Text

Arnav Gupta

April 18, 2024

Contents

1 Character Sets

Text means a series of *characters* (alpha, digit, whitespace, special). Sets of characters form a *writing system*, which need standardized encodings in binary.

1.1 ASCII

American Standard Code for Information Interchange was originally created as 7-bit encoding that was later expanded to 8-bits for total 256 characters.

Has 52 Latin characters, 10 digits, common symbols, and control characters.

American standard from 1968 onwards.

Extended 128 characters had no agreed upon encoding, so there are 15 ISO standard *code pages* for different encodings. This assumes using the correct code page to exchange international text.

1.2 Unicode

A superset of ASCII that can hold up to 1,114,112 characters, but only encodes 110,000 characters so far. Allows every character in every language to have a unique encoding, so it has replaced ASCII in common usage.

Structure is:

- 0 to 127 have same meaning as ASCII
- 128 to 256 are common signs and accented characters
- after 256, more accented characters
- after 800, Greek alphabet, then Cyrillic and more

Character codes are written as U+ and then a 4-digit hexadecimal, so Unicode encoding needs more than 1 byte (implementation not defined).

1.3 UTF-8

Universal Character Set Transformation Format 8 bit uses a multi-byte variable width encoding.

Internally, browsers use 4 byte wide characters, but in sending/receiving/storing text, this bloats storage and does not work with software that sends or receives in 1-byte units.

First byte contains info on how many bytes to expect, where 0 means only this one, 110 means 2 total, 1110 means 3 total, 11110 means 4 total. Other bytes in a character start with 10 which identifies a continuation byte.

2 Internationalization (i18n)

Designing and developing software so it can be adapted to different cultures and languages:

- use i18n features like Unicode chars and bidirectional text
- support locale formats for numbers, currency, date, time, etc.

- plan for regional differences in storing info
- separate localization elements from source code and cotent

Localization (l10n) is the act of implementing i18n. **Locale** means the region and language (ex. en_{CA} or fr_{CA}).

For localization:

1. use HTML data attribute to identify i18n text elements with `<label data-i18n="label-name"...`
2. create JSON translation data structures
3. use preferred browser locale: `navigator.language`
4. add a locale switcher (recommended)

3 Validation

Interfaces often need to validate text input typed by the user (required fields, format, ranges, uniqueness).

Form validation reasons:

1. system need correct data in the right format since the model expects certain data types or logic doesn't work
2. guides users (ex. secure passwords)
3. protects the system from attacks through unprotected text submission (ex. SQL injection)

Guidelines:

- place error messages near fields
- use colour to differentiate errors from normal field states
- when possible, accept data formatted in different ways
- when possible, filter invalid characters from being entered
- when possible, validate a field before input is complete

HTML form uses:

- `<form>` to group different input widgets together

- `<label>` with for attribute and `<input>` with corresponding id attribute
- use placeholder attribute to display an example value (or as a compact label)

Built-in HTML validation:

- use specific `type` attributes such as number or email
- use attributes for validation such as required, minlength/maxlength, min/max, pattern
- use CSS pseudo-class selectors for validation feedback such as :required or :invalid

3.1 Regular Expressions

A sequence of characters that specifies a search pattern in text:

- from language theory and theoretical CS
- a regex pattern describes a deterministic finite automaton (DFA)

Used in form validation to test if string is in correct format.

3.2 Constraint Validation API

Only available on some widgets like button, input, and select.

Using `novalidate` attribute in the form turns off standard validation messages.

API properties and methods are `validity` and `checkValidity()`.

3.3 Custom Validation

For custom input widgets, must write a custom validator:

- create classes for invalid, etc.
- listen to input event for custom widget
- test against conditions (like regex)

3.4 Input Formatting and Masking

Form text formatted as it is typed.

Input formatting updates the string in textfield as the user types, so the input event listener rewrites the textfield with standard formatting.

Input masking provides a graphical representation of the final format and fills it in as the user types:

- input event listener rewrites textfield with standard formatting and placeholders
- can use more elaborate formatting with custom element