

# Advanced Web Programming

## CAT 2: Componentization

This CAT will address the creation of the fundamental components of the case study, and the understanding of the operation of properties, parameters and division of elements into components.

### Competencies

This CAT develops the following competences of the Bachelor's Degree in Techniques for Software Development:

- **CB5.** Develop learning skills necessary to undertake further studies with a high degree of autonomy.
- **CT3.** Adapt to new software development technologies and to future environments, updating professional skills.
- **CE6.** Design and build computer applications using development, integration and reuse techniques.


### Objectives

The specific objectives of this CAT are:

- Get familiar with how Vue.js works.
- Know basic concepts of components in Vue.


## Reading Resources and Learning Guide

As we did in the previous CAT, below we indicate the reading resources that we consider important that you review and study to carry out this CAT. We also indicate the sections (or specific chapters) that you should read carefully for the correct execution of this CAT. For each of the recommended parts, we include a comment to motivate our recommendation. Of course, if you want to go deeper, you can review and read the non-recommended sections in this list of each resource.

 *The component concept is not created in the Web world, but has existed in the area of software engineering for a long time. This concept was already introduced in one of the reference books on object-oriented software development, entitled "Object-oriented Software Construction", written by Bertrand Meyer and published in 1988. If you want to review the component concept from a software engineering (and not web engineering) point of view, we recommend reading Chapter 8 of the book [Fundamentals of Software Architecture](#), written by Mark Richards and Neal Ford.*


### Vue docs

- Essentials. <https://vuejs.org/guide/essentials/application.html>  
In this series of chapters, a brief review is made of the basic characteristics of Vue, its structure and the basic principles for using this framework. Pay special attention to the Template Syntax, Reactivity Fundamentals, and Component Basics chapters, as this is where most of the concepts needed to complete this CAT are developed.

 *Vue 3 has two types of APIs: Options API and Composition API. The use of each API implies a different programming style and the use of different methods. In this CAT **we will make use of the Options API and its documentation**.*

### Book: John Au-Yeung. [Vue.js 3 By Example](#). O'Reilly, 2021

- Chapter 4. Understanding Components  
This chapter shows, through an example project, the main features of the components in Vue 3.

 *As you may have read in the resources, components in Vue are related to Web Components, a more abstract concept that aims to offer a set of functionalities to create reusable elements on the Web. If you want to delve deeper into the concept of Web Components, we recommend reading the [Mozilla Developer Network article](#).*

## Submission format

Through the classroom you must deliver a **zip file with the content of the project's src folder and a pdf file with the answers to the questions asked**. The pdf document must have at least the name of the student in the header.

## Score

- The score for each question is detailed in its statement, **out of a total of 10 points**.
- Questions score a **maximum of 2 points**. The score is detailed in each question.
- The development of the components score a **maximum of 8 points**. The correction will take into account:
  - Correct creation of components and correct file structure (20%)
  - Correct creation of parameters/data (20%)
  - Correct creation of templates based on the information provided and the attached CSS styles (20%)
  - Correct management of highlighted post-it with background color or default color, and priority color management (10%)
  - Creation of example case with test data. (20%)
  - Overall code quality (10%)

## Contact

If you have to ask something through the forum or by email, copy your code to an online platform and send the link to avoid problems with the UOC mail (since it eliminates the .js files to avoid injecting malicious code). We recommend using either of these two options:

- [Codepen](#) (for simple code snippets)
- [CodeSandBox](#) (for more complex examples)

**Note:** In case of publishing some code in the forum, they must be generic queries and not direct solutions to the exercises. Making exercise solutions accessible to other classmates, although it may not have a direct intention, will be considered cheating and will be penalized academically at the course level.

## Intellectual property and plagiarism

The UOC Academic Regulations establish that the evaluation process is based on the student's personal work and presupposes the authenticity of the authorship and the originality of the done exercises.

The lack of originality in the authorship or the misuse of the conditions in which the evaluation of the subject is carried out, is an infraction that can have serious academic consequences.

The student will be graded with a fail (D/0) if a lack of originality is detected in the authorship of any continuous evaluation test (CAT) or final test, either because they have used unauthorized material or devices, or because they have copied verbatim from the internet, or has copied notes, from CAT, from materials, manuals or articles (without the corresponding citation) or from another student, or for any other irregular conduct.

## Questions

Answer the following questions. It is not necessary to develop them in a real project:

### Exercise 1. (0.5 points)

Define a Vue component with the following characteristics:

- Name: test-element
- Params:
  - id: number
  - items: number
  - size: string
  - width: number
  - available: boolean
- Data:
  - count: number. Initial Value: 0
  - values: array. Values:
    - {id: 1, items: 3, size: 'M', width: 90, available: false}
    - {id: 2, items: 5, size: 'L', width: 98, available: true}
- Template: <h1>I am odd</h1>

### Exercise 2. (0.25 points)

Add a method called `increaseItemsCount` to the `test-element` component created above, which receives a `items` attribute as a parameter, and adds 1 to the `count` property if `items` is a multiple of 3.

### Exercise 3. (0.25 points)

Modify the template of the `test-element` component so that it displays the `h1` header only if the value of `count` is odd. In case it is even, it should show the following:

```
<h1>I'm not and odd package</h1>
```

### Exercise 4. (0.5 points)

Modify the following *template* so that it adds the `btn-primary` class if the `primary` parameter is true:

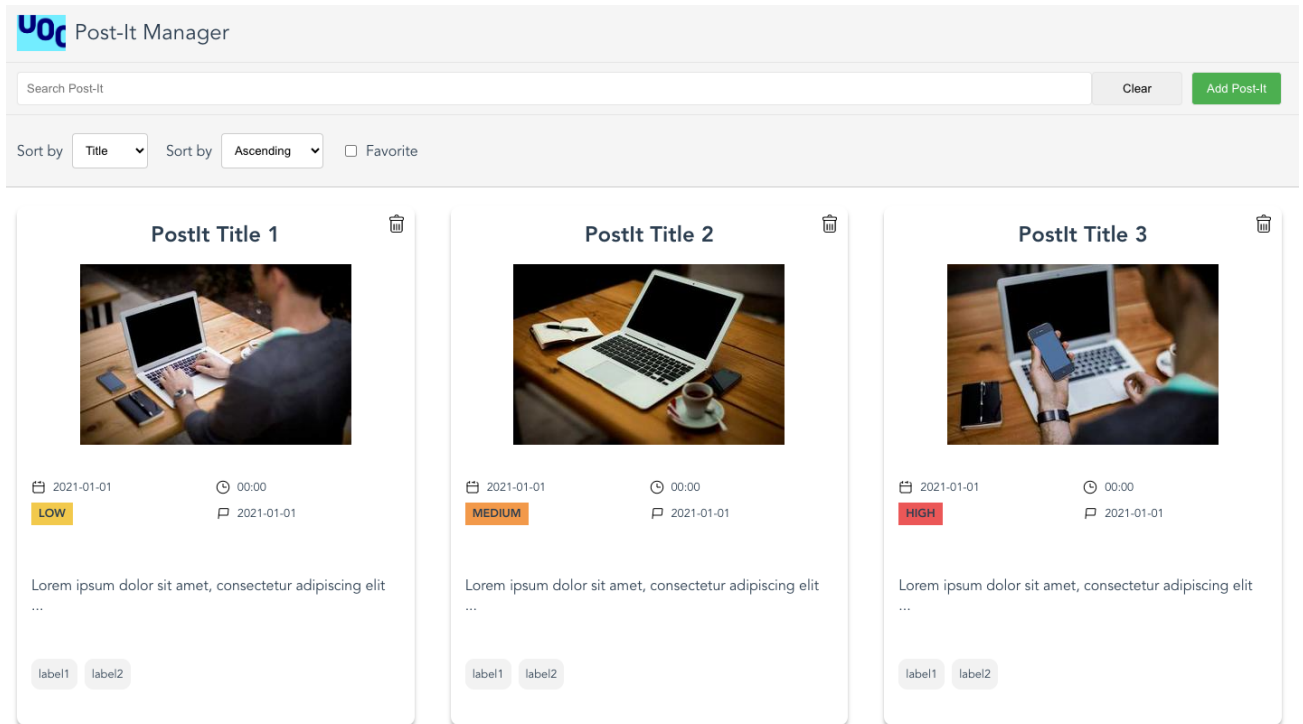
```
<button class="btn">Send packages</button>
```

### Exercise 5. (0.5 points)

Add a method called `sendPackage` to the `test-element` component created above, that call `increaseItemsCount` for each `item` element. This method should be called when the button created in the previous exercise has been clicked. Then, a message with the items delivered should be shown. Example:

```
12 items has been delivered
```

## Presentation of the case study



The objective throughout this CAT and the following ones is to develop an application in Vue.js to manage a list of post-it notes. This application will allow you to display, delete and create notes through an interface similar to the one shown in the image above. To facilitate the development, this CAT and the following ones will address each of the different aspects of Vue.js we will work on during the course.

## Components

In this CAT we will address the structure and basic development of the components that make up the application, without taking into account the origin of the data, or the communication between them.

**Note: In this CAT it will only be necessary to test the behavior of the post-it board with test data. Methods, watchers or any kind of business logic will not be evaluated in this CAT**

### PostIt.vue

This component will be in charge of managing our minimum unit of information: a post-it note.

This component will receive an object with the following parameters:

- `id`: Note ID
- `title`: Note title or name
- `image`: Note image URL
- `date`: Date of creation
- `time`: Time of creation
- `dueDate`: Due date of the note
- `dueTime`: Due Time of the note
- `priority`: Priority. Can be "low", "medium" or "high"
- `content`: Note text content
- `labels`: List of labels that the note has
- `featured`: Boolean that show if the post is featured

**Note: These parameters will need to be passed to the component as part of a `PostIt` object.**

This component must contain a template with the following characteristics:

- The content will be enclosed by a `<div>` whose class must be `.postit`
- The class `.featured` will be added to this `<div>` if the post-it passed to it as a prop contains the `featured` attribute or the `featured` attribute is true. Being an optional parameter, it will be necessary to manage its inclusion or not depending on whether the parameter is passed or not. You can find more information to implement this feature [here](#).
- Inside `.postit` there will be the following elements:
  - A button with the `.delete-postit` class that will take care of deleting the notes. The button must contain the `delete-button.svg` image that is attached to the `assets` folder.
  - An `<h2>` heading with class `.postit-title` with the title of the post-it.
  - A `<div>` with the class `.postit-image` that contains the image of the post-it.
  - A `<div>` with the `.postit-info` class that contains information about the time, due date, priority, etc... To display this information, the html structure will be as follows:
    - The params that will be shown in this part are: date, time, priority and due date.



- Each element will be composed of a `<div>` with the `.postit-“param”` class that contains the image icon for its param (located in the assets folder) and a `<span>` with the param data.
- Note: Priority does not have an icon image. The `<span>` for priority use the `.priority` class and also the class for its level of priority.
- A `<div>` with the class `.postit-content` that contains the note data
- A `<div>` with the `.postit-labels` class that contains the list of labels. To display this information, the html structure will be as follows:
  - Each label should be in one `<span>` element with the `.label` class in it.

To simplify development the component, the CSS style is provided:

```
.postit {
  background-color: #ffffff;
  border-radius: 10px;
  box-shadow: 0px 4px 4px rgba(0, 0, 0, 0.25);
  display: flex;
  flex-direction: column;
  padding: 1rem;
  position: relative;
  margin: 20px;
}
.postit-title {
  font-size: 1.5rem;
  font-weight: 600;
  margin: 0;
}
.postit-image {
  margin: 1rem 0;
}
.postit-info {
  display: flex;
  flex-wrap: wrap;
  flex-direction: row;
  justify-content: space-between;
  margin: 1rem 0;
}
.postit-date,
.postit-time,
.postit-priority,
.postit-due-date {
  display: flex;
  flex-direction: row;
  align-items: center;
  width: 50%;
  margin-bottom: 0.5rem;
}
```

```

}
.postit-date img,
.postit-time img,
.postit-due-date img {
  width: 1rem;
  margin-right: 0.5rem;
}
.postit-date span,
.postit-time span,
.postit-due-date span {
  font-size: 0.8rem;
}

.priority {
  text-transform: uppercase;
  font-weight: 600;
  font-size: 0.8rem;
  padding: 0.2rem 0.5rem;
  display: flex;
  align-items: center;
  justify-content: center;
}
.priority.low {
  background-color: #f2c94c;
}
.priority.medium {
  background-color: #f2994a;
}
.priority.high {
  background-color: #eb5757;
}

.postit-content {
  margin: 1rem 0;
  text-align: left;
}

.postit-labels {
  display: flex;
  flex-wrap: wrap;
  flex-direction: row;
  justify-content: flex-start;
  margin: 1rem 0;
}

.postit-labels .label {

```

```
background-color: #f2f2f2;
border-radius: 10px;
padding: 0.5rem;
margin-right: 0.5rem;
margin-bottom: 0.5rem;
font-size: 0.8rem;
}

.delete-postit {
position: absolute;
top: 10px;
right: 10px;
background: transparent;
border: 0;
padding: 0;
cursor: pointer;
}

.delete-postit img {
width: 20px;
}
```

## PostItList.vue

This component will be in charge of loading the list of Post-It notes, so you will have to iterate through a series of `PostIt.vue` type components. This component will receive the following parameters:

- `postItList`: Post-it notes array

This component must contain a template with the following characteristics:

- The content will be enclosed within a `<div>` with the `.postit-list` class
- Within this `<div>`, as many `PostIt.vue` components must be loaded as there are notes in the array received by parameter.

To simplify development the component, the CSS style is provided:

```
.postit-list {
display: grid;
width: 100%;
grid-template-columns: 1fr 1fr 1fr;
}
```

## SearchBar.vue

This component will be in charge of carrying out searches and showing the form to add new Post-It.

This component must contain a template with the following characteristics:

- The content is encompassed within a `<div>` with the class `.search-bar`
  - Inside it must contain an entry with the placeholder “Search Post-It”
  - Must contain a `.clear` class button with the text “Clear”
  - In turn, it must contain a `.add` class button with the text “Add Post-It”

To simplify development, the CSS style is provided:

```
.search-bar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px 20px;
  background-color: #f5f5f5;
  border-bottom: 1px solid #e5e5e5;
}
.search-bar input {
  width: 100%;
  padding: 10px;
  border: 1px solid #e5e5e5;
  border-radius: 4px;
  outline: none;
}
.search-bar button {
  padding: 10px;
  border: 1px solid #e5e5e5;
  border-radius: 4px;
  outline: none;
  cursor: pointer;
  min-width: 100px;
}
.search-bar .clear {
  margin-right: 10px;
}
.search-bar .add {
  background-color: #4caf50;
  color: #fff;
}
```

## FilterBar.vue

This component will be in charge of filtering the collection of notes shown according to certain criteria.

This component must contain a template with the following characteristics:

- The content is encompassed within a `<div>` with the class `.filter-bar`
  - It must contain a div with the class `.filter-bar__select`
    - It must contain a label and a select to be able to sort (in future CATs) by date, priority, and title
  - It must contain a div with the class `.filter-bar__select`
    - It must contain a label and a select to be able to sort the collection of post-it (in future CATs) in ascending and descending order.
  - It must contain a div with the class `.filter-bar__check`
    - It must contain a label and a checkbox to be able to filter by featured or show all.

To simplify development, the CSS style is provided:

```
.filter-bar {
  display: flex;
  align-items: center;
  padding: 20px;
  background-color: #f5f5f5;
  border-bottom: 1px solid #ccc;
}
.filter-bar__select {
  display: flex;
  align-items: center;
  margin-right: 20px;
}
.filter-bar__select label {
  margin-right: 10px;
}
.filter-bar__select select {
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
  outline: none;
}
.filter-bar__check {
  display: flex;
  align-items: center;
}
.filter-bar__check input {
```

```
margin-right: 10px;
}
.filter-bar__check label {
  cursor: pointer;
}
```

## ModalLayer.vue

This component will take care of displaying the content defined in the layer that calls it on an overlay layer. In this case it will be a form to add new postit to the collection, but it will be built in such a way that it is abstract and can display any type of component.

This component must contain a template with the following characteristics:

- A class <div> .modal. Inside it must contain:
  - And<div> with class.modal\_\_overlay
  - And<div> with class.modal\_\_content. Inside it must contain:
    - And<div> with class.modal\_\_header. Inside it, you must insert the content of type name="header". [Clue](#)
      - Must contain a button with class .modal\_\_close containing the image close-button.svg attached in the resources for this CAT.
    - And<div> with class .modal\_\_body. Inside you must insert the type content name="body"

To simplify development, the CSS style is provided:

```
.modal {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  z-index: 999;
}
.modal__overlay {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.5);
}
.modal__content {
  position: absolute;
  top: 50%;
```

```

left: 50%;
width: 90%;
height: 90%;
transform: translate(-50%, -50%);
background-color: #fff;
border-radius: 4px;
padding: 30px;
}
.modal__header {
display: flex;
align-items: center;
justify-content: space-between;
margin-bottom: 20px;
position: relative;
}
.modal__close {
background-color: transparent;
border: none;
cursor: pointer;
position: absolute;
top: 0;
right: 0;
}
.modal__close img {
width: 20px;
}

```

## PostItForm.vue

### Add a new Post-It

Title

Date

Priority

Low

Content

Featured

☐

Image

Time

Due Date

Labels

Add PostIt

This component will be in charge of managing the creation of Post-It in future CAT. Its style will be similar to the image shown above. It is basically a form that, in the future, will issue new items to be added to the post-it list.

This component must contain a template with the following characteristics:

- And<div> with class.postit-form. Inside it must contain:
  - A <form> with class.postit-form\_\_form. Inside it must contain:
    - An input (and its corresponding label for each previously defined parameter of the postit data model. It must be contained within a div with class.postit-form\_\_form-group
    - A type button submit with class .postit-form\_\_submit. It must be contained within a div with class.postit-form\_\_submit

To simplify development of the component, the CSS style is provided:

```
.postit-form__form {
display: flex;
flex-wrap: wrap;
}

.postit-form__form-group {
margin-bottom: 20px;
width: 50%;
display: flex;
flex-direction: column;
```



```

}
.postit-form__form-group label {
display: block;
margin-bottom: 10px;
text-align: left;
font-size: 14px;
}
.postit-form__form-group input,
.postit-form__form-group textarea,
.postit-form__form-group select {
width: 80%;
padding: 10px;
border: 1px solid #e5e5e5;
border-radius: 4px;
outline: none;
text-align: left;
}
.postit-form__form-group input:focus,
.postit-form__form-group textarea:focus {
border-color: #000;
}
.postit-form__form-group textarea {
height: 100px;
}
.postit-form__submit {
padding: 10px;
border: 1px solid #e5e5e5;
border-radius: 4px;
outline: none;
cursor: pointer;
min-width: 100px;
background-color: #4caf50;
color: #fff;
}
.postit-form__submit:hover {
background-color: #43a047;
}

```

## App.vue

This component will be in charge of the global management of the application and in turn will compose the general structure of the application.

For this CAT 4 **Post-It must be generated passing test values as parameters**, to simulate the appearance of the presentation image of the case study.

This component must contain a template with the following characteristics:

- Inside a template div there will be two sections with the following classes: header and content
- The template must contain a tag header with a header class. You will need to import a logo image with the class logo and add an application title (class title). You can use the UOC logo or any test image. The application name can also be modified.
- A div content should contain the following components:
  - SearchBar
  - FilterBar
  - PostItList, embedded in a main tag with main class. It must pass the list of example notes list as a parameter.
  - ModalLayer, assigning the header the title "Add new Post-It" and the body the component PostItForm. The component ModalLayer will only be displayed if the parameter showModal (must be created in App.vue component) is true

To simplify development, the CSS style is provided:

```
body {
  padding: 0;
  margin: 0;
  font-family: "Avenir", Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
}
.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 0 20px;
  background-color: #f5f5f5;
  border-bottom: 1px solid #e5e5e5;
}
.header__left {
  display: flex;
  align-items: center;
```

```
}  
.logo {  
  height: 40px;  
  margin-right: 10px;  
}  
.title {  
  font-size: 24px;  
  font-weight: 400;  
}
```