



Универзитет „Св. Кирил и Методиј“ во Скопје  
**ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И  
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО**

---

Дипломска работа

*Hacker News Clone*

---

Ментор:  
**Проф. Д-р Костадин Мишев**

Изработил:  
**Стефан Марковски 203180**



Универзитет „Св. Кирил и Методиј“ во Скопје  
**ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И  
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО**

Скопје 2025

---

Тема:

**Hacker News Clone**

---

Автор:

**Стефан Марковски**

---

Научна област:

**Развој и дизајн на веб страници**

---

Ментор:

**Проф. Д-р Костадин Мишев**

---

Датум на одбрана:

**XX.XX.2025**

---

Членови на комисија:

**Магдалена Костовска**

-  
Факултет за информатички  
науки и компјутерско  
инженерство

**Костадин Мишев**

-  
Факултет за информатички  
науки и компјутерско  
инженерство

**Ана Тодоровска**

-  
Факултет за информатички  
науки и компјутерско  
инженерство



## Апстракт

---

Овој проект има за цел да развие модернизирана верзија на платформата Hacker News со фокус на унапредување на визуелниот изглед, функционалноста и целокупното корисничко искуство. Hacker News, како платформа за споделување технолошки вести, проекти и идеи, иако содржински вредна, се соочува со застарен интерфејс и ограничена употребливост. Проектот предлага нов дизајн што е почист, поинтуитивен и визуелно привлечен, овозможувајќи лесна навигација и подобрена читливост.

Главната цел е задржување на основната функционалност на оригиналната платформа, додека се интегрираат современи технолошки решенија кои ќе го унапредат искуството на корисниците. Користејќи го Algolia API за пребарување и официјалното Hacker News API за преземање содржини, апликацијата овозможува брз и прецизен пристап до релевантни информации. Податоците се прикажуваат преку ефикасна пагинација, а кеширањето преку Local Storage овозможува офлајн пристап до последните 15 отворени објави.

Проектот користи Vue.js како основна frontend рамка, овозможувајќи реактивно управување со податоци и компоненти. За стилизирање се користи рачно напишан CSS, што обезбедува целосна контрола врз дизајнот. Backend-от не е посебно развиван, туку се потпираме на постоечки API-ја, што го поедноставува одржувањето на системот и гарантира точност и ажурност на содржините.

Проектот исто така цели да:

- Воведе кориснички интерфејс со модерен, интерактивен изглед.
- Овозможи ефикасно прелистување на објави, коментари и прашања.
- Обезбеди брзо и интелегентно пребарување преку Algolia.
- Подобри перформанси и намали оптоварување преку пагинација.
- Обезбеди офлајн пристап со локално кеширање на содржини.



Универзитет „Св. Кирил и Методиј“ во Скопје  
**ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И  
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО**

## Содржина

---

1. Вовед .....	5
2. Цели .....	5
3. Користени технологии .....	6
4. Имплементација .....	7
5. App.vue .....	8
6. Router.vue .....	9
7. Header .....	10
8. Views .....	11
9. MainPage .....	11
10. SideBar .....	13
11. SideBarElement .....	15
12. MainContent .....	17
13. Filters .....	20
14. Item .....	22
15. Loader .....	25
16. Api -----	26
17. Заклучок и идни подобрувања -----	27
18. Користени технологии и код -----	28



## 1. Вовед

---

Целта на овој проект е да се развие модернизирана верзија на платформата Hacker News, подобрувајќи го нејзиниот кориснички интерфејс и искуство. Проектот има за цел да обезбеди почист, поинтуитивен дизајн, додека ги задржува сите основни функционалности на оригиналната платформа.

Hacker News е широко користена платформа за споделување и дискусија на технолошки вести, проекти и идеи. Иако нуди вредна содржина, нејзиниот тековен кориснички интерфејс изгледа застарен и му недостасуваат некои современи подобрувања на употребливоста. Овој проект има за цел да создаде подобрена верзија користејќи го Algolia API за Hacker News, воведувајќи подобар дизајн, подобрена читливост и полесна навигација.

Главниот фокус е да се испорача оптимизирано искуство при прелистување со функционалности како што се подобрени можности за пребарување, кеширање за офлајн пристап и лесна пагинација. Користејќи модерни frontend технологии и официјални API-ја, целта е да се изгради апликација која останува верна на основната функционалност на Hacker News, но значително го подобрува нејзиниот изглед и пристапност.

## 2. Цели

---

### **Подобрување на корисничкиот интерфејс и искуство:**

Наместо статичен и минималистички дизајн, оваа верзија ќе воведо почист и попривлечен изглед со интерактивни елементи и интуитивна навигација, што ќе го направи користењето попријатно за корисниците.

### **Обезбедување ефикасен начин за прелистување на објави, коментари и прашања:**

Корисниците ќе можат лесно да пристапат до сите категории на содржина без непотребни преоптоварувања и со организирани секции кои ќе им овозможат побрзо наоѓање на релевантни теми.



Универзитет „Св. Кирил и Методиј“ во Скопје  
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И  
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

### **Имплементација на напредно пребарување:**

Користењето на Algolia API ќе овозможи побрзо и попрецизно пребарување на содржините, што ќе им помогне на корисниците брзо да најдат теми од интерес.

### **Овозможување пагинација за полесна навигација:**

Со цел подобро искуство, податоците ќе бидат прикажани во мали, управливи делови наместо да се вчитуваат сите одеднаш, што ќе ја зголеми брзината на апликацијата и ќе го намали оптоварувањето на серверот.

### **Кеширање на најновите објави за офлајн пристап:**

Со имплементација на кеширање, корисниците ќе можат да пристапуваат до последните 15 отворени објави дури и кога немаат интернет конекција, што значително го подобрува корисничкото искуство.

### **Преземање податоци користејќи официјални API-ја за точна и ажурирана содржина:**

Овој пристап гарантира дека содржините ќе бидат секогаш точни и навремени, а интеграцијата со официјалните API-ја ќе го олесни одржувањето на системот.

## **3. Користени технологии**

---

Во овој проект користиме неколку технологии и алатки за да обезбедиме подобро корисничко искуство при прелистување на содржините од Hacker News.

### **Frontend:**

**Vue.js:** За развој на корисничкиот интерфејс користиме Vue.js, прогресивен JavaScript framework кој овозможува ефикасно и реактивно управување со податоците и компонентите.

**CSS:** Стиловите се рачно напишани со CSS, што ни овозможува целосна контрола врз изгледот и дизајнот на апликацијата, обезбедувајќи конзистентен и прилагодлив интерфејс.

### **Backend:**

Не користиме сопствен backend, бидејќи се потпираме на постоечките API-ја за преземање на потребните податоци.



## API-ja:

**Hacker News API:** Официјалното Hacker News API обезбедува пристап до различни податоци како објави, коментари, корисници и други информации. Ова API е едноставно и лесно за користење, овозможувајќи ни да ги преземеме најновите и најпопуларните објави од платформата.

**Algolia API:** За подобро пребарување, го користиме Algolia API, кое овозможува брзо и ефикасно пребарување низ содржините на Hacker News. Algolia обезбедува моќни алатки за пребарување и филтрирање, што го подобрува корисничкото искуство при навигација низ објавите.

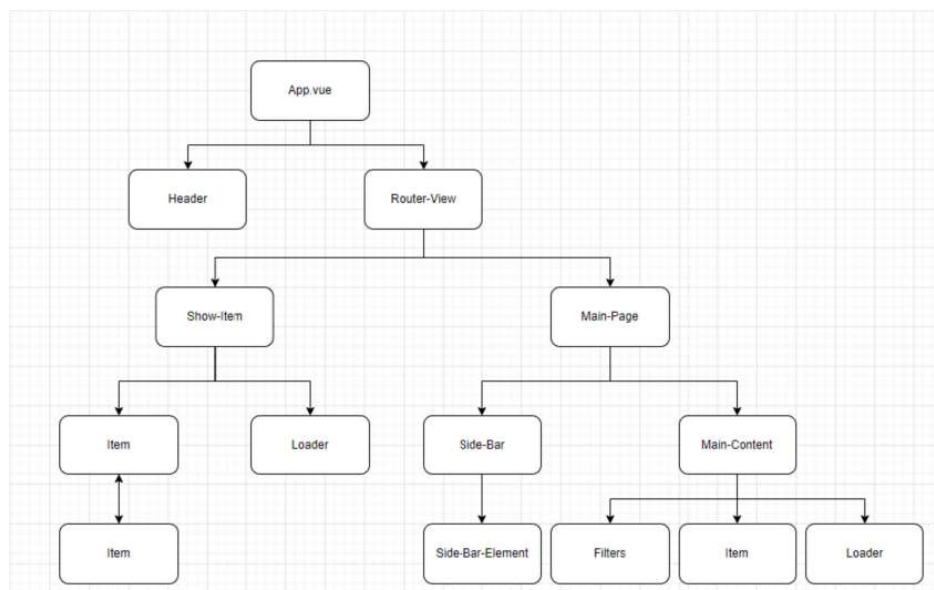
## Кеширање:

За да овозможиме офлајн пристап до последните 15 отворени објави, користиме **Local Storage** на прелистувачот. Ова ни овозможува да зачуваме податоци локално на уредот на корисникот, обезбедувајќи пристап до неодамна прегледаните објави дури и без интернет конекција.

Со комбинирање на овие технологии и алатки, создаваме апликација која е брза, ефикасна и пријатна за користење, обезбедувајќи подобро искуство при прелистување на содржините од Hacker News.

## 4. Имплементација

За добро да ја опишам имплементацијата на оваа апликација би сакал да навлезам длабоко во сите нејзини компоненти и да го објаснам начинот на кои тие функционираат и меѓусебно соработуваат за да се добие посакуваниот резултат.





## 4.1 App.vue

Оваа компонента претставува основата на една Vue апликација, организирана така што главниот дел од интерфејсот се состои од Header и динамички содржини прикажани преку router-view. Идејата е да се овозможи интерактивно искуство каде што корисникот може да пребарува и навигира низ различни страници, при што резултатите од пребарувањето остануваат достапни низ целата апликација.

```
<template>
  <Header :query="query" :updateQuery="updateQuery"/>
  <router-view :query="query"/>
</template>

<script>
import {defineComponent, ref} from "vue";
import Header from "@/components/header/Header.vue";

1+ usages  ⚙ Stefan
export default defineComponent( options: {
  components: { Header },
  setup() {
    1+ usages  ⚙ Stefan
    const query = ref( value: '' );

    const updateQuery = (value) => {
      query.value = value
    }

    return {
      query,
      updateQuery
    }
  }
});
</script>
```





Во `setup()` функцијата се дефинира реактивната променлива `query`, која служи за зачувување на тековниот внес на корисникот. Оваа вредност се менува преку функцијата `updateQuery`, овозможувајќи `Header` компонентата да го контролира нејзиното ажурирање. Благодарение на реактивноста на `Vue`, секоја промена на `query` автоматски ќе се одрази во другите компоненти што ја користат.

Клучниот дел од оваа структура е `<router-view>`, кој динамички прикажува различни содржини во зависност од рутата што е активна. Ова овозможува модуларен пристап во кој секоја страница може да го користи истиот `query` без потреба од сложени механизми за споделување на податоци. Така, кога корисникот пребарува од `Header`, резултатите веднаш се ажурираат во главната содржина.

Стилизирањето е минималистичко, при што `body` е поставен да зафаќа речиси цел екран. Ова гарантира конзистентен изглед на апликацијата, овозможувајќи прилагодливост на различни уреди.

На овој начин, компонентата функционира како централен хаб што ги поврзува `Header` и рутите, управувајќи со заедничката состојба на пребарувањето. Оваа архитектура ја прави апликацијата флексибилна, интуитивна и лесна за проширување.

## 4.2 Router.vue

---

За да функционира `router-view` од претходната компонента во фајлот `Router.js` ги дефинираме рутите на кои ќе ги мапира посакуваните погледи (views). Го користиме “`vue-router`” и ги дефинираме следните патеки: - “`/`” за главната страница каде се прелистуваат сите постови, односно компонентата `MainPage.vue` - “`/:id`” за прегледување на специфичен пост, односно компонентата `ShowItem.vue`

```
import { createRouter, createWebHistory } from 'vue-router';
import MainPage from "@/views/MainPage.vue";
import ShowItem from "@/views/ShowItem.vue";

const routes : [{path: string, component: Def...} = [
  { path: '/', component: MainPage, name: 'MainPage' },
  { path: '/:id', component: ShowItem, props: true, name: 'ShowItem' }
];

const router : Router = createRouter({ options: {
  history: createWebHistory(),
  routes,
}});

1+ usages  ± Stefan
export default router;
```



## 4.3 Header



Search  
Hacker News



Search titles by title, url or author



Компонентата Header се наоѓа на врвот на сите погледи и содржи поле за внесување каде што корисниците можат да пребаруваат. Како што пишува корисникот, влезот се отфрла и се емитува на App.vue за да се направи глобалното барање. Компонентата, исто така, вклучува основно брендирање и визуелна структура, помагајќи да се ориентираат корисниците при навигацијата на страницата. Неговата примарна техничка улога е да го изолира функционалноста за пребарување од главната логика на содржината. Благодарение на реактивноста на Vue, овој дизајн им овозможува на повиците за пребарување да активираат промени во други компоненти во реално време.

```
<script>
import Logo from "@assets/icons/hacker-news-logo.svg"
import SearchIcon from "@assets/icons/search-icon.svg"
import FiltersIcon from "@assets/icons/filter-icon.svg"

export default { Show usages  Stefan
  name: 'Header',
  props: {
    query: {
      type: String,
      required: true
    },
    updateQuery: {
      type: Object,
      required: true
    }
  },
  setup() {
    return {
      Logo,
      SearchIcon,
      FiltersIcon
    }
  }
}
</script>
```



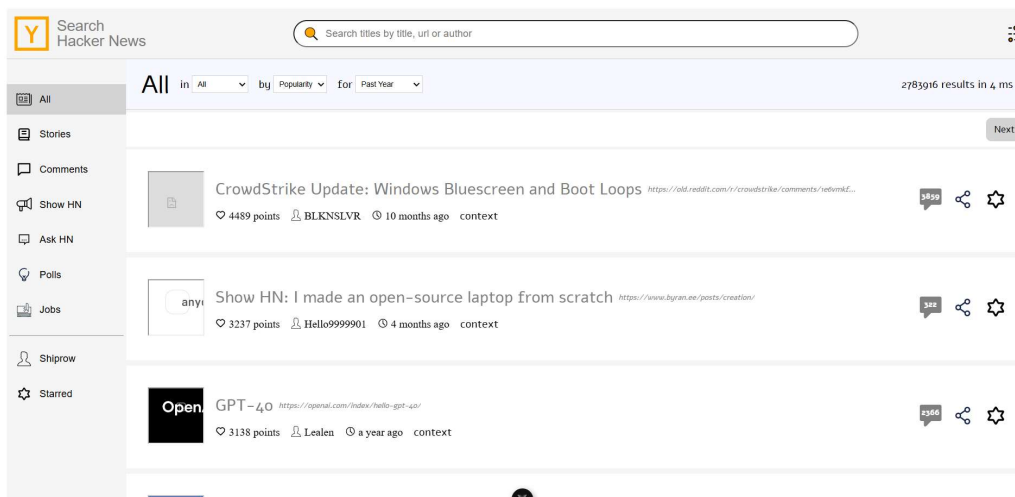
```
<template> Show component usages
<div id="header">
  <router-link to="/" class="logo-container">
    
    <p class="logo-text">Search<br/> Hacker News</p>
  </router-link>
  <div class="search-bar-container">
    
    <input type="text" placeholder="Search titles by title, url or author" class="search-text-field" :value="query"
      @input="event => updateQuery(event.target.value)"/>
  </div>
  
</div>
</template>
```

## 4.4 Views

Следно би сакал подлабоко да навлезам во логиката и функционалноста на двата различни погледи (views) кои ги мапиравме во нашиот рутер, анализирајќи како тие комуницираат со останатите делови од апликацијата, како ги обработуваат податоците и на кој начин ги прикажуваат на корисникот.

### 4.4.1 Main Page

MainPage, претставува централниот дел на апликацијата, каде што корисникот може да избира различни категории, сортирања и временски рамки за прикажување на содржините. Главната структура се состои од SideBar, кој им овозможува на корисниците да направат избори, и MainContent, каде што тие избори влијаат врз прикажаната содржина.





```
<template>
  <div id="main-page">
    <SideBar :selectedOption="selectedOption" :updateSelectedOption="updateSelectedOption"/>
    <MainContent :selectedOption="selectedOption" :updateSelectedOption="updateSelectedOption" :query="query"/>
  </div>
</template>

<script>
  ...

  1+ usages  ▲ Stefan
  export default defineComponent({
    name: "MainPage",
    components: {SideBar, MainContent},
    props: {
      query: {
        type: String,
        required: true
      }
    }
  },
```

```
  setup() {
    const savedOption = JSON.parse(localStorage.getItem(key: "selectedOption") || "{}");

    const selectedOption = ref({
      selectedCategory: savedOption.selectedCategory || {
        text: categoryOptions[0].text,
        value: categoryOptions[0].value
      },
      selectedSort: savedOption.selectedSort || {
        text: sortingOptions[0].text,
        value: sortingOptions[0].value
      },
      selectedTimeFrame: savedOption.selectedTimeFrame || {
        text: timeFrameOptions[0].text,
        value: timeFrameOptions[0].value
      }
    });

    1+ usages  ▲ Stefan
    const updateSelectedOption = (option) => {
      localStorage.setItem("selectedOption", JSON.stringify(option));
      selectedOption.value = option;
    }

    return {
      selectedOption,
      updateSelectedOption
    }
  }
},
```



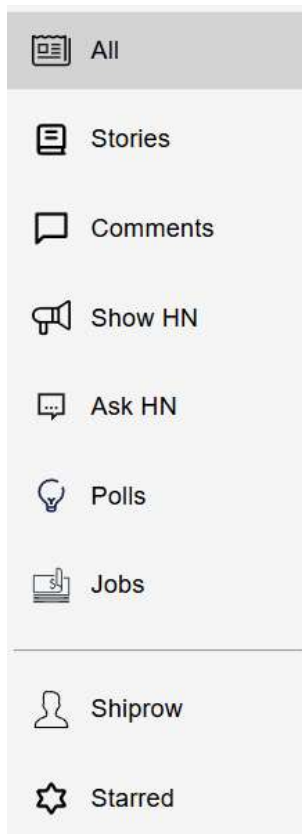
Во `setup()` функцијата, дефинирана е реактивната променлива `selectedOption`, која ги зачувува избраните критериуми за филтрирање. Оваа вредност се иницијализира од `localStorage`, што значи дека апликацијата ќе ги памети последните избори на корисникот дури и по повторно вчитување. Ако нема зачувани податоци, се користат основните вредности од три различни групи опции: категории, сортирање и временска рамка.

Кога корисникот ќе направи промена во `SideBar` или `MainContent`, функцијата `updateSelectedOption` го ажурира `selectedOption` и ги зачувува промените во `localStorage`. Со ова, апликацијата обезбедува конзистентност во корисничкото искуство, а изборите остануваат зачувани и достапни по повторно отворање на страницата.

Дополнително, `MainPage` ја прима `query prop`, што значи дека пребарувањето што корисникот го внесува на друго место во апликацијата ќе биде достапно и во оваа компонента. Ова овозможува интегрирање на пребарувањето со опциите за филтрирање, што го подобрува начинот на прегледување на содржината.

#### 4.4.1.1 Sidebar

---





SideBar компонентата служи како панел за избор на категории, кој им овозможува на корисниците да навигираат низ различни делови на апликацијата. Нејзината структура е организирана така што ги прикажува сите достапни елементи, поделени во две групи: главни категории и посебни опции што се одвоени со хоризонтална линија (<hr>).

Главната функционалност е управувањето со избраната категорија. Кога корисникот ќе кликне на некој елемент, се повикува функцијата `selectOption`, која го ажурира `selectedOption` објектот. Овој објект содржи информации за тековно избраната категорија, а промените автоматски се пренесуваат преку `updateSelectedOption`, што гарантира конзистентност низ целата апликација.

За секој елемент во листата, се прикажува `SideBarElement` компонента, која е задолжена за визуелното претставување на опциите. Ако избраната категорија се совпаѓа со тековниот елемент, `isSelected` својството е поставено на `true`, што овозможува стилизирање на активната опција.

Дизајнот е едноставен, но ефикасен. SideBar има фиксна ширина и висина, поставена така што го зафаќа поголемиот дел од екранот, но не доминира над главната содржина. Благодарение на `position: sticky`, sidebar-от останува фиксиран при скролирање, овозможувајќи корисниците лесно да се движат низ опциите без да го губат од вид.

На овој начин, SideBar обезбедува интуитивен начин за филтрирање на содржината, а неговата интерактивност гарантира дека избраните критериуми веднаш ќе се применат на главната содржина на апликацијата.

```
setup(props) {  
  // usages: Stefan  
  const selectOption = (value, text) => {  
    let updatedOption;  
    updatedOption = {  
      ...props.selectedOption,  
      selectedCategory: {  
        text: text,  
        value: value  
      },  
    };  
    props.updateSelectedOption(updatedOption);  
  };  
  
  return {  
    elements,  
    selectOption,  
  };  
},  
});  
</script>
```





```
<template>
  <div id="sidebar-container">
    <div
      v-for="(element, index) in elements.slice(0, elements.length - 2)"
      :key="index"
      @click="selectOption(element.value, element.text)"
    >
      <SideBarElement :element="element" :isSelected="selectedOption.selectedCategory.value === element.value"/>
    </div>
    <hr class="separator"/>
    <div
      v-for="(element, index) in elements.slice(elements.length - 2)"
      :key="index"
    >
      <SideBarElement :element="element"/>
    </div>
  </div>
</template>

<script>
import ...

1> usages  ⚡ Stefan
export default defineComponent({ options: {
  name: "SideBar",
  components: {SideBarElement},
  props: {
    selectedOption: {
      type: Object,
      required: true
    },
    updateSelectedOption: {
      type: Object,
      required: true
    }
  }
}
```

#### 4.4.1.1.1 SideBarElement



SideBarElement, претставува индивидуален елемент во sidebar-от, кој прикажува икона и текст за секоја категорија или опција. Нејзината главна улога е да овозможи визуелно претставување на опциите што корисникот може да ги избере, истовремено обезбедувајќи индикација за активната селекција.

Во нејзината структура, главниот `<div>` има динамичка класа што се менува врз основа на `isSelected` prop. Ако дадениот елемент е избран, му се додава класата `selected`, што му дава поинаков стил, овозможувајќи му на корисникот јасно да види кој елемент моментално е активен.

Визуелниот дел е едноставен, но функционален. Иконата се прикажува како слика (`<img>`), додека името на категоријата се прикажува во `<span>`. Двата елемента се распоредени хоризонтално и имаат дефинирани стилови кои обезбедуваат конзистентен изглед.



Дизајнот е интерактивен – кога корисникот ќе го помине покажувачот на глумчето над елементот, позадината ја менува бојата (:hover ефект), а кога елементот е избран, се нагласува со `background-color: lightgrey`. Ова ја подобрува корисничката интеракција, овозможувајќи јасно и брзо препознавање на активната опција.

Со овој пристап, `SideBarElement` го олеснува процесот на селекција на категории, обезбедувајќи визуелно пријатно и интуитивно искуство за корисниците.

```
<template>
  <div :class="['sidebar-element-container', {'selected': isSelected}]">
    
    <span class="element-text">{{ element.text }}</span>
  </div>
</template>

<script>
1+ usages  ▲ Stefan
export default {
  name: "SideBarElement",
  props: {
    element: {
      type: Object,
      required: true
    },
    isSelected: {
      type: Boolean
    }
  }
}
</script>
```





Универзитет „Св. Кирил и Методиј“ во Скопје  
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И  
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

#### 4.4.1.2 MainContent

All


in All

by Popularity

for Past Year

2783916 results in 4 ms

Next




CrowdStrike Update: Windows Bluescreen and Boot Loops

<https://old.reddit.com/r/crowdstrike/comments/1edvmkf...>

4489 points BLKNSLVR 10 months ago context

3859




Show HN: I made an open-source laptop from scratch

<https://www.bryan.ee/posts/creation/>

3237 points Hello9999901 4 months ago context

322



GPT-4o

<https://openai.com/index/hello-gpt-4o/>

3138 points Lealen 1 year ago context

2366

MainContent е централниот дел на апликацијата, каде што се прикажуваат резултатите од пребарувањето или филтрирањето. Нејзината главна функција е да прикаже список на статии или објави врз основа на тековните критериуми на корисникот, како што се избраната категорија, начин на сортирање и временски интервал.

Во нејзиниот шаблон, најпрво се прикажува Filters компонентата, која му овозможува на корисникот да ги промени филтрите. Под неа, доколку уредот е офлајн, се појавува известување дека се прикажуваат зачувани податоци. Главниот дел е списокот со објави (Item компоненти), каде што се наоѓаат содржините што ги враќа API-то. Доколку нема резултати, се прикажува порака која известува за тоа, а ако податоците сè уште се вчитуваат, се прикажува Loader компонента. На дното се наоѓаат копчињата за навигација меѓу страниците, кои овозможуваат корисникот да се движи низ резултатите.

Stories

in Stories


by Popularity

for Past Year

2783916 results in 4 ms

You are currently offline, showing cached items.

Next



CrowdStrike Update: Windows Bluescreen and Boot Loops

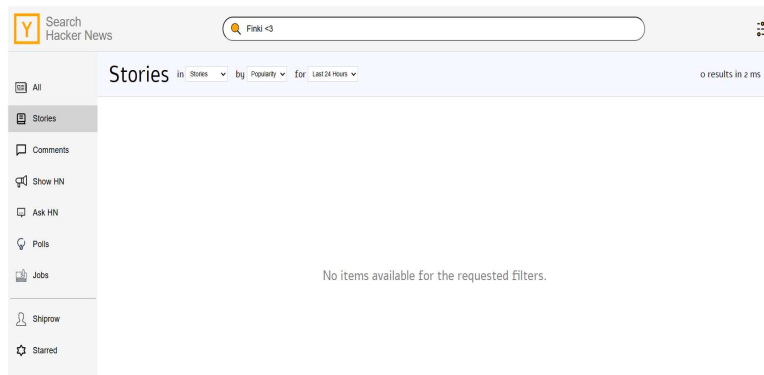
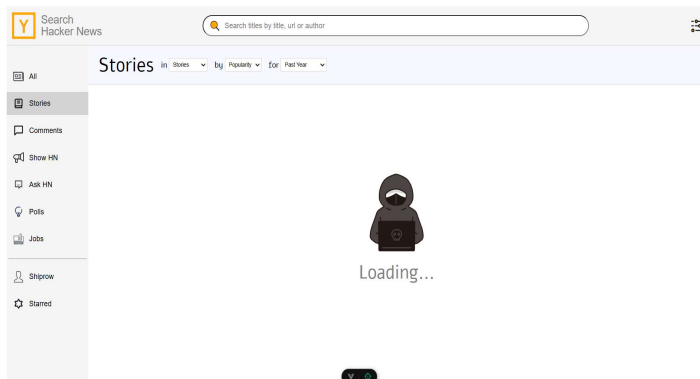
<https://old.reddit.com/r/crowdstrike/comments/1edvmkf...>

4489 points BLKNSLVR 10 months ago context

3859



Универзитет „Св. Кирил и Методиј“ во Скопје  
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И  
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО



Во логиката на компонентата, се користат `ref` променливи за управување со податоците, вчитувањето и офлајн режимот. `fetchItems` функцијата го повикува API-то за да ги добие резултатите, а во случај на грешка ги користи кешираните податоци од `localStorage`. Следењето (`watch`) на `selectedOption` и `query` овозможува автоматско ажурирање на резултатите кога корисникот ги менува критериумите.

```
<template>
  <div id="main-content-container">
    <Filters :selectedOption="selectedOption" :updateSelectedOption="updateSelectedOption" :processing-info="processingInfo"/>
    <p v-if="offlineMode" class="offline-text">You are currently offline, showing cached items.</p>
    <div v-if="!isLoading && items.length > 0" class="item">
      <div class="pages-container" v-if="currentPage + 1 !== totalPages || currentPage !== 0">
        <div>
          <button class="page-button text" v-if="currentPage !== 0" @click="fetchItems( page, currentPage - 1)">
            Previous
          </button>
        </div>
        <button class="page-button text" v-if="currentPage + 1 !== totalPages" @click="fetchItems( page, currentPage + 1)">
          Next
        </button>
      </div>
      <Item v-for="item in items" :key="item.objectID" :item="item" :query="query"/>
    </div>
    <div v-if="!isLoading && items.length === 0" class="filler-container">
      <p class="no-items-text text">No items available for the requested filters.</p>
    </div>
    <div v-if="isLoading" class="filler-container">
      <Loader/>
    </div>
  </div>
</template>
```



```
const fetchItems = async (page) => {
  isLoading.value = true;
  currentPage.value = page
  Api.searchItems(props.selectedOption, props.query, page)
    .then(response => {
      processingInfo.value = {
        numElements: response.nbHits,
        timeInMS: response.processingTimeMS
      }
      items.value = response.hits;
      localStorage.setItem("cachedItems", JSON.stringify(response));
      offlineMode.value = false;
      totalPages.value = response.nbPages;
    })
    .catch(() => {
      items.value = JSON.parse(localStorage.getItem( key: "cachedItems")).hits;
      offlineMode.value = true;
    })
    .finally( onFinally: () => {
      isLoading.value = false;
    });
};

watch( source: () => props.selectedOption, cb: () => {
  fetchItems( page: 0);
}, options: { deep: true });

watch( source: () => props.query, cb: () => {
  fetchItems( page: 0);
});

onMounted( hook: () => fetchItems( page: 0));
```



```
export default {  
  name: "Main-Content",  
  components: {Loader, Filters, Item },  
  props: {  
    selectedOption: {  
      type: Object,  
      required: true  
    },  
    updateSelectedOption: {  
      type: Object,  
      required: true  
    },  
    query: {  
      type: String,  
      required: true  
    }  
  },  
  setup(props) {  
    const items = ref( value: []);  
    const processingInfo = ref()  
    const isLoading = ref( value: true);  
    const offlineMode = ref( value: false);  
    const currentPage = ref( value: 0);  
    const totalPages = ref( value: 0);  
  }  
}
```

#### 4.4.1.2.1 Filters



Компонентата Filters претставува клучен елемент во архитектурата на апликацијата, овозможувајќи динамично филтрирање на податоците врз основа на различни критериуми. Оваа компонента се состои од повеќе изборни полиња (select), кои на корисникот му дозволуваат да ги прилагоди резултатите што ги гледа.

Основните критериуми за филтрирање вклучуваат категорија, начин на сортирање и временска рамка, што ги прави прикажаните податоци поприлагодливи на потребите на корисникот.



```
<template>
  <div id="filters-bar">
    <div class="filters-container">
      <p class="selected-tag">{{ selectedOption.selectedCategory.text }}</p>
      <div class="select-filter-container">
        <p class="text">in</p>
        <select class="select-filter" v-model="selectedOption.selectedCategory.value"
          @change="handleFilterChange(selectedOption.selectedCategory.value, 'selectedCategory', 'categoryOptions')">
          <option v-for="option in getOptions( type: 'categoryOptions' )" :key="option.value" :value="option.value">
            {{ option.text }}
          </option>
        </select>
      </div>
    </div>
    <div class="select-filter-container">
      <p class="text">by</p>
      <select class="select-filter" v-model="selectedOption.selectedSort.value"
        @change="handleFilterChange(selectedOption.selectedSort.value, 'selectedSort', 'sortingOptions')">
        <option v-for="option in getOptions( type: 'sortingOptions' )" :key="option.value" :value="option.value">{{ option.text }}</option>
      </select>
    </div>
    <div class="select-filter-container">
      <p class="text">for</p>
      <select class="select-filter" v-model="selectedOption.selectedTimeFrame.value"
        @change="handleFilterChange(selectedOption.selectedTimeFrame.value, 'selectedTimeFrame', 'timeFrameOptions')">
        <option v-for="option in getOptions( type: 'timeFrameOptions' )" :key="option.value" :value="option.value">
          {{ option.text }}
        </option>
      </select>
    </div>
    <div>
      <p v-if="processingInfo" class="text">{{processingInfo.numElements}} results in {{processingInfo.timeInMS}} ms</p>
    </div>
  </div>
</template>
```

```
setup(props) {
  1+ usages  ⚡ Stefan
  const handleFilterChange = (newValue, type, selectedArray) => {
    let optionsTMP = props.selectedOption;
    optionsTMP[type] = {
      text: getOptions(selectedArray).find(option => option.value === optionsTMP[type].value).text,
      value: optionsTMP[type].value
    }
    props.updateSelectedOption(optionsTMP);
  }

  1+ usages  ⚡ Stefan
  const getOptions = (type) => {
    if (type === 'categoryOptions')
      return categoryOptions
    if (type === 'sortingOptions')
      return sortingOptions
    if (type === 'timeFrameOptions')
      return timeFrameOptions
  }
  return {
    handleFilterChange,
    getOptions
  }
}
</script>
```

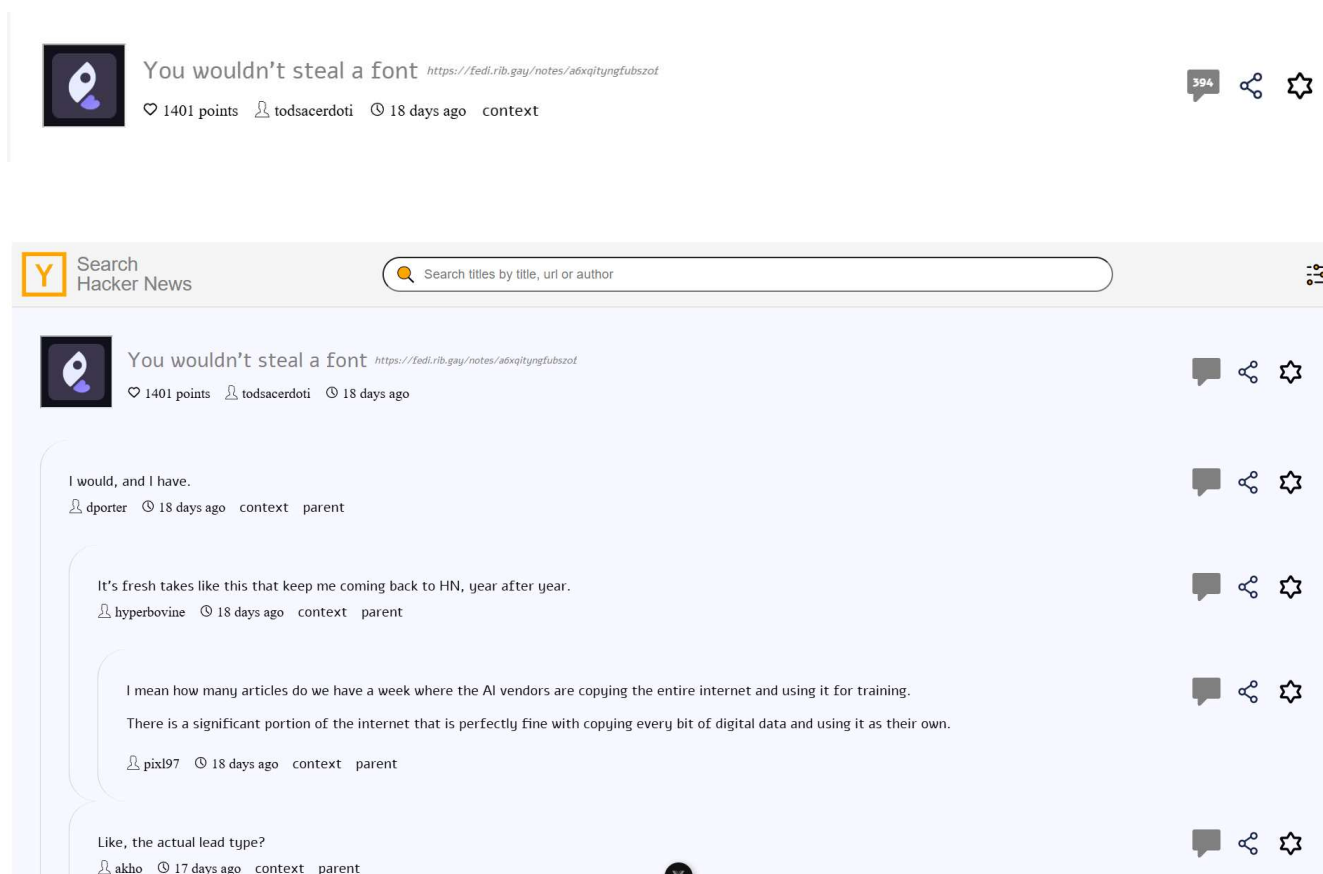


Од техничка перспектива, компонентата `Filters` работи со реактивен објект `selectedOption`, кој ја содржи тековната состојба на филтрите. Кога корисникот ќе направи промена во некое од `select` полињата, се повикува методот `handleFilterChange`, кој ја променува избраната опција и ја испраќа назад до родителската компонента преку `updateSelectedOption`. Овој пристап осигурува дека целата апликација е усогласена со тековните филтрирани податоци.

Во однос на имплементацијата, компонентата користи неколку надворешни модули за дефинирање на опциите, како што се `categoryOptions`, `sortingOptions` и `timeFrameOptions`. Ова ја прави кодната база почиста и поорганизирана, бидејќи сите можни вредности за филтрирање се чуваат надвор од самата компонента. Покрај тоа, методот `getOptions` ги повикува овие модули кога е потребно, со што се избегнува непотребно дуплирање на податоци.

Исто така во `Local Storage` се зачувува последната состојба на филтрите така што на следната посета на порталот прикажани ќе бидат артикли со истите филтри. Надежно да се намали непотребно кликање, чекање и дополнителни повици.

#### 4.4.1.2.2 Item







Компонентата Item претставува централна единица за прикажување на индивидуални елементи во апликацијата, овозможувајќи прикажување на вести, коментари или прашања од платформата Hacker News. Таа е дизајнирана да обезбеди структуриран и визуелно привлечен приказ на содржината.

Основната функционалност на оваа компонента е да прикаже информации како насловот, авторот, бројот на поени, бројот на коментари и времето на објавување. Доколку елементот содржи URL, истиот се вградува во iframe, овозможувајќи корисникот да добие визуелен преглед без да мора веднаш да ја напушти апликацијата.

```
<template>
  <div id="item-container">
    <div :class="['item', {'centered': !item.text}]">
      <div :class="['item-content', {'centered': !item.text}]">
        <a v-if="item.url" target="_blank" :href="item.url" class="preview-container">
          <div>
            <iframe src="item.url" scrolling="no" sandbox="" class="site-preview"/>
          </div>
        </a>
      <div class="item-info-container">
        <p class="title text">
          <span @click="item.objectID && navigateToItem(item.objectID)" class="title-link" v-if="item.title"
            v-html="highlightText(item.title)"></span>
          <a :href="item.url" class="redirect-link-text text" v-if="item.url" target="_blank"
            v-html="highlightText(item.url)"></a>
        </p>
        <p v-if="item.text || item.comment_text" class="text comment-text"
          v-html="highlightText(item.text || item.comment_text)"></p>
        <div class="item-stats-container">
          <div class="item-stat" v-if="item.points">
            
            <span>{{ item.points }} points</span>
          </div>
          <div class="item-stat">
            
            <span>{{ item.author }}</span>
          </div>
          <div v-if="item.created_at" class="item-stat">
            
            <span>{{ moment.unix(item.created_at).fromNow() }}</span>
          </div>
          <div v-if="item.story_title || (item.story_id !== item.id)" @click="navigateToItem(item.story_id)"
            class="parent-story-name">
            <span class="text">{{ item.story_title ? 'on: ' + item.story_title : 'context' }}</span>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>

<div v-if="item.parent_id" @click="navigateToItem(item.parent_id)" class="parent-story-name">
  <span class="text">parent</span>
</div>
</div>
</div>
<div class="icons-container">
  <div class="comments-icon-container" @click="navigateToStory(item.story_id)">
    <span class="num-comments text">{{ item.num_comments }}</span>
    
  </div>
  
  
</div>
</div>
<div v-if="item.children && typeof item.children[0] === 'object'" v-for="(child) in item.children"
  :key="child.objectID" class="child">
  <Item :item="child" :query="query"/>
</div>
</div>
</template>
```



```
export default {
  name: "Item",
  props: {
    item: {
      type: Object,
      required: true
    },
    query: {
      type: String,
      required: true
    }
  },
  setup(props) {
    const router = useRouter()
    const {item} = props;

    // usage: <Item>
    const navigateToItem = (objectId) => {
      router.push(`/ ${objectId}`);
    }

    // usage: <Item url="">
    const navigateToURL = (url) => {
      router.replace(url)
    }

    // usage: <Item text="">
    const highlightText = (text) => {
      let queryTMP = props.query;
      if (!queryTMP)
        return text;
      const words = queryTMP.split(' ').map(word => word.replace(/([a-zA-Z0-9]+)/g, '$1'));
      const regex = new RegExp(`\\b(${words.join('|')})\\b`, 'g');
      return text.replace(regex, `<span class="highlight">$1</span>`);
    };
  }
};
```

Во техничка смисла, компонентата прифаќа два props параметри: item, кој го содржи објектот со сите податоци за прикажување, и query, кој се користи за означување на пребаруваните термини во насловите и текстовите на објавите. Реактивната функционалност е имплементирана во setup методот, каде што се дефинираат различни функции за управување со интеракциите. Vue Router се користи за динамично менување на страниците, овозможувајќи лесна навигација кон детали за одредена објава, нејзиниот родителски коментар или почетниот пост од дискусијата.

Прикажувањето на содржината е прилагодено за различни типови на податоци. Доколку објавата има наслов, тој се прикажува како кликлив елемент што води до целосниот приказ на содржината. Покрај тоа, ако постои URL, истиот се прикажува веднаш до насловот, а корисниците можат да кликнат на него за да ја посетат оригиналната страница. Во случај кога објавата е коментар, нејзиниот текст е истакнат како коментарска содржина, со јасна визуелна разлика во однос на статиите или линковите.

Еден од клучните аспекти на компонентата е обработката на статистички податоци поврзани со објавите. Овие податоци се прикажани во форма на мали икони придружени со текст, што го олеснува нивното разбирање. На пример, бројот на поени е претставен со икона на срце, додека времето на објавување е прикажано преку момент-базирана обработка на timestamp-от. Доколку елементот е дел од подолга дискусија, се прикажува линк што води до неговиот родителски коментар или почетниот пост на темата.





За подобрување на корисничкото искуство, компонентата имплементира функција за означување на текст, која ги нагласува зборовите од пребарувањето во рамките на резултатите. Оваа функција креира регуларен израз базиран на внесениот query, овозможувајќи динамично означување на релевантните термини во насловите, линковите и коментарите.

Компонентата исто така поддржува рекурзивно рендерирање, што значи дека ако објавата има подкоментари, тие ќе бидат прикажани преку вградување на истата компонента во нејзината структура. Ова овозможува хиерархиски приказ на дискусиите, со јасна визуелна разлика помеѓу главните објави и нивните одговори. Секој подкоментар е вовлечен со лев маргинален раб, обезбедувајќи интуитивна навигација низ различните нивоа на дискусијата.

#### 4.4.1.2.3 Loader



Loading...

Компонентата Loader служи за прикажување на екран за вчитување додека апликацијата обработува или вчитува податоци. Таа обезбедува визуелна индикација за корисникот дека моментално се извршува процес што бара време, како што е преземање на вести, коментари или други ресурси преку API. Со оваа компонента се подобрува корисничкото искуство, бидејќи им овозможува на корисниците да знаат дека системот е активен и дека нивното барање се обработува.

Во мојата апликација искористив статички Loader без анимации, со цел да се придржам кон едноставниот и минималистички кориснички интерфејс инспириран од Hacker News. Наместо визуелно привлечни елементи или ефекти, Loader-от е прикажан како обичен текст "Loading..." и слика од "хакер" поставен на центарот од екранот. Овој пристап ја нагласува содржината, а не дизајнот, и овозможува фокусирано и брзо корисничко искуство без одвлекување на вниманието.

Иако е едноставен, овој статички Loader е доволен за да го информира корисникот дека апликацијата работи во позадина, без да го нарушува минималистичкиот изглед или да внесува дополнителни визуелни елементи што не се неопходни.

```
<template>
  <div id="loader">
    
    <p class="loading-text">Loading...</p>
  </div>
</template>

<script>
import HackerIcon from "@assets/icons/hacker-icon.svg"

/* usage: <Stefan>
export default {
  name: "Loader",
  setup() {
    return {
      HackerIcon
    }
  }
}
</script>
```



## 4.5 Аpi

Овој модул користи Axios за да овозможи комуникација со Algolia API, кој обезбедува податоци за Hacker News. Креира Axios инстанца со основен baseURL поставен на <http://hn.algolia.com/api/v1>, а сите барања имаат Content-Type: application/json во заглавието. Ова овозможува лесно извршување на HTTP барања кон API-то со стандардизирани поставки.

```
import axios from 'axios';

const apiClient : AxiosInstance = axios.create({
  baseURL: 'http://hn.algolia.com/api/v1',
  headers: {
    'Content-Type': 'application/json',
  },
});
```

```
export default {
  searchItems (selectedOptions, query, page) : any {
    const path : string = selectedOptions.selectedSort.value === 'date' ? '/search_by_date' : '/search';

    let timeInSeconds : number = Math.floor(Date.now() / 1000);

    switch (selectedOptions.selectedTimeFrame.value) {
      case 'last_24_hours':
        timeInSeconds -= 24 * 60 * 60;
        break;
      case 'past_week':
        timeInSeconds -= 7 * 24 * 60 * 60;
        break;
      case 'past_month':
        timeInSeconds -= 30 * 24 * 60 * 60;
        break;
      case 'past_year':
        timeInSeconds -= 365 * 24 * 60 * 60;
        break;
      default:
        timeInSeconds = null;
    }

    const params : {} = {
      query: query ? query : undefined,
      tags: selectedOptions.selectedCategory.value !== 'all' ? selectedOptions.selectedCategory.value : undefined,
      numericFilters: timeInSeconds ? `created_at_i>${timeInSeconds}` : undefined,
      page: page
    };

    return apiClient.get(path, { params }).then(response : AxiosResponse<any> => response.data);
  },
  searchSingleItem(id) : any {
    return apiClient.get(`/items/${id}`).then(response : AxiosResponse<any> => response.data);
  }
}
```

Главната функција, `searchItems`, се користи за пребарување постови врз основа на дадени критериуми. Прво, таа одредува која рута да се користи – `/search_by_date` ако корисникот сака резултатите да бидат сортирани според датум, или `/search` за сортирање според релевантност. Потоа, врз основа на избраниот временски период, пресметува временски отпечаток (timestamp)



кој се користи за филтрирање на резултатите. Ако е избрано „last\_24\_hours“, „past\_week“, „past\_month“ или „past\_year“, од тековното време се одзема соодветниот број секунди, додека ако нема временско ограничување, `timeInSeconds` останува `null`.

Конечниот објект `params` ги содржи барањата што ќе се испратат до API-то. Ако има внесен `query`, тој се додава во параметрите. Ако е избрана специфична категорија на содржина (на пр. „story“, „comment“), таа се додава како `tag` во барањето. Доколку е дефиниран временскиот филтер, се додава `numericFilters`, кој проверува дали резултатите се креирани по одредениот временски праг. Исто така, `page` параметарот овозможува пагинација, испраќајќи резултати за соодветната страница.

Дополнително, `searchSingleItem` овозможува пребарување на единечен елемент според неговиот ID. Оваа функција испраќа GET барање до `/items/{id}` и враќа податоци за тој елемент. Ова е корисно за добивање на детали за одредена приказна или коментар во Hacker News. Обете функции враќаат `Promise`, што значи дека може да се користат со `.then()` или `async/await` за обработка на одговорот.

## 5. Заклучок и идни подобрувања

---

Овој проект претставува подобрен и модернизиран начин за прелистување на содржините од Hacker News, со посебен фокус на корисничкото искуство и визуелниот изглед. Користејќи `Vue.js` како основа за фронтенд делот и интегрирајќи ги официјалните API-ја за преземање на податоци, успеавме да создадеме брза, ефикасна и интуитивна апликација која нуди полесен пристап до најновите и најпопуларните објави. Дополнително, со користење на `Local Storage` овозможивме кеширање на последните 15 отворени објави, што овозможува офлајн пристап и континуирано прелистување без потреба од интернет конекција.

Во иднина, постојат различни можности за подобрување на проектот, со цел да се зголеми неговата функционалност и привлечност за корисниците. Воведувањето на можност за зачувување на омилените објави би им овозможило на корисниците подобро организирање на содржините што им се интересни. Унапредувањето на корисничкиот интерфејс со поддршка за темна режим и прилагодливи теми би донело поголема флексибилност и удобност при користењето на апликацијата.

Дополнително, интеграцијата на систем за известувања би овозможила корисниците да добиваат информации за нови и значајни објави во реално време. Со овие подобрувања, проектот би можел да прерасне во уште покорисна и подинамична платформа за сите љубители на Hacker News.



Универзитет „Св. Кирил и Методиј“ во Скопје  
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И  
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

## 6. Користени технологии и код

---

[1] Vue.js - <https://vuejs.org/>

[2] Algolia API - <https://hn.algolia.com/api>

[3] Hacker News API - <https://github.com/HackerNews/API>

[4] Код - <https://github.com/203180/Hacker-News>