



IIT Madras BSc Degree

Copyright and terms of use

IIT Madras is the sole owner of the content available in this portal - onlinedegree.iitm.ac.in and the content is copyrighted to IIT Madras.

- Learners may download copyrighted material for their use for the purpose of the online program only.
- Except as otherwise expressly permitted under copyright law, no use other than for the purpose of the online program is permitted.
- No copying, redistribution, retransmission, publication or exploitation, commercial or otherwise of material will be permitted without the express permission of IIT Madras.
- Learner acknowledges that he/she does not acquire any ownership rights by downloading copyrighted material.
- Learners may not modify, publish, transmit, participate in the transfer or sale, create derivative works, or in any way exploit, any of the content, in whole or in part.

Vue

Declarative Rendering

- Reactivity
- Directives
- Event handling
- Class and Style binding
- Lists and Loops

Reactivity

- Auto-update in response to changes in data
- Binding between model (data) and view (display)

Focus on **What** instead of **How**

Why?

- User interaction is fundamentally reactive
 - Respond to changes in input
- Change in one parameter may need multiple updates on screen
 - User logs in
 - Update navigation: add logged in links
 - Update displayed list of courses
 - Change colours based on theme

How?

- Server tracks *state*
 - user logged in?
 - date/time?
- Server responds with complete HTML based on present state
 - Capture different layouts, visibility in views
 - Render appropriately for each user
 - Fully server-side
- Client-JS
 - Login controller retrieves user model
 - Client side JS goes through each element and updates as needed (jQuery, vanilla JS)

Vue - directives

- v-bind: one way binding - update variable, reflects on display
 - v-model: two way binding - inputs, checkboxes etc: form data
- v-on: event binding

Class binding

- Dynamically modify class of an element
- Special support for bind - object
- Multiple classes attached based on key-values
 - If value for a given key is true, that key is applied as a class or style

Conditional rendering

- `v-if="argument"`
 - what follows is shown when argument evaluates to true
 - JS based - changes DOM
- `v-show="param"`
 - only if the show parameter evaluates to true
 - always rendered and present in DOM - only CSS display parameter changed

Looping

- Iterate over any JS iterable
 - Array, Object, String etc.
- Usage similar to Jinja `{{}}` templates
- Examples:
 - `v-for="item in items"` // items is an array
 - `v-for="value in obj"` // obj is an object
 - `v-for="(value, name) in obj"` // name -> key: "key" has another meaning in v-for
 - `v-for="(value, name, index) in obj"` // index -> numerical index

Loop Keys

- Looping “creates” new DOM elements
 - Vue needs some way to keep track of elements if updating needed
 - Virtual DOM “diffing” used to find what changes to reflect on screen
 - Vue uses heuristics to see what can be minimized
- For simple updates, simple heuristics are sufficient
 - For more complex updates, may not be easy to track what has “changed” or what is “new”
- Provide a “key”
 - Key must be unique for each loop element
 - Updating item with same key will automatically update only relevant items
- Rule of thumb: provide a key where possible - index, ID, ...

ViewModel

Model / View

- Model:
 - The “data” of the application
 - Usually stored on server in database
- View:
 - Displayed to end user (or to non-human consumer)
 - Rendering of data

Problem: sometimes *view* needs more info than needed for *model*, or perhaps there is derived information

Example

- Form with username, password, repeat_password
 - Should repeat_password be in model? Where should it be stored? How should it be used?
- Page with top comments, top posts
 - Should top_comments, top_posts be in model?
 - Should they be derived from other data?
 - What does the displayed information correspond to?

Example credits: <https://www.c-sharpcorner.com/UploadFile/abhikumarvatsa/what-is-model-and-viewmodel-in-mvc-pattern/>

ViewModel

- Yet another “Pattern”
- Create model constructs with additional data / derived data
- “bind” to view
 - Auto update view on change in data
 - (possibly) auto update data when changed in view
- Why?
 - cleaner code

Vue - ViewModel - Vue Instance

“Although not strictly associated with the [MVVM pattern](#), Vue’s design was partly inspired by it.”

- Vue documentation

- Data binding

“data” of Vue instance is the “instance data”

- Update to data will be reflected in the view
- Update in the view *can* change the data

MVC vs MVVM

- NOT either-MVC-or-MVVM
- Controller:
 - Convey actions to model
 - Call appropriate view based on inputs and model
- ViewModel:
 - Create framework for data binding
 - Can still use controllers to invoke actions

Computed Properties

- Often need to work with *derived* data
 - Take original data and modify it according to some function / logic
 - Examples: Boolean on-off on styles depending on values in data; navigation links depending on history
- Each time the source data changes, update the derived data

Computed properties

- Auto-update
- *Cached* based on their *reactive dependencies*

Computed “setter” also possible - see documentation for example

Watcher

- Explicitly look for changes
- Can be used for imperative code
- For more complex logic than just updating a property

When possible, use computed property instead of watcher

- more *declarative*
- caching

Components

Reuse

- DRY principle: Don't Repeat Yourself
- Examples:
 - News items on IITM front page
 - “People also bought” items on Amazon
- Same structure, formatting, repeated

Refactor

- Change code without changing functionality
- Mainly for readability and maintainability - not functionality

Vue Component Structure

- Properties:
 - passed down from parent - customize each instance
- Data:
 - individual data of the present instance
 - Also it's own watchers, computed properties etc.
- Template:
 - how to render
 - render functions possible - see docs
 - *Slots*

Templates

- `{{}}` format - similar to Jinja
- Safety features:
 - will not interpolate text into tags - why?
 - errors on unclosed divs etc
- More complex render functions possible
 - JSX: mix JS + HTML - similar to React

Slot

- Main element of text:
 - use like regular tag
- Properties can be defined in tag

Reactivity

How does reactivity work?

- Need to track when data is
 - accessed
 - modified
- Add methods to objects
 - Everything in JS is an object!

Object.defineProperty()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty

Examples from: <https://blog.logrocket.com/your-guide-to-reactivity-in-vue-js/>

```
const data = {  
  count: 10  
};
```

```
const newData = {  
  
}
```

```
Object.defineProperty(newData, 'count', {  
  get() { return data.count; },  
  set(newValue) { data.count = newValue; },  
});
```

```
console.log(newData.count); // 10
```

```
newData.count = 20;
```

```
const data = {  
  count: 10  
};  
  
const newData = {  
  
}  
  
function track(){  
  console.log("Prop accessed")  
}  
function trigger(){  
  console.log("Prop modified")  
}  
Object.defineProperty(newData, 'count', {  
  get() {track();return data.count; },  
  set(newValue) { data.count = newValue;trigger(); },  
});  
  
console.log(newData.count);  
// Prop accessed  
// 10  
  
newData.count = 20;  
// Prop modified
```