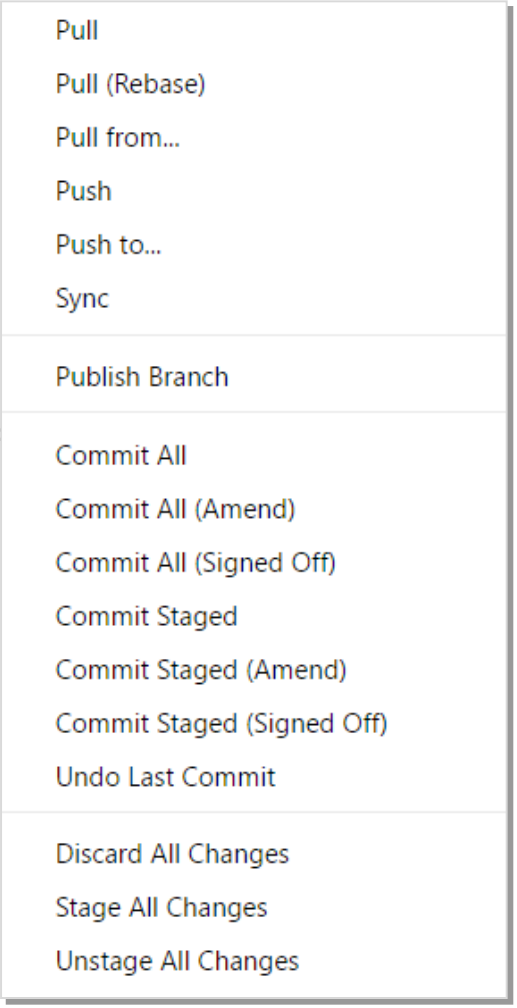
Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

#7: Recognition rather than recall

Short-term memory can hold around 5-7 elements

- CLI vs GUI

```
C:\> git add -A -- c:\Users\mcagalj\...\index.js
C:\> git commit --quiet --allow-empty-message --f
C:\> git status -z -u
C:\> git symbolic-ref --short HEAD
C:\> git remote --verbose
C:\> git config --get commit.template
C:\> git push
```



#7: Recognition rather than recall

Nav.jsx (Working Tree) - web-page - Visual Studio Code

File Edit Selection View Go Debug Tasks Help

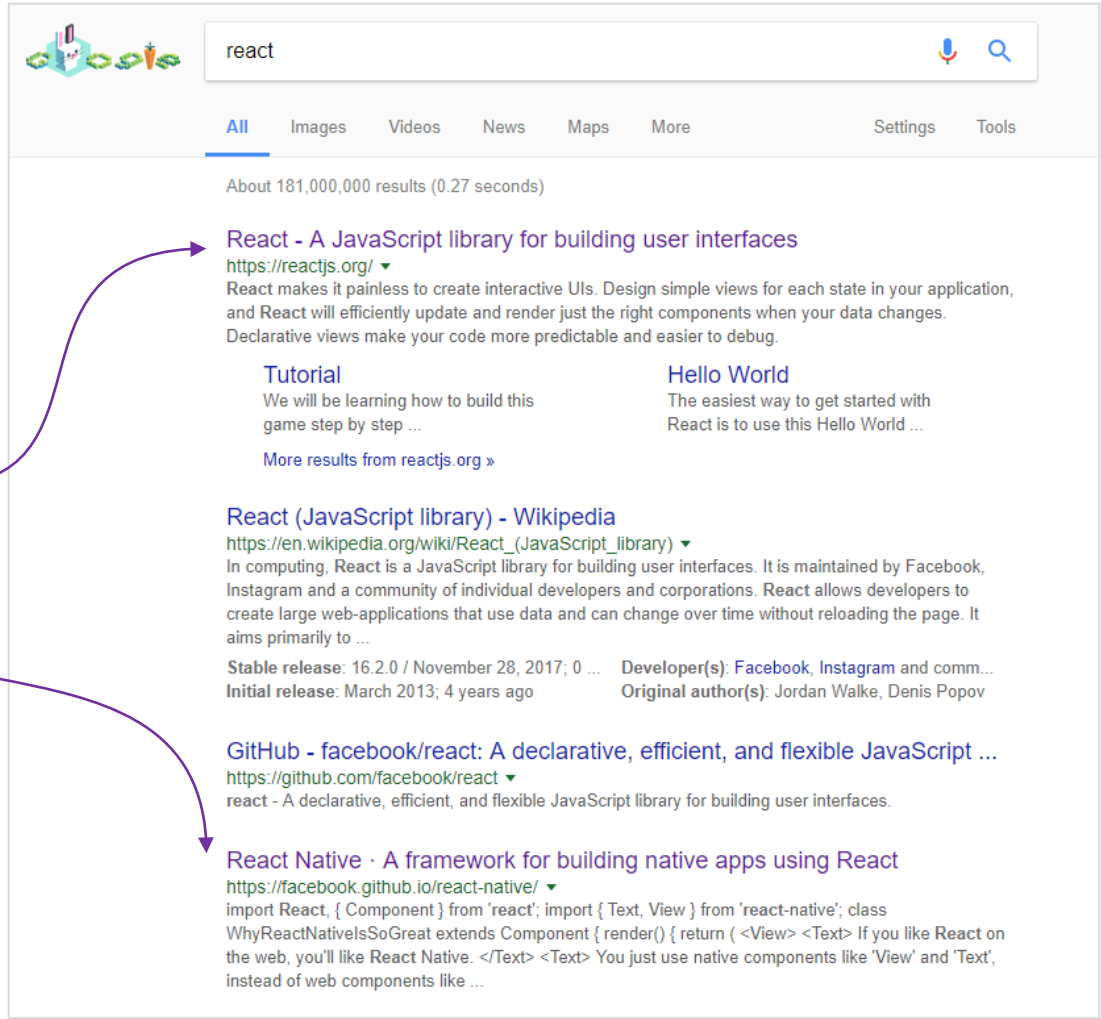
index.js | Nav.jsx (Working Tree)

```
1   import React, { Component } from 'react'
2   import classnames from 'classnames'
3   import 'styles/components/basic/Nav.css'
4   import { NavLink } from 'react-router-dom'
5
6   import Hamburger from 'components/basic/Hamburger.jsx'
7   import LogoutButton from 'components/basic/LogoutButton.jsx'
8
9   class Nav extends Component {
10      constructor(props) {
11          super(props)
12      }
13
14      state = {
15          isMenuOpen: false
```

```
1   import React, { Component } from 'react'
2   import classnames from 'classnames'
3   import 'styles/components/basic/Nav.css'
4   import { NavLink } from 'react-router-dom'
5
6   import Hamburger from 'components/basic/Hamburger.jsx'
7   import LogoutButton from 'components/basic/LogoutButton.jsx'
8
9   class Nav extends Component {
10      constructor(props) {
11          super(props)
12 +        this.linksThatFit = this.props.links
13 +        this.linksRest = []
14 +        this.maxWidth = 0
15 +
16 +        this.windowSize = 0,
17 +        this.windowResized = 0
18      }
19
20      state = {
21 +        isMenuOpen: false,
22 +        isRestOpen: false
23 +    }
24 +
25 +    _processNavLinks() {
26 +        const guardWidth = 200;
27 +
28 +        let maxWidth = this.containerElement.getBoundingClientRec
29 +        // Mobile devices often emit 'resize' when scrolling.
30 +        // We react only on changes in the width.
```

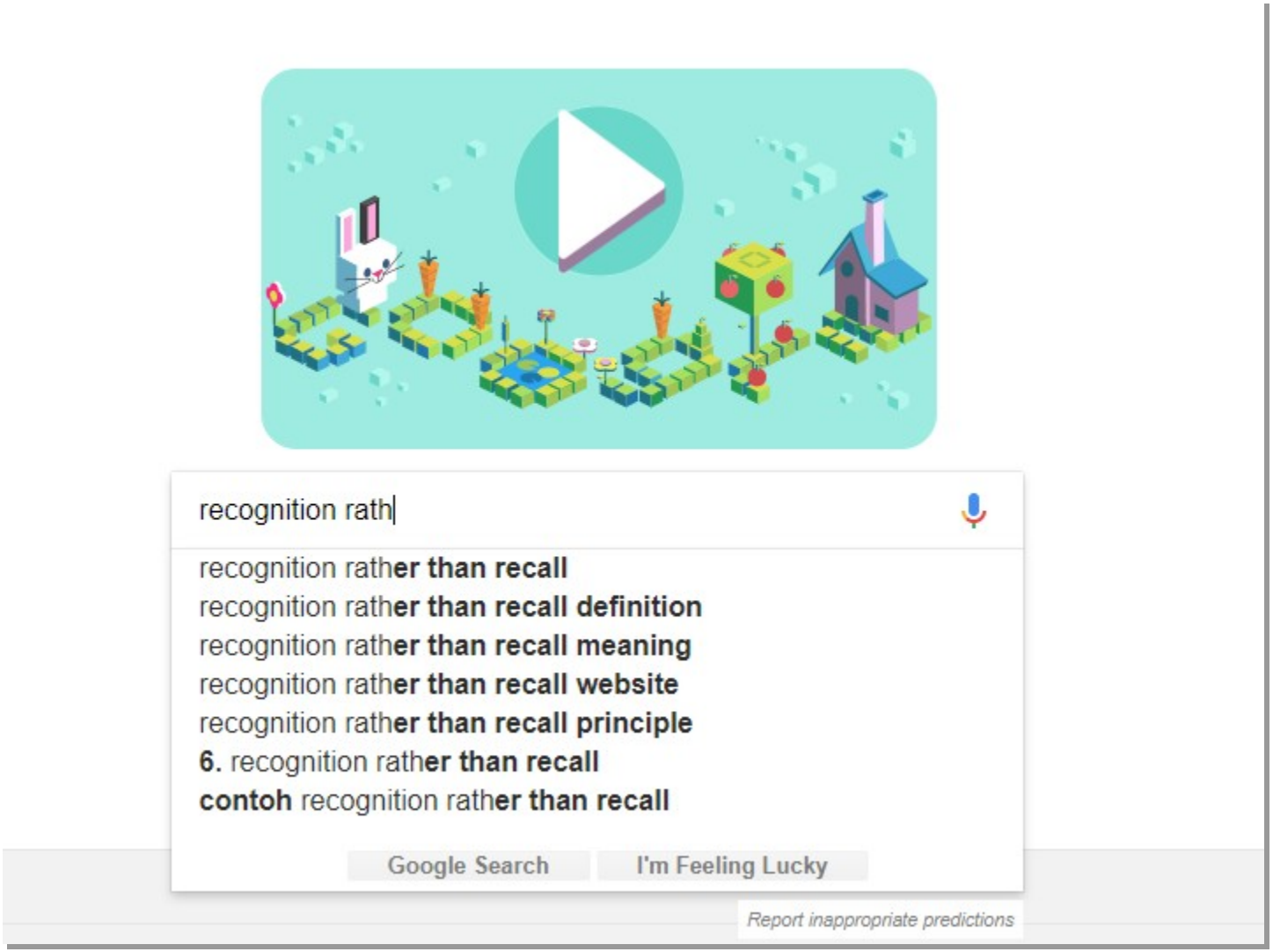
master* 0 0 | Ln 12, Col 1 | Spaces: 4 | UTF-8 | CRLF | JavaScript React

#7: Recognition rather than recall



Visited links

#7: Recognition rather than recall



#8: Flexibility and efficiency of use

Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

#8: Flexibility and efficiency of use

Provide easily-learned shortcuts for frequent operations

- Keyboard accelerators
- Command abbreviations
- Bookmarks
- History (command line interfaces)
- Templates e.g. ppt

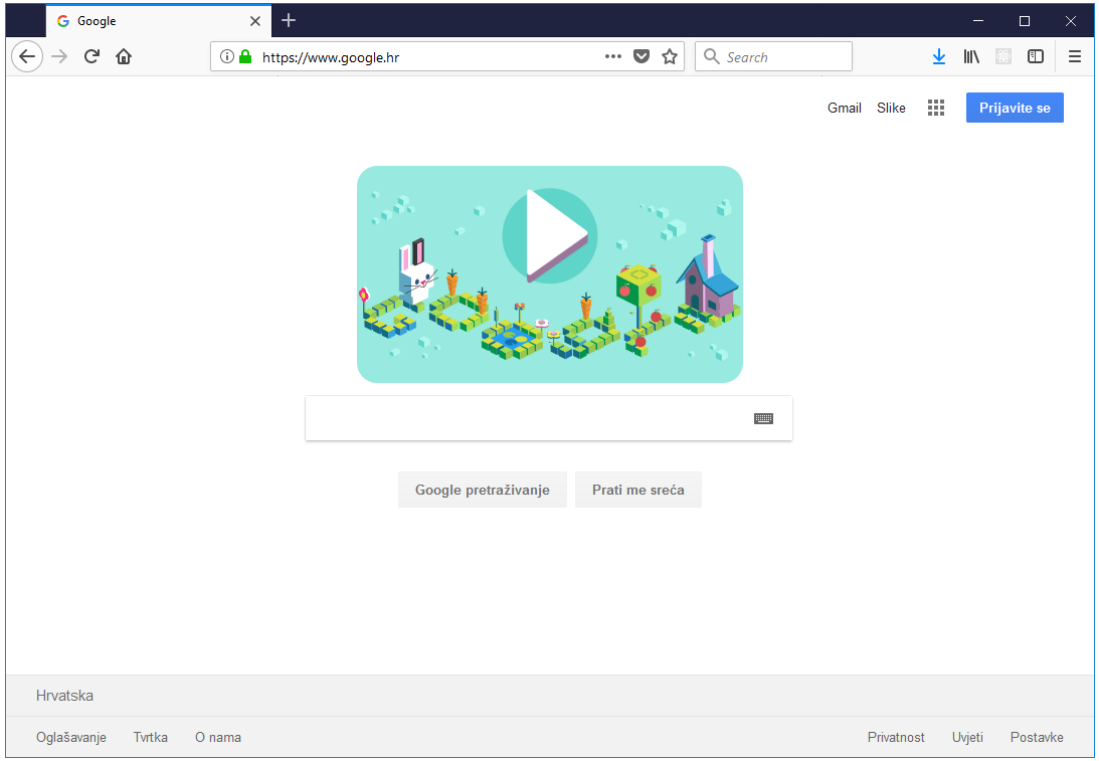
#9: Aesthetic and minimalist design

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

#9: Aesthetic and minimalist design

Less is more

- Omit extraneous info, graphics, features
- Provide only relevant data



#10: Help and documentation

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

#10: Help and documentation

Users don’t read manuals ☺

But manuals and online help are vital when user is frustrated or in crisis

Help should be:

- Searchable
- Context-sensitive
- Task-oriented
- Concrete
- Short

#10: Help and documentation

Hello World - React

https://reactjs.org/docs/hello-world.html

React Docs Tutorial Community Blog Search docs v16.2.0 GitHub

Hello World

The easiest way to get started with React is to use this Hello World example code on CodePen. You don't need to install anything; you can just open it in another tab and follow along as we go through examples. If you'd rather use a local development environment, check out the Installation page.

The smallest React example looks like this:

```
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);
```

QUICK START

- Installation
- **Hello World**
- Introducing JSX
- Rendering Elements
- Components and Props
- State and Lifecycle
- Handling Events
- Conditional Rendering
- Lists and Keys
- Forms
- Lifting State Up
- Composition vs Inheritance
- Thinking In React