

# מדריך 3+ VUE

### הקדמה :

VUE הינה ספרייה שתופסת פופולריות לאחרונה בעולם ה JS בדומה ל REACT ו ANGULAR נחשבת כאחת הספריות הנוחות לשימוש בתעשיה ובנוסף ורסטילית לעבודה עם פלטפורמות שונות כגון וורדפרס ועוד....

### כיצד להתקין את הספרייה VUE בצורה נכונה:

npm init @vitejs/app <project-name>

ה PROJECT NAME ישתנה לשם תיקיית הפרויקט שתתוקן באותו ספרייה שניתן את הקוד ב CMD (בלי <>)  
לאחר מכן תשאל איזה פרוייקט תרצה להתקין … הבחירה כמובן צריכה להיות VUE בשאלה השניה תשאל אם תרצה לעבוד כ TS ולפי הצורך (כרגע המדריך יעבוד לפי VUE הרגיל)

לאחר מכן יש להכנס לתקייה ולכתוב את הפקודה הבאה כדי להתקין את המודלים של סביבת הפיתוח VUE:

Npm install

לבסוף יש כדי לטרנספל במצב פיתוח את הפרוייקט כדי להציגו בדפדפן (על פי הקוד הנוכחי) יש לכתוב את הקוד הבא:

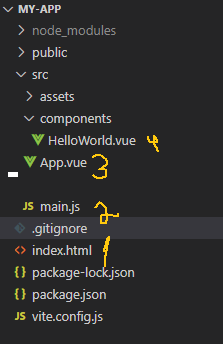
Npm run dev

לבסוף יש להכנס לכתובת הבאה כדי להריץ את הפרוייקט:

<http://localhost:3000/>

המלצה שלנו לעבוד עם העורך VS CODE ולהוריד את התוסף **VETUR בתוכנת VS CODE**

### סדר הקבצים כיצד פועל ב VUE:



### הכנת סביבת העבודה למדריך זה:

כדי שמבחינת עיצוב התוצאות של האפליקציות במדריך יראו סבירות נחבר לקובץ ה INDEX HTML שלנו בכל האפליקציות CDN של BOOTSTRAP 5 בתגית ה HEAD

מכאן:  
<https://getbootstrap.com/>

### כיצד קבצי הקומפינטות (תכנות) של VUE נראים

נתחיל עם יצירת אפליקציית COUNTER קטנה ובהמשך נסביר את עניין הקומפוננטות ב VUE

לכן בתור התחלה נעבוד עם קובץ ה APP.VUE שנמצא בתיקיית ה SRC בלבד ונתחיל כאשר הוא נראה כך (תוכלו להעתיק מהחוברת):

<template>

</template>

<script>

export default {

name:'App',

}

</script>

<style>

</style>

תגית ה TEMPLATE מרמזת שהיא הולכת להכיל את ה VIEW (כלומר את ה HTML שלנו והתצוגה של הקומפנינטה {כרגע מבחינתכם תוכנת ה VUE})

תגית ה SCRIPT תייצג את המוח והתכנות של הקומפוננטה

תגית ה STYLE תכיל עיצוב קונקרטי לקומפוננטה ולקובץ ה VUE הספציפי שאנחנו עובדים עליו וישפיע על התגיות שנמצאות ב TEMPLATE

#### 4.א נתחיל בחלק ה TEMPLATE ונשים את הקוד הבא :

<template>

<div class="container">

<h1>Welcome to VUE 3</h1>

<h2>Counter: </h2>

</div>

</template>

הצעד הבא שנרצה לעשות הוא שתגית ה H2 תהפוך להיות צבע אדום דרך הגדרה בקקומפנינטה באזור הסטייל



#### 4.ב ובשביל זה נוסיף בתגיות STYLE את הקוד הבא:

<style>

h2{

color:red;

}

</style>

יפה אז כבר הבנו כיצד ה TEMPLATE וה STYLE עובדים וכמו שהנכם רואים די דומה ל HTML (תגית הטמפלט) ו CSS (תגית הסטייל) שאנחנו יודעים ומכירים מהעבר :)

וגם ה JS די דומה רק עם יכולות מיוחדות שעכשיו נראה (שמקצרות לנו דווקא את הדרך לכתיבה וממש עוזרות)

#### 4.ג הגדרת משתנים בתוך הטמפלט

דבר ראשון בואו נגדיר ב TEMPLATE משתנה שהמידע שלו יגיע מה SCRIPT (בשורות הבאות של המדריך תבינו בדיוק כיצד איך) בתגית H2 נוסיף משתנה בשם counter שיגיע מאובייקט STATE:

<h2>Counter:{{counter}} </h2>

הצעד הבא שנעשה הוא בתגית ה SCRIPT נייבא את יכולת ה" REACTIVE " של VUE ונגדיר משתנה בשם COUNTER

הסיבה שאנחנו לא מייצרים ישר משתנה בשם COUNTER ומשתמשים בו הוא כדי לנצל את יכולות ה VIRTUAL DOM של VUE .

<script>

export default {

name:'App',

data() {

return {

counter:”Moshe”

}

},

methods: {

}

}

</script>

כאשר נפעיל את האפליקציה עכשיו יוצג ליד ה COUNTER המילה MOSHE,

אך במקרה שלנו אנחנו רוצים בעצם לייצר קאונטר של מספר שעל ידי לחיצה בכפתור יתחיל לספור ולכן נשנה את ערך ה COUTNER מ BIDEN ל 1 שימו לב למפאיין METHODS שישמש אותנו בשביל יצירת מיטודות שישפיעו על משתנים שהגדרנו ב DATA

### EVENTS/ אירועים והגדרת פונקציות/מיטודות ב VUE

בנוסף נוסיף בתגית הSCRIPT במאפיין METHODS את הפונקציה הבאה

export default {

name:'App',

data() {

return {

counter:1

}

} ,

methods: {

add1Counter (){

this.counter++

}

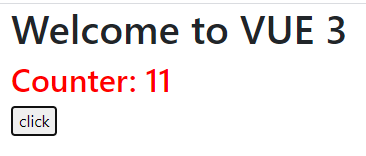
}

}

ובטמפלט מתחת לתגית ה H2 נוסיף את הקוד הבא של כפתור עם האזנה ללחיצה על ידי הפקודה @click :

<button @click="add1Counter()">click</button>

*להלן התוצאה הסופית (לאחר לחיצה 10 פעמים על כפתור ה CLICK):*



ולהלן הקוד מלבד אזור ה STYLE (שהגדרנו בסעיף 4.ב ומעתה לא נתייחס כמעט לתגית STYLE בקוד בהמשך המדריך ונשתמש בקלאסים של BS אלא במקרים מיוחדים {התגית סטייל תשאר ריקה} ) של כל האפליקציה

<template>

<div class="container">

<h1>Welcome to VUE 3</h1>

<h2>Counter: {{counter}}</h2>

<button @click="**add1Counter**()">click</button>

</div>

</template>

<script >

export default {

name:'App',

data() {

return {

counter:1

}

} ,

methods: {

**add1Counter** (){

this.counter++

}

}

}

\* שימו לב שניתן במקום @ לכתוב **v-on**: אך סימן @ חוסך את הכתיבה

בנוסף לא חובה להוסיף סוגריים לפונקציה שנקראה מ EVENT

### תנאים ב VUE -

כמו שכבר הינכם יודעים תנאים הם עניין חשוב וב VUE ניתן להכניס אותם בקלות(בהמשך המדריך נדבר עליהם יותר בהרחבה) אך בשביל הדוגמא הראשונית בוא נוסיף לטמפלט שלנו שה H1 יוצג רק אם ה COUNTER גבוה מ4

כל מה שנצטרך לעשות הוא פשוט להוסיף את הדיירקטיב (כמו אטרביוט ב HTML רגיל) V-if בצורה הבאה על H1 :

<h1 **v-if**="counter > 4">Welcome to VUE 3</h1>

ושימו לב איך בתוך הערך יכלנו להכניס פשוט תנאי פשוט אם COUNTER גדול מ 4, נוכל בשיטה דומה להכניס משתנה שהוא BOOLEAN ובעזרת כפתור שיפעיל פונקציה ותשנה אותו ל TRUE,FALSE (TOGGLE

**המלצה** למדו גם את ELSE ו ELSE IF של VUE

### כיצד להעביר פרמטרים לפונקציה שנקראת מ EVENT

לצורך הדוגמא הבאה נייצר כפתור שיוסיף 5 יחסיר 2 ושניהם ישתמשו באותה פונקציה שתקבל פרמטר, אז קודם נוסיף נגדיר את הפונקציה בשם UPDATECOUNTER בתגית הSCRIPT במאפיין METHODS מתחת לפונקציה הקיימת שיצרנו קודם בשם ADD1COUNTER

updateCounter(event,val) {

this.counter += val;

}

\* שימו לב שהעברנו 2 פרמטרים , מהסיבה הפשוטה שלפעמים נרצה מידע על EVENT עצמו (נניח בגרירה איפה ציר X או Y או בלחיצה מי האלמנט שהפעיל את הפונקציה) ועכשיו נייצר את 2 הכפתורים שיפעילו את הפונקציה בתגית ה TEMPLATE:

<button @click="updateCounter($event,5)">add 5</button>

<button @click="updateCounter($event,-2)">reduce 2</button>

בדוגמא של הקוד שימו לב שכארגיומנט/פרמטר ראשון העברנו את $EVENT שמכיל בתוכו את כל המאפיינים של ה EVENT ככה שנוכל להתשמש בו , ורק אז את הערך של ה VAL מהפונקציה שלנו

### כיצד לייצר לולאות V-FOR ב VUE

דבר ראשון נעדכן את STATE (המאפיינים ב DATA ) שלנו ונוסיף משתנה מסוג מערך בשם ar עם 3 אובייקטים שיכילו שם ו ID בSCRIPT (פורמט JSON ) בפונקציית DATA :

data() {

return {

counter: 1,

**ar**: [

{ name: "Koko", id: 1 },

{ name: "Moshe", id: 2 },

{ name: "David", id: 3 },

],

};

}

ובטמפלט נוסיף את הקוד הבא :

<ul>

<li v-for="item in **ar**" :key="item.id">

{{item.name}}

</li>

</ul>

שימו לב לדייקרטיב V-FOR ולדיירקטיב :KEY (שמומלץ להוסיף)  
נתחיל עם V-FOR : השימוש בו מזכיר את FOR IN ב JS ו ECMA ואפילו את ה MAP ונוכל בעצם להוציא כל פעם שהלולאה עובדת את הערך של המאפיינים ולהדפיס אותם (במקרה זה הדפסנו את מאפיין NAME)

לגבי מאפיין :KEY שהוא אופציונאלי - חכם להוסיף אותו בשביל ה VIRTUAL DOM של VUE שנדבר עליו בהמשך … אך בקצרה המאפיין הזה דואג שהתצוגה בדפדפן תיהיה מהירה יותר כאשר יש אינטקרטיביות לרשימה כגון הוספה /החסרה /עריכה וכו' …

### כיצד לייצר ATTRIBUTE BIND (או מאפיינים שמקבלים ערך דינמי)

נניח שהגדרנו בתוך ה STATE (יש להגדיר בתוך האובייקט שב RETURN פונקציית DATA ) שלנו את המאפיין הבא:

myInput:"color",

עכשיו אם נרצה להוסיף את הערך של המאפיין MYINPUT לתוך אטרביוט לא נוכל להשתמש ב {{}} אלא נצטרך לכותב את הקוד הבא:

<input v-bind:type="myInput" >

וכפי שאתם רואים כל מה שנצטרך לעשות כדי VUE יבין שאני רוצה לקבל את הערך של MYINPUT ולא את הביטוי עצמו הוא להוסיף v-bind: לפני האטרביוט הרגיל , וכך זה יעבוד בכל האטרביוטים כגון HREF, SRC וכו'....

כמו כן ב VUE החדש ניתן לשמוט את V-BIND ולהשתמש רק בנקודתיים ":" בצורה הבאה:

<input **:**type="myInput" >

### כיצד לייצר קלאס דינמי

דבר ראשון נייצר קלאס בשם BOX בתוך תגית ה STYLE בקובץ עם המאפיינים הבאים:

.box{

border:4px solid green;

width:400px;

height: 400px;

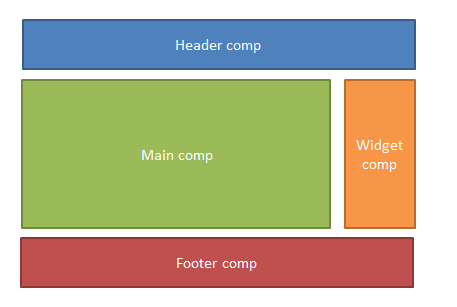
}

נייצר DIV ב TEMPLATE עם הביטוי BOX ONE וניתן לו קלאס דינמי שיעבוד במידה וה COUNTER גדול מ4:

<div :class="{box: counter > 4}">box one</div>

ושימו לב שבעצם בצורה כזו תוכלו לתת תנאי לקלאסים של CSS שיעבדו רק במצבים מסויימים (כמובן כמו ב V-IF ניתן להכניס משתנה BOOLEAN)

### מה היא קומפנינטה:

ב VUE נוכל לבנות את האפליקציה /אתר שלנו (מעכשיו ייכתב אפליקציה בלבד אך הכוונה גם לאתר) ממרכיבים (קומפונינטות בשם המקצועי שלהם VUE/ריאקט) לדוגמא: אתר בסיסי שיש לו אזור HEADER,MAIN,WIDGETS SIDE,FOOTER נוכל לחלק אותו לקומפנינטות בצורה הבאה:  


וכפי שהבנתם מפרק 4 בחוברת זו כל קומפוננטה מורכבת מתגיות TEMPLATE - שמיצגת את התצוגה (VIEW) של הקומפנינטה בכתיבת HTML עם הכנסת משתנים בעזרת {{}} מה SCRIPT - שאחראי על המוח של התוכנה ולהעביר לטמפלט נתונים ותגית הSTYLE שאחרית על עיצוב ה CSS של הקומפנינטה בנוסף ניתן להכניס לכל קומפנינטות מספר קומפנינטות בנות נוספות וגם לקומפנינטות בנות ניתן להכניס עוד קומפנינטותו וכך הלאה

### כיצד מייצרים קומפנינט חדשה ומחברים אותה לאפליקציה

מייצרים את הקומפנינטה הראשונה שלנו ומחברים אותה ל APP, אז נתחיל שוב מ APP נקי שנראה כך:

<template>

<div class="container">

<h1>{{title}}</h1>

</div>

</template>

<script >

export default {

name: "App",

data() {

return {

title:"Learn componentes app"

};

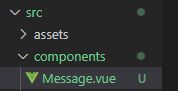
}

};

</script>

* בתחתית ניתן להוסיף כמון תגית SCRIPT

בנוסף בתוך תקיית components נייצר קומפנינטה חדשה בשם Message.vue



והקופנינטה תראה כך מבחינת קוד:

<template>

<div class="bg-info text-white p-3 h-4">

Mesage: new component of message

</div>

</template>

<script>

</script>

<style>

</style>

למי שלא מודע ל BS הקלאסים שנוספו יהפכו את הדיב לרקע בצבע תכלת עם טקסט לבן, פדינג וטקסט בגודל H4 , כמובן יש לחבר בקובץ index.html קובץ מ CDN של BS כפי שהתבקשת בסעיף 3

עכשיו כדי שנוכל להשתמש בקומפנינטה בקומפנינטת APP נצטרך לכתוב את הקוד הבא בקומפנינטת ה APP שלנו בתגית הSCRIPT קודם:

<script >

import **Message** from './components/Message.vue';

export default {

**components**: **{** **Message }**,

name: "App",

data() {

return {

title:"Learn componentes app"

};

}

};

</script>

* *שימו לב שהיה עלינו לעשות IMPORT קודם כל ל קומפנינטה (ניתן לתת לה כל שם אך נהוג לתת את השם של הקובץ שלה (במקרה שלנו MESSAGE) ויש להגדיר מאפיין componentes בתוך ה EXPORT DEFAULT עם האובייקט שעשינו לו IMPORT כמאפיין*

וכל מה שנשאר כדי שנוכל להשתמש בקומפנינטה בטמפלט , הוא פשוט לכתוב אותה בתוך הטמפלט כתגית (לא לשכוח את הסלש "/" בסופה) :

<template>

<div class="container">

<h1>{{title}}</h1>

<Message />

</div>

</template>

### כיצד ליצור STYLE מקומי

כיצד לעשות תגית STYLE של קומפוננטה תשפיע רק על הקומפוננטה ולא תעביר את העיצוב של הסלקטורים בה לקומפוננטות אחרות:

בשביל זה כל מה שנצטרך לעשות זה להוסיף אטרביוט על תגית ה STYLE בשם SCOPED

<style **scoped**>

h2{

color:red;

}

</style>

### כיצד להעביר PROPS מקופנינטת אב לקומפנינטת ילד

( נניח במקרה שלנו קומפנינט APP (אב) תעביר מידע לקומפנינטת MESSAGE (ילד)

נניח שנרצה שנוכל להשתמש בקומפנינטה של MESSAGE מספר פעמים אך כל פעם ה קומפנינטה תציג הודעה שונה (מה שיקרה באמת במהלך אפליקציה אמיתית) כל מה שנצטרך מקומפנינטת האב (APP) לעשות הוא לכתוב בתוך תגית הקומפנינטה ילד שם של מאפיין שאנחנו נחליט עליו (נניח שאנחנו נחליט על TXT מלשון טקסט ואת הערך שלו במקרה שלנו סטרינג בגרשיים { בדומה ל ATTRIBUTE על תגית HTML רגילה} )

<template>

<div class="container">

<h1>{{title}}</h1>

<Message **txt**="hello this is props text" />

</div>

</template>

בנוסף בקומפנינטת ילד (MESSAGE) נצטרך קודם להגדיר ב SCRIPT את קבלת ה PROPS בצורה הבאה:

<script>

export default {

props: [**'txt'**]

}

</script>

ובטמפלט של הקומפינטה MESSAGE נוכל פשוט להשתמש ב PROPS כמו שהשתמשנו במאפיין :

<template>

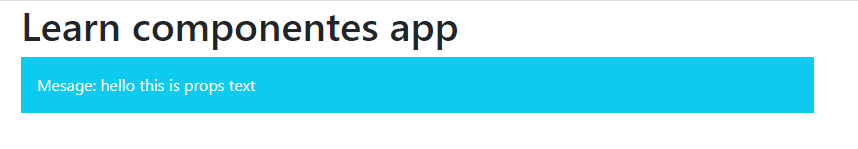
<div class="bg-info text-white p-3 h-4">

Mesage: {{**txt**}}

</div>

</template>

ולהלן התוצאה:



במידה והיינו רוצים נניח להעביר את המשתנה TITLE שנמצא בAPP בתור ה PROPS TXT היינו צריכים לכתוב אותו כדיירקטיב כפי שלמדנו בסעיף 9 עם ":"

<Message :**txt**="title" />

כי אחרת הוא היה מעביר את המילה TITLE ולא את הערך שלה

### כיצד להחזיר מידע מקופנינטת בן לקומפנינטת אב

נתחיל בזה שנייצר כפתור ב APP שנלחץ עליו יסתיר/יציג את קומפנינטת ה MESSAGE בצורה הבאה:

<script >

import Message from './components/Message.vue';

export default {

components: { Message },

name: "App",

data() {

return {

title:"Learn componentes app",

showMessage:false

}

},

methods:{

toggleMessage(){

this.showMessage = !this.showMessage

}

}

};

</script>

אז כפי שהנכם רואים הוספנו לקופנינטה שלנו מאפיין בשם SHOWMESSAGE (שישמש עם V-IF אם להציג את הקומפנינטת MESSAGE) ובנוסף יצרנו מיטוד/פונקציה בשם TOGGLEMESSAGE שמשנה את SHOWMESSAGE מ FALSE ל TRUE וההפך

עכשיו נוסיף עדיין בקומפנינטת ה VUE כפתור שיפיעיל את הפוקנציה בתוך הTEMPLATE:

<button @click="toggleMessage">toggle</button>

עכשיו מה שנרצה לעשות זה לייצר בתוך הקומפנינטת MESSAGE עצמה כפתור שידע גם להפעיל את הפונקציה TOGGLEMESSAGE שנמצאת אצל קומפנינטת האב שלו ה APP

כדי לעשות את זה אנחנו נצטרך לייצר בתוך ה MESSAGE פונקציה שכאשר נפעיל אותה תשגר אירוע שאנחנו נמציא ונוכל דרך קומפנינטת האב בדומה ל CLICK להאזין לאירוע החדש שהמצאנו … אז נתחיל עם הכתיבה שנכניס לתוך קומפנינטת ה MESSAGE ב TEMPLATE :

<template>

<div class="bg-info text-white p-3 h-4">

Mesage: {{txt}}

<button @click="**closeMessage**()" class="btn btn-dark">close</button>

</div>

</template>

ונייצר את הפונקציה CLOSEMESSAGE שהגדרנו בכתפתור החדש שיצרנו לו שתשגר אירוע

<script>

export default {

props: [**'txt'**],

methods:{

**closeMessage**(){

this.**$emit**(*'onClose'*)

}

}

}

</script>

ושימו לב לפקודה חדשה שגליתם שיש לקומפנינטה בשם $emit שמקבלת את שם ה EVENT שנרצה לשגר לקומפנינטת האב (במקרה שלנו ONCLOSE שאנחנו המצאנו יכלנו לקרוא לה גם MOSHE)

עכשיו כל מה שנשאר לנו כפי שהוזכר קודם הוא להאזין לאירוע החדש בשם ONCLOSE שיצרנו ממש כמו שהיינו מאזינים לאירוע לחיצה (@CLICK) ונציב אותו על תגית הקומפנינטה עצמה ככה שהטמפלא של ה APP יראה כך בסוף התהליך:

<template>

<div class="container">

<h1>{{title}}</h1>

<Message @*onClose*="toggleMessage" v-if="showMessage" **txt**="hello this is props text" />

<button @click="toggleMessage">toggle</button>

</div>

</template>

* במידה והיינו רוצים להעביר מידע שהפונקציה אצל קומפנינטת האב תוכל לאסוף אז ב פקודת ה EMIT היינו צריכים לכתוב (נניח את הסטרינג "mango"

this.**$emit**(*'onClose',null, “mango”*)

ובהגדרת הפונקציה כדי לאסוף את המידע נניח מפרמטר שנקרא VAL היינו צריכים להגדיר (בדומה לסעיף 7):

toggleMessage(event,val){

this.showMessage = val

}

### מה זה SLOTS ומה שימושיה

אז SLOTS בעצם דומה ל INNER HTML של תגית רגילה שאנחנו מכירים ב HTML רק ל COMPONENT

בוא נניח שאנחנו נרצה שתיהיה לנו אפשרות להוסיף לינקים לתוך ה MESSAGE שלנו :



אפשרות אחת שתגידו היא להכניס את המידע כדיירקטיב (אך הבעיה שלא תוכלו להעביר תגיות באופן פשוט והם יכתבו כפשוטם ולא יוצג כתגיות)

ולכן נשתמש בשיטה ייחודית של VUE (לעומת REACT ו ANGULAR) בשם SLOTS:

אז קודם ב תוך קומפנינטת APP נזריק את קומפנינטת MESSAGE כמו תגית מכילה עם פתיחה וסגירה:

<Message @onClose="toggleMessage" v-if="showMessage" txt="hello this is props text" >

<template v-slot:**links**>

<a href="#">more info</a>

</template>

</Message>

שימו לב שדאגנו שתגית הMESSAGE תיהיה תגית מכילה (נפתחת ונסגרת) ובאופן מוזר כתבנו שוב תגית TEMPLATE שנמצאת בתוכה וכדי להבדיל אותה מהתגית TEMPLATE של הקומפנינטה הכנסנו דיירקטיב בשם V-SLOT עם הערך LINKS (השם שנשתמש בו בקומפנינטת הבת ) ובתוכו לינק אחד (יכלנו גם להכניס 5 תגיות שונות ויותר וכל תגית HTML שעולה לנו בראש כגון תגיות IMG, H1 וכו'..... )

כדי להציג את הלינק שנמצא ב TEMPLATE בתגית האב (APP) באזור שנרצה ספציפית בתוך הקומפנינטה נכתוב את הקוד הבא ב TEMPLATE

<template>

<div class="bg-info text-white p-3 h-4">

Mesage: {{txt}}

<button @click="closeMessage()" class="btn btn-dark">close</button>

<br />

<slot name="links" />

</div>

</template>

ושימו לב לתגית ה SLOT שהוספנו לה דיירקטיב בשם NAME שמכיל בדיוק את השם שהכנסנו ב V-SLOT בקומפנינטת האב שמפעילה אותה (links).

### מה היא תגית TELEPORT

כפי ששמתם לב הסיטנקס של VUE (ויו) עובד בצורה של מקונן כלומר אני כותב קומפנינטות בתוך קופמנינטות … ודוגמא לבעייתיות של השיטה לעתים נדירות :נניח שיש לי תגית שאני רוצה שתיהיה במצב FIXED (צפה קבוע) במצב הנוכחי יוצא שכשאני מגדיר את הקומפנינטה בתוך קומפנינטה אחרת שהיא עדיין נמצאת בתוך DIV כלשהו , TELEPORT מאפשר לי בקלות להוציא אותה ל DIV ושהקומפנינטה מבחינת הDOM תיהיה בת ישירה של תגית ה BODY , ל TELEPORT יש יכולות גם להעביר קומפנינטה לתוך תגית אחרת במסמך שנבתנה עם ה VUE או בצורה שונה (נניח WORDPRESS ) לפי ה ID של התגית עם הDIRCTIVE TO

### תגובה לשינוי PROPS

כיצד לבצע פעולות על מאפיין בקמופנינטה כל פעם שיש שינוי ב PROPS שהקומפנינטה מקבלת מקופנינטת האב שלה

לדוגמא נניח יש לנו קומפנינטה שמציגה תאריך … וצריכה להציג בחישוב כמה ימים נותרו לתאריך היעד

מה שנוכל לעשות הוא להוסיף ל OPTIONS שלנו מאפין בשם WATCH בדומה ל METOD , ולתת שם של פונקציה בשם של ה PROPS, ובכל פעם שה PROPS DEADLINE במקרה שלנו ישתנה הוא יפעיל פונקציה שתעשה חישוב על מאפיין הימים שלנו

<template>

<div class="col-lg-6 mx-auto text-center">

<h2>Date count down: {{ **deadline** }}</h2>

<h3>Days left:{{ **days** }}</h3>

</div>

</template>

<script>

export default {

props:['deadline'],

data(){

return{

**days**:99

}

},

watch:{

**deadline**(){

console.log("watch")

let time = Date.parse(this.**deadline**) - Date.now()

this.**days** = Math.floor(time / (1000\*60\*60\*24));

}

},

}

</script>

### שימוש בFORM ו INPUTS

#### 19.א כיצד לאסוף מידע מ INPUT,

כדי לאסוף מידע מ INPUT ישנה דרך ממש קלילה שנקראת V-MODEL ומשתמשים בה על ידי הצבת DIRECTIVE על INPUT מה שמאפשר ביצוע של 2 WAY BINDING

לדוגמא ב TEMPLATE נכניס INPUT עם הדיירקטיב V-MODEL בשם EMAIL1

<input **v-model**="email1" placeholder="Enter your email" type="email" class="form-control border-0"/>

<h2>Your Email:{{email1}}</h2>

ובתוך ה OPTIONS ניצור משתנה/מפאיין בשם EMAIL1 , ובעצם כל שינוי שנבצע ב INPUT , המשתנה EMAIL1 יקבל את הערך שנמצא בתוך ה VALUE של ה INPUT

<script>

export default{

data(){

return{

email1:"mindalizeev@gmail.com"

}

}

}

</script>

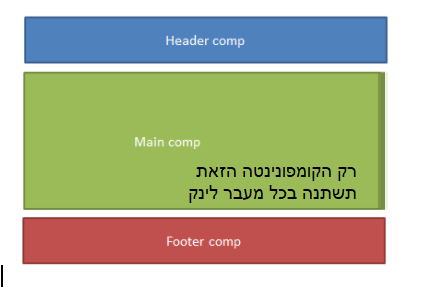
ושימו לב כאשר תתחילו להקליד ב INPUT ב H2 יכתב אחרי הנקודתיים בדיוק מה שמוצג ב INPUT

את ה V-MODEL ניתן להכניס באותו אופן ל SELECT BOX ו TEXTAREQ ולכל סוג INPUT

### כיצד לעבוד מול ROUTE

כפי שהנכם הבנתם אחד מיתרונתיה הרבים של VUE היא היכולת להריץ אפלקציית עמוד אחד , כך שכאשר אנחנו עוברים בין דפים , במקום לרפרש את הדפדפן כל פעם מחדש רק האזור שצריך להתשנות ישתנה .

לדוגמא רוב האתרים בעולם בנויים שב 90 אחוז מהעמודים שלהם יש HEADER ו FOOTER ויהיה חבל שנעבור בין עמודים שנצטרך לטעון מחדש את כל העמוד ולא רק את אזור ה MAIN…



ולכן בשביל זה הומצא ה ROUTER,

כדי שנוכל לעבוד עם ה ROUTER מומלץ מלכתחילה להתקין גרסת תומכת ROUTER של ה CLI שלנו ב VUE

ולכן נתקין מחדש את הVUE עם היכולת ROUTE שבאה בנויה בתוכו

לכן נצטרך להתקין את ה VUE בצורה שונה מסעיף 1 , ודבר ראשון נתקין את ה VUE CLI על המחשב שלנו ב CMD:

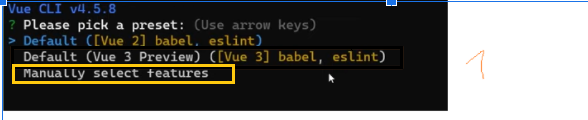
npm install -g @vue/cli

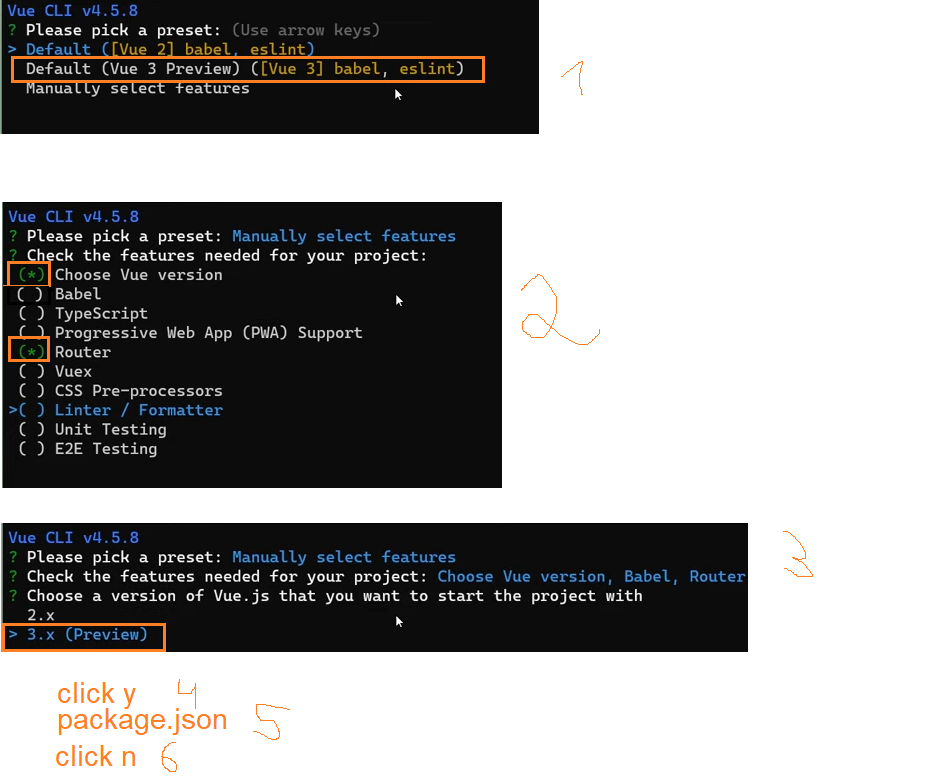
הדבר הבא שנעשה נתקין פרוייקט חדש בשם VUE-ROUTER עם הקוד הבא:

Vue create vue-router

כמובן את ה ה vue-router ניתן לשנות לשם אחר של הפרוייקט

ברגע שנבצע זאת יפתח החלון הבא ולהלן הצעדים שיש לבחור:





לאחר שנבצע את כל המהלכים הללו הפרוייקט יראה די דומה למה שעשינו עד עכשיו עם כמה שינויים קטנים:

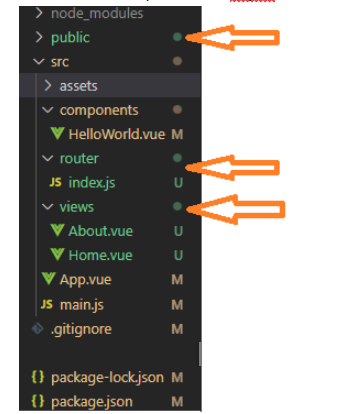
והפעם כדי להריץ את הפרוייקט נכתוב ב CMD

Npm run serve

והכתובת גישה לאפליקציה במצב DEV תיהיה http://localhost:8080/

מעתה כל פרוייקט חדש ככה נתקין אותו וככה נפעיל אותו (עד להודעה חדשה {עדכון חוברת})

ושימו לב לתקיות חדשות עם קבצים חדשים שנוספו שנסביר אותם כעת:



לפני הכל שימו לב שמעכשיו קובץ ה INDEX.HTML נמצא בתוך תקיית PUBLIC בסביבת פיתוח של ה CLI החדש… שמבחינתנו חיבור ה FLATSOME וה BS יתרחש שם.

לאחר מכן נתחיל מקובץ ה JS INDEX בתקיית ROUTER שנעשה לו שינוי קטן ויראה כך:

import { createRouter, createWebHistory } from 'vue-router'

import Home from '../views/Home.vue'

import About from "../views/About.vue"

const routes = [

{

path: '/',

name: 'Home',

component: Home

},

{

path: '/about',

name: 'About',

component: About

}

]

const router = createRouter({

history: createWebHistory(process.env.BASE\_URL),

routes

})

export default router

* לפני שנסביר מה קורה כאן שימו לב לאזור הצהוב בתוך ה ROUTES בהגדרת הרואט ל ABOUT שמחקנו את מה שכתוב (שזה פשוט שיטה מתקדמת יותר שנדבר עליה בהמשך אם יהיה צורך) והפכנו אותו לדומה להגדרה של רואט עמוד הבית שקל יותר להסביר

אז נתחיל מה IMPORT הראשון של היכולות שנקראות createRouter ו createWebHistory שהן מאפשרות את המעבר בין הלינקים ללא רפרוש (שנגיע לקובץ ה APP VUE נבין כיצד מתפעלים אותם)

לאחר מכן יצרנו מערך שבו נגדיר את כל הכתובות שה VUE ידע לזהות ואיזה קומפנינטה הוא יפעיל , דוגמא בטבלה הבאה

|  |  |
| --- | --- |
| כתובת | קומפוננטה שתפעל באזור תגית <router-view/> ב APP |
| http://localhost:8080/ | views/home.vue |
| http://localhost:8080/about | views/about.vue |

#### 20.א כיצד לייצר ראוט חדש סטטי

אם נניח היינו רוצים שיהיה כתובת לקומפנינטה גלריה חדשה שנמציא נניח -> <http://localhost:8080/gallery> מה שהיינו צריכים לעשות בתיאוריה הוא להוסיף אובייקט לROUTES שנראה כך:

{

path: '/gallery’,

name: 'Gallery',

component: Gallery

}

וכמובן לעשות לקובץ קומפנינטה IMPORT בתחילת הקובץ .

לאחר מכן ניתן לראות את המשתנה ה router שלו אנחנו גם עושים EXPORT שנייבא בקובץ main.js שמגדיר הגדרות שונות בניהן מה קורה שלוחצים BACK או NEXT בכפתור הדפדפן וכיצד לנהל את ה URL/ROUTE שהמשתמש נמצא בו.

אם נסתכל על קובץ ה MAIN (חלקו) נוכל לראות משהו מעניין שנוסף

import router from './router'

createApp(App).**use**(router).mount('#app')

שימו לב שלפני שהאפליקציה מפעילה את הקומפנינטה הראשונה APP היא משתמשת במידלוואר (פונקצית אמצע) שמשתמש ב ROUTER שנמצא בקובץ INDEX בתקיית ROUTER שראינו מעלה .

לאחר מכן שנסתכל על קומפנינטת ה APP עצמו נוכל לראות את הדברים הבאים:

<template>

<div id="nav">

<router-link to="/">Home</router-link> |

<router-link to="/about">About</router-link>

</div>

<**router-view**/>

</template>

נתחיל אם זה שנסתכל על DIV עם ה ID בשם NAV ...

דבר ראשון שנבחין שהם לינקים אך במקום להשתמש בתגית A הוא משתמש בתגית ***router-link*** ובמקום האטרביוט HREF יש שימוש ב דיירקטיב ***to*** …. דרך אגב כאשר נקמפל את האפליקציה ונכנס אליה ב

Localhost:8080

תוכלו לראות שאם תעשו INSPECT התגית שתוצג הינה A , אך מה שמיוחד בלינקים הללו שבמקום לרפרש את כל הדף הם רק ישנו בדפדפן את הכתובת למעלה ויטענו את הקומפנינטה בהתאם למה שהגדרנו בקובץ router/index.js בתוך האזור שהתגית ***router-view*** מופיעה….

וזה כל המורכבות כיצד הרואטר עובד , מכאן נוכל לייצר קומפנינטות נוספות עם כל היכולות שלמדנו עד כה

#### 20 ב. כיצד לייצר רואט עם פרמטר

פעמים רבות נרצה לייצר ראוט שנוכל להעביר לו פרמטר ונוכל לאסוף אותו , למשל נייצר קומפנינטה ב FRUITS שיהיה בה מערך של פירות ולפי ה URL/ROUTE היא תדע להציג למשתמש את הפרי הנכון

לדוגמא נניח יש לנו מערך :

Ar =ar:["melon","banana","kiwi"]

ומה שנרצה לעשות אם לדוגמא יכנסו ל URL הבא:

<http://localhost:8080/fruits/2>

יוצג הפרי KIWI אבל אם לעומת זאת יכנסו לכתובת

[http://localhost:8080/fruits/](http://localhost:8080/fruits/2)0

יוצג פרי המלון

אז על מנת לבצע זאת קודם כל בקובץ ה INDEX JS בתקיית ה ROUTE בתוך מערך ROUTES נגדיר את ה PATH הבא:

{

path: '/fruits/:id',

name: 'Fruits',

component: Fruits

}

* הנקודתיים לפני ה ID מרמז שאיי די זה לא כתובת סטטית אלא פרמטר שנוכל לשנות אותו ולאסוף אותו מה URL.

בנוסף נייצר קומפנינטה בשם FRUITS שנראת כך:

<template>

<h2>Fruits {{ar[0]}}</h2>

</template>

<script>

export default {

data(){

return{

ar:["melon","banana","kiwi"]

}

}

}

</script>

כרגע הקומפנינטה שנפעיל אותה תמיד תציג MELON לא משנה אם נכנס לכתובת :

<http://localhost:8080/fruits/2>,

[http://localhost:8080/fruits/](http://localhost:8080/fruits/2)1

וכו'...

לכן כדי שנוכל לאסוף את הפרמטר נייצר משתנה ב DATA בשם ID (אפשר גם שם אחר) וכדי לאסוף את הפרמטר נכתוב את הקוד הבא ב DATA:

data(){

return{

ar:["melon","banana","kiwi"],

id:this.$route.params.id

}

}

ובתוך ה TEMPLATE נחליף ב AR את ה 0 ל ID

<h2>Fruits {{ar[id]}}</h2>

#### 20 ג. כיצד לייצר ROUTE שמבצע REDIRECT בשבילנו ,

אז נניח נרצה שאם מגיעים ל FRUITS נשוגר אוטומטית ל fruits/0

{

path: '/fruits',

redirect: "/fruits/0"

}

#### 20.ד כיצד ליצור דף 404 שיוצג שמגיעים לכתובת לא נכונה

אז בעצם שחושבים על זה במונחי VUE מה שנרצה לעשות היא לייצר קומפנינטת ברירת מחדל שתטען אם לא נמצא שום ראוט מתאים

אז קודם ניצור קומפוננטה בשם Page404 כמובן תוכלו לעצב אותו כרצונכם עם תמונות רקעים ועוד….

<template>

<h1>Page 404 not found!</h1>

</template>

<script>

export default {

}

</script>

לאחר מכן פשוט נוסיף בקובץ ה INDEX JS בתקיית ה ROUTE בתוך מערך ROUTES את ה PATH הבא:

{

path: '/:catchAll(.\*)',

name: 'page404',

component: Page404

}

ובעצם ה PATH בשם '/:catchAll(.\*)' אומר שכל רואט אחר מעבר לאלו שומפיעים כאן (מכיוון שהוא נמצא אחרון ברשימה) יפעיל את הקומפנינטה PAGE404

#### 20 . ה כיצד לשגר ידנית דרך ה SCRIPT לכתובת מסויימת

פעמים רבות נתקל במצבים שבהם נרצה לשגר את המשתמש לכתובת מסויימת לאחר פעולה מסויימת שהתרחשה ב JS כגון בקשת FETCH או הכנסת מידע בטופס…. לכן כדי לבצע את הפעולה בקלות והפקודה נקראת (כאן מוצגת בתוך פונקציה שנמצאת ב MEHTODS של ה OPTIONS ב SCRIPTS ויכול להקרא מכפתור למשל:

methods:{

goPage(){

this.$router.push({ name:"About"})

}

}

20. ו אפשרות נוספת להתקנת VUE ROUTER

קודם כל יש להתקין דרך NOM את הגרסה האחרונה של VUE ROUTER בפקודה הבאה:

npm install vue-router@4

לאחר מכן יש לייצר קובץ router.js עדיף ישירות בתקית SRC

שיראה כך:

// need to install vue-router@4

import { createRouter, createWebHistory } from 'vue-router'

import Home from "../components/Home.vue";

import About from "../components/About.vue";

const routes = [

{

path: '/',

component: Home

},

{

path: '/about',

component: About

}

]

const router = createRouter({

history: createWebHistory(),

routes

})

export default router

* שימו לב שבדוגמא הנל יש לעשות IMPORT נכון בהתאם לקומפנינטות שקיימות לכם באפליקציה של ה HOME וה ABOUT

ובסנוף בקובץ ה MAIN.JS שיראה כך (עם פונקציה ה MIDDLEWARE בשם USE)

import { createApp } from 'vue'

import App from './App.vue'

import router from "./router/router.js"

createApp(App).use(router).mount('#app')

כל השאר זהה לכל התתי סעיפים שקשורים ל סעיף 20 כולל הוספת לינק ב 20.א עם <router-link

### כיצד לבצע בקשת FETCH בצורה נכונה

לפני שנתחיל את כתיבת ה API החדש של VUE בשם COMPOSTION API הגיע הזמן להבין איך לבצע בקשת API לכתובת מסויימת , במקרה שלנו

נניח שנרצה להביא את רשימת המוצרים שמופיעים כ JSON בכתובת הבאה:

<http://fs1.co.il/bus/shop.php>

אז נייצר לעצמנו קומפנינטה חדשה בשם Shop.vue בתקיית ה VIEWS ונגדיר את ה PATH של הSHOP (עשו זאת בעצמכם לפי סעיף 20 א')

וקוד ב SCRIPT של הקומפנינטה SHOP יראה כך:

<script>

export default {

data() {

return {

**ar**: [],

};

},

mounted() {

console.log("mounted");

this.doApi();

},

methods: {

doApi() {

let url = "http://fs1.co.il/bus/shop.php";

fetch(url)

.then((resp) => resp.json())

.then((data) => {

console.log(data);

this.**ar** = data;

});

},

},

};

</script>

* שימו לב שהוספנו ב OPTIONS שלנו פונקציה חדשה בשם MOUNTED שזאת פונקציה שתעבוד פעם ראשונה שהפונקציה מוצגת ומאותחלת (גם מעבר בין ראוטים וחזרה לרואט שמציג את הקומפנינטה יפעיל את הפונקציה)
* הדבר הבא שקורא כבר די מזכיר את JS עם ECMA6 אנחנו קוראים למיטודה שיצרנו ב שם DOAPI שמבצעת פעולת FETCH (ניתן לעבוד גם **ASYNC AWAIT**) ובסוף התהליך שימו לב שאנחנו משווים את ה מאפיין AR ל DATA שזה המידע שקיבלנו מה API

לאחר שאנחנו מודעים לזה שה AR מכיל את המידע של ה JSON מהבקשת API בTEMPLATE של קומפנינטת ה SHOP נכתוב את הקוד הבא כדי לייצר רשימה:

<template>

<div class="container">

<h1>Shop comp</h1>

<ul>

<li v-for="item in ar" :key="item.id">

{{ item.name }} - {{ item.price }} NIS

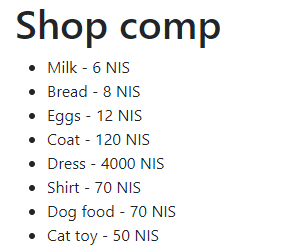
</li>

</ul>

</div>

</template>

ולהלן התוצאה:



### מה זה VUE COMPOSITION API - שיטה חדשה 2021+

החל משנת 2021 החברה של VUE פיתחו API חדש לשפה (בדומה להוקס של REACT) שמאפשרים תכנות קל יותר ונכון יותר ב VUE … הקונספט עדיין נשאר זהה עם שינויי סיטקס קטנים, נתחיל כיצד נראה טמפלט של VUE החדש , במקרה הזה נתחיל מ 0 ונשתנה את קומפנינטת HOME שנמצאת בתקיית VIEWS שתראה כך בתור התחלה:

<template>

<div class="home">

<h1>Test home</h1>

</div>

</template>

<script>

export default {

setup(){

//data

//methods

//computed

//lifecycle

//return stuff to template dom

}

}

</script

שימו לב שבשיטת הCOMP (מעכשיו ירשם גך COMPOSITION) אנחנו נכתוב את רוב הקוד שלנו בתוך פונקציית ה SETUP …

בתור התחלה בואו נייצר משתנים (סטטים שלא נתנים לשינו בהמשך נבין איך ניתן לעשות להם REACTIVE כך ששינוי שלהם ישתנה גם ב TEMPLATE אוטומטית (דורש REF שנלמד בהמשך)

לכן בטמפלט נכתוב את הבא בקומפנינטה HOME:

<template>

<div class="container">

<h1>Test home</h1>

<div>

Name: {{name}} - Score: {{*score*}}

</div>

</div>

</template>

וכדי לייצר את את המשתנים ב DATA נכתוב:

<script>

export default {

setup(){

//data

let name = "sonic";

let *score* = 100;

//methods

//computed

//lifecycle

//return stuff to template dom

return {name,*score*}

}

}

</script>

שימו לב שיצרנו אותם כרגע כמשתנים רגילים לחלוטין , ומה שחשוב הוא ה RETURN שעשינו להם בסוף ה SETUP כדי שהטמפלט של הקומפנינטה תכיר אותם

בואו נוסיף ל TEMPLATE כפתור שיפעיל מיטודה בשם CHANGE NAME:

<div>

Name: {{name}} - Score: {{score}}

<br>

<button @click="changeName">change name</button>

</div>

וב SETUP של ה SCRIPT:

setup(){

//data

let name = "sonic";

let score = 100;

//methods

const changeName = () => {

name = "mario"

console.log(name)

}

//return stuff to template dom

return {name,score, changeName}

}

* לחיצה על הכפתור לא תעבוד

#### 22.א איך לעבוד מול REF

שננסה להפעיל את הקוד בסעיף הקודם נראה שלחיצה על כפתור לא יהיה שינוי בשם והסיבה לכך שאנחנו חייבים לפעול עם REF על מנת שנוכל לשנות משתנה שהוא יהיה REACTIVE ולכן הקוד יכתב כך (שימו לב לעשות IMPORT כמו בשורה הראשונה של ה SCRIPT

<script>

import { **ref** } from '@vue/reactivity';

export default {

setup(){

//data

let name = **ref**("koko");

let score = **ref**(50);

//methods

const changeName = () => {

name.**value** = "mario"

console.log(name)

}

//return stuff to template dom

return {name,score, changeName}

}

}

</script>

* בפונקציה עצמה שימו לב שיש לדבר עם מאפיין VALUE ולא ישירות עם משתנה NAME מכיוון שכרגע הוא אובייקט , וה REF של REACTIVE עובד בצורה כזו שברגע שמשנים את המאפיין הוא ידע לעדכן אותו בטמפלט.

#### 22. ב שימוש במשתנה STATE עם REF בצורה נכונה יותר: שימו לב שבדומה ל REACT נהוג שאת המשתנים שיצרנו נאגד בתוך משתנה מסוג אובייקט והמשתנים יהיו מאפיינים של המשתנה STATE ולכן נייצר את אותו אפליקציה בצורה הבאה:

<template>

<div class="container">

<h1>Test home</h1>

<div>

Name: {{state.name}} - Score: {{state.score}}

<br>

<button @click="changeName">change name</button>

</div>

</div>

</template>

<script>

import { ref } from '@vue/reactivity';

export default {

setup(){

//data

let state = ref({name:"moshe",score:20})

// let name = ref("koko");

// let score = ref(50);

//methods

const changeName = () => {

state.value.name = "mario"

console.log(state)

}

//return stuff to template dom

return {state, changeName}

}

}

</script>

* הכתיבה די דומה לקודם : היתרון של שיטה זו שאם ניצור מאפיין נוסף בקומפנינטה לא יהיה לנו צורך לעשות לו RETURN גם, החסרון של השיטה היא שכדי לשנות עכשיו את המאפיין אנחנו נדרשים לכתוב state.value.name

ולכן הומצא כדי לעבוד עם השיטה הזאת דרך חדשה מלבד REF (שבדרך כלל ישמש אותנו יותר לקישור עם INPUT ו SELECT BOX עם V-MODEL שלמדנו בסעיף 19) בשם REACTIVE

#### 22 ג. כיצד לעבוד עם REACTIVE

שימו לב ל IMPORT ולקוד שהוספנו בתוך הSCRIPT:

import { ref ,reactive } from '@vue/reactivity';

export default {

setup(){

//data

// let state = ref({name:"moshe",score:20})

let **state** = reactive({name:"david",score:80})

// let name = ref("koko");

// let score = ref(50);

//methods

const changeName = () => {

**state**.name = "mario"

console.log(state)

}

//return stuff to template dom

return {state, changeName}

}

}

על פניו כמו שניתן לראות הכתיבה די זהה ל REF אך עם שינוי קטן , שימוב לב שבפונקציה לא היינו צריכים לכתוב state.value כדי לדבר עם המאפיין אלא רק state.name , בנוסף ה REACTIVE עובד מאחורי הקלעים בצורה חכמה יותר כאשר עובדים עם אובייקט STATE מ REF, וכפי שהוסבר בסעיף הקודם REF ישמש אותנו בעיקר לתקשורת עם תגיות INPUT ו SELECT או כאשר יש לנו קומפנינטה שמכילה משתנה אחד או 2 גג (בלי קשר ל INPUT)

\* בהמשך תראו שלא ניתן להאזין לשינויים במשתנים מסוג REACTIVE ועוד… וגם שם נעדיף להשתמש ב REF

### 23 . כיצד לעבוד עם COMPUTED

נניח שיש לנו מערך של שמות של עובדים שחלק מהתאים במערך מתחילים עם MR וחלק עם MS ואנחנו רוצים לשלוף במהירות ולייצר פונקציה שתחזיר רק את MR (גברים) נוכל להשתמש ב COMPUTED בצורה הבאה:

import { ref ,reactive } from '@vue/reactivity';

import { computed , watch } from '@vue/runtime-core';

export default {

setup(){

//data

// let state = ref({name:"moshe",score:20})

let state = reactive({name:"david",score:80})

let workers = ref(["mr mario","mr luigi","ms piggy","ms queen","ms croft","mr koko"])

const maleWorkers = computed(() => {

// יחזיר רק את התאים שכתוב בהם mr

return workers.value.filter(name => name.includes("mr "))

})

const changeName = () => {

state.name = "mario"

console.log(state)

}

//return stuff to template dom

return {state, changeName, maleWorkers}

}

}

* במקרה זה רק דוגמא לשימוש שלאחר מכן בטמפלט נוכל לבצע לולאה ולייצר את רשימת הגברים , אך היתרון הגדול של COMPUTED שאם יוצרים מערך שמתפלט בריאקטביות נניח כמו חיפוש ההחזרת מידע מתבצעת בצורה מהירה יותר

<ul>

<li v-for="item in maleWorkers" :key="item">

{{item}}

</li>

</ul>

### 24. כיצד WATCH עובד והשימוש בו

פעמים רבות נרצה שכאשר יהיה שינוי ב PROPS שנקבל מקומפנינטת אב או שינוי במאפיין STATE כלשהו שתתבצע פעולה כלשהי… בשביל זה המציאו את ה WATCH , במקרה שלנו נעשה שכאשר משתנה NAME1 שמחובר ל REF (שימו לב שלא ניתן להאזין למאפיין של STATE )יתווסף למאפיין SCORE אחד:

setup(){

//data

// let state = ref({name:"moshe",score:20})

let state = reactive({name:"david",score:80})

let name1 = ref("koko")

let workers = ref(["mr mario","mr luigi","ms piggy","ms queen","ms croft","mr koko"])

const maleWorkers = computed(() => {

return workers.value.filter(name => name.includes("mr "))

})

**watch**(name1,() => {

console.log("aaa")

state.score += 1;

})

const changeName = () => {

state.name = "mario"

name1.value = "mario"

console.log(state)

}

//return stuff to template dom

return {state, changeName, maleWorkers}

}

}

* שימו לב שיש לעשות IMPORT לפקודת WATCH שנעשתה כבר בסעיף 23 ושבשביל הבדיקה יצרנו משתנה NAME1 שלא שומש בTEMPLATE

### 25. כיצד לעבוד מול PROPS שמגיע מאב בשיטת COMPS

דבר ראשון ניצור קופנינטה בשם MESSAGES בתוך תקיית COMPONENTS ונחבר אותה לקומפנינטת HOME בצורה הבאה בטמפלט (בדיוק כמו בשיטה הישנה של OPTIONS API) :

<template>

<div class="container">

<h1>Test home</h1>

<div>

Name: {{state.name}} - Score: {{state.score}}

<br>

<button @click="changeName">change name</button>

<***Messages*** />

</div>

</div>

</template>

דבר שני שנעשה בדיוק כמו בשיטה הישנה זה לייבא את הקומפנינטה ב IMPORT ולחבר אותה במאפיין COMPONENT בדומה למה שעשינו בסעיף 14 בספר

<script>

import { ref ,reactive } from '@vue/reactivity';

import { computed, watch } from '@vue/runtime-core';

import Messages from '../components/Messages.vue';

export default {

components: { Messages },

setup(){

//data

// let state = ref({name:"moshe",score:20})

let state = reactive({name:"david",score:80})

let name1 = ref("koko")

let workers = ref(["mr mario","mr luigi","ms piggy","ms queen","ms croft","mr koko"])

const maleWorkers = computed(() => {

// יחזיר רק את התאים שכתוב בהם mr

return workers.value.filter(name => name.includes("mr "))

})

const changeName = () => {

state.name = "mario"

name1.value = "mario"

}

//return stuff to template dom

return {state, changeName, maleWorkers}

}

}

</script>

הצעד הבא שארצה לעשות הוא להעביר לקופנינטה את המשתנה name שנמצא ב STATE כPROPS בשם USER נניח

על ידי הגדרת ה USER כדיירקטיב בכתיבת הקומפנינטה בTEMPLATE בשיטה הבאה (דומה לסעיף 14):

<Messages :user="state.name"/>

וכל מה שנצטרך לעשות בקומפנינטה עצמה של MESSAGES הוא להגדיר את המשתנה בתוך מאפיין PROPS:

<template>

<h1>Message work - {{user}}</h1>

</template>

<script>

export default {

props:["user"]

}

</script>

#### 25 .א כיצד לבצע מניפולציות על PROPS שמגיע מהאב בתוך ה SETUP בקומפננטת הבן :

בוא נניח שנרצה תמיד להוסיף MR לשם שמגיע מה PROPS USER

כל מה שנצטרך לעשות זה להעביר כפרמטר ב SETUP את הPROPS ונייצר פונקציות שמשתמשת בפקודת ה COMPUTED בצורה הבאה בתוך קומפנינטת MESSAGES:

<script>

import { computed } from '@vue/runtime-core'

export default {

props:["user"],

setup(**props**){

const ***addMr*** = computed(() => {

return "mr "+**props**.user

})

return { ***addMr*** }

}

}

</script>

ולאחר מכן בתוך הטמפלט נכתוב :

<h1>Message work - {{***addMr***}}</h1>

### 26. שימוש ב LIFECYCLE ב API COMP:

כדי להשתמש בפונקציות של מחזורי חיים של VUE בשיטה ה COMPOSITION כל מה שנצרך הוא לקרוא לפוקנציות בתוך ה SETUP

ישנם 3 לייף סייקלים עיקריים ב VUE:

|  |  |
| --- | --- |
| יפעיל את הפוקנציה כל פעם שהקומפנינטה מוצגת בפעם הראשונה (אם נניח יש ראוט ונעבר לראוט אחר שיעלים את הקומפנינטה ונחזור הקומפנינטה תפעיל את עצמה מחדש) | onMounted |
| יקרא רגע לפני שהקומפנינטה נמחקת | onUnmounted |
| בכל שינוי בסטייט או PROPS שמגיע מהאבא ההפונקציה תקרא | onUpdated |

דוגמא לשימוש פקודת LIFECYCE בתגית ה SCRIPT (יופעל ברגע שנפעיל את הקופמנינטה וטוב לקריאה לפעולת API דרכו כגון FETCH , AXIOS ועוד...:

import { onMounted } from 'vue'

export default {

setup () {

onMounted(() => {

console.log('mounted ! after comp loaded')

})

}

}

בעתיד

* לכתוב על ELSE
* לכתוב על SWITCH
* לכתוב על REDUX עם VUE
* לכתוב על PAYPAL
* לכתוב על DEBUGGER