Небольшой мануал по работе с Vue.JS с минимально необходимым набором пунктов для успешного выполнения лабораторной работы номер 4.

0. Инициализация проекта.

Начать, разумеется, стоит с инициализации проекта. Вся работа с Vue и зависимостями будет происходить через NPM (Node Package Manager). Поэтому, первым делом необходимо установить node.js и npm на свой компьютер. Все инструкции для установки можно найти здесь.

После успешной установки node и npm (ориентироваться следует на последние LTS-версии, который на момент написания соответствуют 14.5.1 и 6.14.8, соответственно), можно переходить к установке vue.

Делается это выполнением следующей команды в терминале (важное уточнение: в unix-системах нужно вызывать эту команду с правами суперпользователя):

```
$ npm install -g @vue/cli
```

После выполнения этой команды на вашем компьютере будет установлен vue и vue-cli (инструмент, необходимый для того, чтобы облегчить работу с инициализацией проекта на vue).

Теперь, когда vue и vue-cli глобально установлены на вашем компьютере можно приступать к инициализации проекта:

```
$ vue create 03_vue_examples
```

* 03_vue_examples - название приложения.

После чего поступит предложение о выборе варианта установки:

```
Vue CLI v4.5.9
? Please pick a preset: (Use arrow keys)
❯ Default ([Vue 2] babel, eslint)
```

```
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
  Manually select features
```

Следует выбрать последний вариант.

Из всего предложенного списка дополнительно пригодится только Router:

```
Vue CLI v4.5.9
? Please pick a preset: Manually select features
? Check the features needed for your project:
❯◉ Choose Vue version
 ◉ Babel
 ◯ TypeScript
 ◯ Progressive Web App (PWA) Support
 ◉ Router
 ◯ Vuex
 ◯ CSS Pre-processors
 ◉ Linter / Formatter
 ◯ Unit Testing
 ◯ E2E Testing
```

Из списка версий нужно выбрать вторую:

```
Vue CLI v4.5.9
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version,
Babel, Router, Linter
? Choose a version of Vue.js that you want to start the project
with (Use arrow keys)
❯ 2.x
  3.x (Preview)
```

В роутере будет использоваться history mode, поэтому нужно нажать Enter:

```
Vue CLI v4.5.9
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version,
Babel, Router, Linter
? Choose a version of Vue.js that you want to start the project
with 2.x
```

```
? Use history mode for router? (Requires proper server setup for
index fallback in production) (Y/n)
```

В качестве линтера нужно выбрать стандартную конфигурацию:

```
Vue CLI v4.5.9
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version,
Babel, Router, Linter
? Choose a version of Vue.js that you want to start the project
with 2.x
? Use history mode for router? (Requires proper server setup for
index fallback in production) Yes
? Pick a linter / formatter config:
  ESLint with error prevention only
  ESLint + Airbnb config
❯ ESLint + Standard config
  ESLint + Prettier
```

В качестве дополнительной настройки линтера всё нужно оставить без изменений, соответственно, нужно просто нажать Enter:

```
Vue CLI v4.5.9
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version,
Babel, Router, Linter
? Choose a version of Vue.js that you want to start the project
with 2.x
? Use history mode for router? (Requires proper server setup for
index fallback in production) Yes
? Pick a linter / formatter config: Standard
? Pick additional lint features: (Press <space> to select, <a> to
toggle all, <i> to invert selection)
❯◉ Lint on save
 ◯ Lint and fix on commit
```

В следующем пункте уточняется необходимость разнесения всех конфигурационных файлов, такой вариант наиболее предпочтителен для обеспечения удобной возможности последующей настройки:

```
Vue CLI v4.5.9
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version,
Babel, Router, Linter
? Choose a version of Vue.js that you want to start the project
with 2.x
? Use history mode for router? (Requires proper server setup for
index fallback in production) Yes
? Pick a linter / formatter config: Standard
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? (Use
arrow keys)
❯ In dedicated config files
  In package.json
```

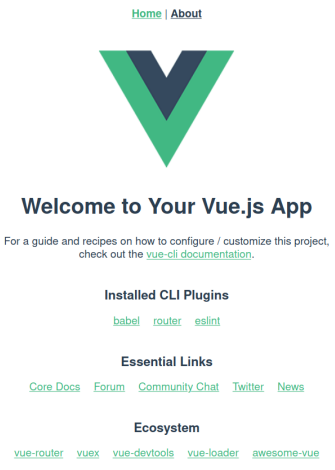
После этого остаётся только немного подождать, пока проект создастся. Теперь стоит попробовать запустить проект, чтобы проверить, что всё прошло хорошо, для этого нужно перейти в директорию проекта:

```
$ cd 03_vue_examples
```

И запустить проект:

```
$ npm run serve
```

Если всё прошло хорошо, после запуска по адресу http://localhost:8080, в браузере появится следующее:



1. Компоненты, роутинг

Видео о том, как работает роутинг на фронтенде можно посмотреть здесь.

Почитать как это работает во vue можно здесь.

В этом мануале будет разобран довольно простой пример работы с роутингом, которого, тем не менее, должно быть достаточно для выполнения лабораторной работы.

В файле конфигурации роутера (src/router/index.js) есть массив со всеми путями, которые используются в приложении:

```
import Vue from 'vue'
import VueRouter from 'vue-router'

// импортируем все представления, по
// которым будем навигировать пользователя
import Home from '@/views/Home.vue'
import About from '@/views/About.vue'

Vue.use(VueRouter)

// заводим массив с роутами
const routes = [
 {
   path: '/',
   name: 'Home',
   component: Home
 },
 {
   path: '/about',
   name: 'About',
   component: About
 }
]

// создаём новый экземпляр класса
// VueRouter, с необходимыми параметрами
const router = new VueRouter({
 mode: 'history',
 base: process.env.BASE_URL,
 routes
})
```

```
// экспортируем сконфигурированный роутер
export default router
```

Создадим отдельное представление в директории src/views/ и несколько компонент в директории src/components, чтобы разобраться с тем, как работает компонентый подход.

Важно не забывать, что компоненты и представления стоит отделять друг от друга. Представление — это страница, которую будет видеть конечный пользователь, переход в роутере должен быть именно на представление, а компонента — это то, что является отдельной частью представления. Частью, которая отвечает за свой кусочек логики (например, навбар или меню).

Создадим компоненту GreetingCard.vue:

```
<template>
  <h1>Привет!</h1>
</template>

<script>
export default {
 name: 'GreetingCard'
}
</script>

<style>
</style>
```

Весь код компоненты во Vue логически можно разделить на три части: template, script, style. Template отвечает за вёрстку, html-код. Script — за все скрипты, которые используются для добавления какой-либо логики (запросы к API, вывод информации и так далее). Style — за все стили компоненты.

Теперь создадим представление Greeting.vue:

```
<template>
 <section>
   <greeting-card />
```

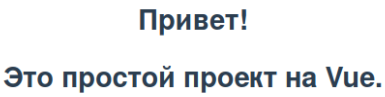
```
    <h2>Это простой проект на Vue.</h2>
  </section>
</template>

<script>
import GreetingCard from '@/components/GreetingCard'

export default {
  name: 'Greeting',

  components: {
    GreetingCard
  }
}
</script>

<style>
</style>
```

Добавим это представление в src/router/index.js:

```
import Vue from 'vue'
import VueRouter from 'vue-router'

// импортируем все представления, по
// которым будем навигировать пользователя
import Greeting from '@/views/Greeting.vue'

Vue.use(VueRouter)

// заводим массив с роутами
const routes = [
  {
    path: '/',
    name: 'Greeting',
    component: Greeting
  }
]

// создаём новый экземпляр класса
// VueRouter, с необходимыми параметрами
```

```
const router = new VueRouter({
 mode: 'history',
 base: process.env.BASE_URL,
 routes
})

// экспортируем сконфигурированный роутер
export default router
```

Откроем приложение по адресу http://localhost:8080:



Всё прошло хорошо, можно переходить к следующему пункту.

Примечание, код главной компоненты src/App.vue был незначительно изменён:

```
<template>
 <main id="app">
   <router-view/>
 </main>
</template>

<style>
#app {
 font-family: Avenir, Helvetica, Arial, sans-serif;
 -webkit-font-smoothing: antialiased;
 -moz-osx-font-smoothing: grayscale;
 text-align: center;
 color: #2c3e50;
}

#nav {
 padding: 30px;
```

```
}

#nav a {
 font-weight: bold;
 color: #2c3e50;
}

#nav a.router-link-exact-active {
 color: #42b983;
}
</style>
```

2. Работа с v-model, props, v-if, v-for, methods, created

Теперь нужно научиться работать с v-model, для этого заведём input в представлении Greeting:

```
  <form @submit.prevent="submitUsername">
    <label for="username">
      Введите своё имя:
    </label>

    <input
      v-model="username"
      id="username"
      name="username"
      type="text"
      placeholder="Иван Иванов"
    />
  </form>
```

Добавим метод data() в секцию script, после объекта components:

```
data: () => ({
  username: ''
})
```

Всё содержимое этого метода отвечает за переменные, которые можно будет использовать внутри этой конкретной компоненты.

Как можно заметить, в input, который был добавлен на страницу, есть атрибут v-model со значением username, которое соответствуют названию переменной в data(). А у самой формы есть атрибут @submit.prevent, который отвечает за вызов метода submitUsername, прерывая стандартное событие submit.

Теперь код надо немного изменить, чтобы иметь возможность передачи сохранённого значения в компоненту GreetingCard. Для начала следует обновить код представления Greeting:

```
<template>
 <section>
   <greeting-card />

   <h2>Это простой проект на Vue.</h2>

   <form
     @submit.prevent="submitUsername"
     ref="form"
   >
     <label for="username">
       Введите своё имя:
     </label>

     <input
       v-model="username"
       id="username"
       name="username"
       type="text"
       placeholder="Иван Иванов"
     />
   </form>
 </section>
</template>

<script>
import GreetingCard from '@/components/GreetingCard'

export default {
 name: 'Greeting',
```

```
components: {
  GreetingCard
},

data: () => ({
  username: '',

  savedUsername: ''
}),

methods: {
  submitUsername () {
    // сохраним username в localStorage
    localStorage.setItem('username', this.username)

    // сохраним его в отдельную переменную,
    // для дальнейшей передачи в компоненту
    // GreetingCard
    this.savedUsername = this.username

    // очистим форму
    this.$refs.form.reset()
  }
},

created () {
  // если localStorage содержит значение по ключу
  // username, то запишем его в наши переменные
  if (localStorage.getItem('username')) {
    this.savedUsername = localStorage.getItem('username')

    this.username = this.savedUsername
  }
}
}
</script>

<style>
</style>
```

Теперь можно перейти к компоненте GreetingCard, в которой необходимо добавить объект props:

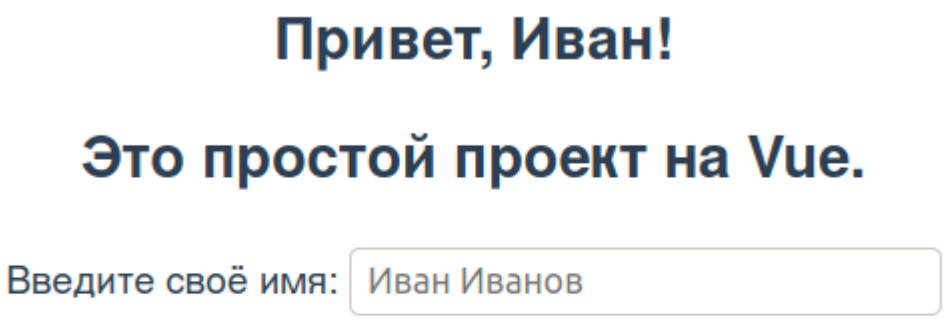
```
<template>
 <h1>Привет, {{ username }}!</h1>
</template>

<script>
export default {
 name: 'GreetingCard',

 props: {
   username: String
 }
}
</script>

<style>
</style>
```

В качестве содержимого объекта указываются переменные, которые можно будет использовать в этой компоненте, получая их из родительской компоненты/представления.

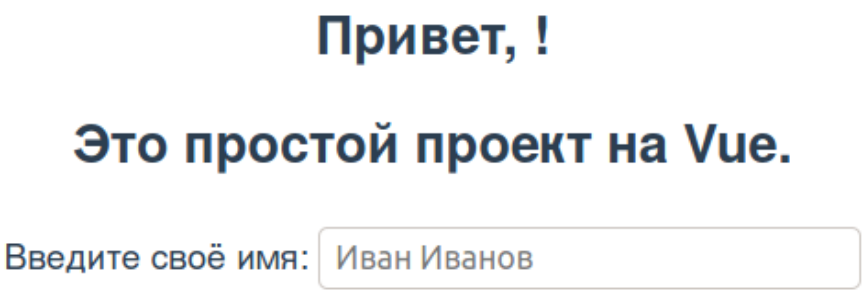
Теперь необходимо добавить передачу какой-либо переменной в GreetingCard, делается это через указание атрибута с именем созданного пропа, значением которого является одна из переменных текущего представления. Двоеточие перед атрибутом сообщает атрибуту о том, что он должен включиться в контекст текущей компоненты и получить данные из той переменной, которую мы туда передаём. Так же, там можно указывать любой валидный JS-код с конечным результатом, но так делать не рекомендуется.

```
 <greeting-card :username="savedUsername" />
```

Теперь нужно проверить работу этой связки, введя имя и нажав на Enter (то есть, вызва событие submit у формы):



Как можно заметить, всё отработало корректно, но есть один неприятный момент. При первоначальном открытии, пользователь увидит следующее:



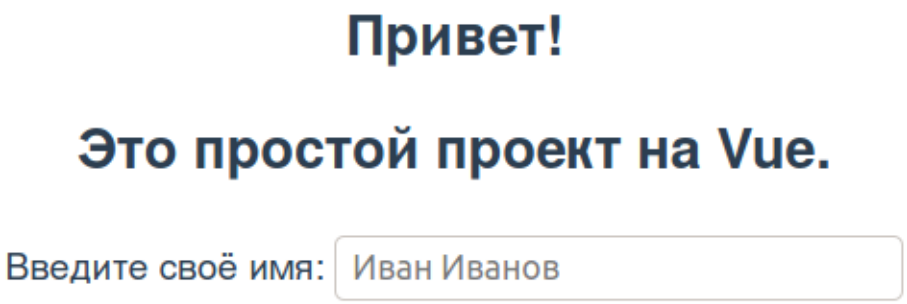
Нам следует избавиться от запятой, если значение, передаваемое в компоненту является пустым. Сделать это можно двумя способами: используя v-show или v-if.

Эффект будет примерно один и тот же, но разница есть: использование v-show просто скрывает или показывает элемент, добавляя или убирая атрибут display: none, а v-if не добавляет или убирает элемент в зависимости от условия.

Используем v-if:

```
<h1>Привет<span v-if="username">, {{ username }}</span>!</h1>
```

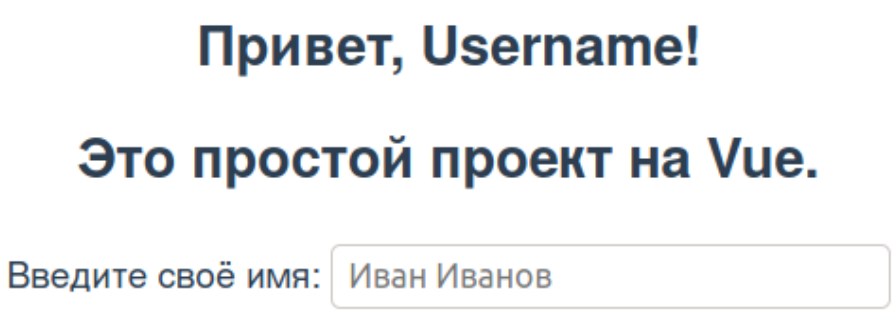
Результат:



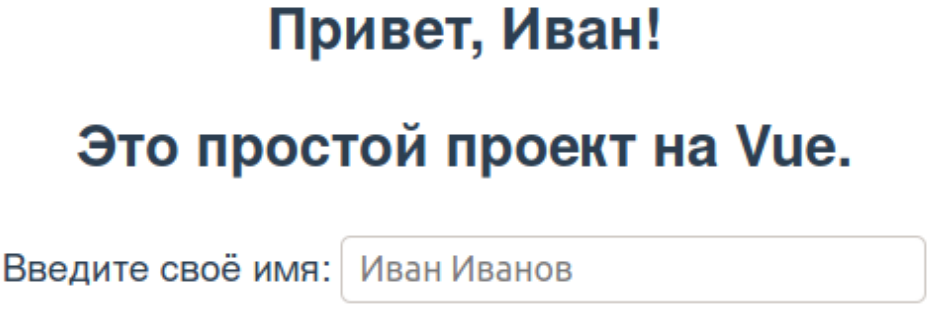
В связке с v-if, можно использовать v-else и v-else-if. Добавим какой-нибудь placeholder, например, Username:

```
<h1>
  Привет,
  <span v-if="username">{{ username }}</span>
  <span v-else>Username</span>!
</h1>
```

До отправки:



После отправки:



Но на этом крутость шаблонов во Vue не заканчивается. Создадим ещё один компонент, чтобы посмотреть, как работают циклы в шаблонах.

В директории src/components создаём файл GreetingList.vue со следующим содержимым:

```
<template>
 <ul class="greeting-list">
   <li
     v-for="greeting in greetings"
     :key="greeting.id"
   >
     {{ greeting.text }}
   </li>
 </ul>
</template>

<script>
export default {
 name: 'GreetingList',

 props: {
   greetings: Array
 }
}
</script>

<style>
.greeting-list li {
 text-align: left;
}
</style>
```

v-for отвечает именно за циклы, здесь мы проходимся по массиву и выводим текст приветствия внутри списка. Атрибут key является уникальным и нужен, чтобы иметь возможность получить доступ к отдельному элементу списка в дальнейшем.

Добавим вывод компоненты в представление Greeting:

```
  <h3>Варианты приветствий:</h3>

  <div class="greeting-list-wrapper">
    <greeting-list :greetings="greetings" />
  </div>
```

Теперь необходимо завести в data() массив объектов greetings:

```
  greetings: [
    { id: 1, text: 'Привет' },
    { id: 2, text: 'Hello' },
    { id: 3, text: 'Hola' }
  ]
```

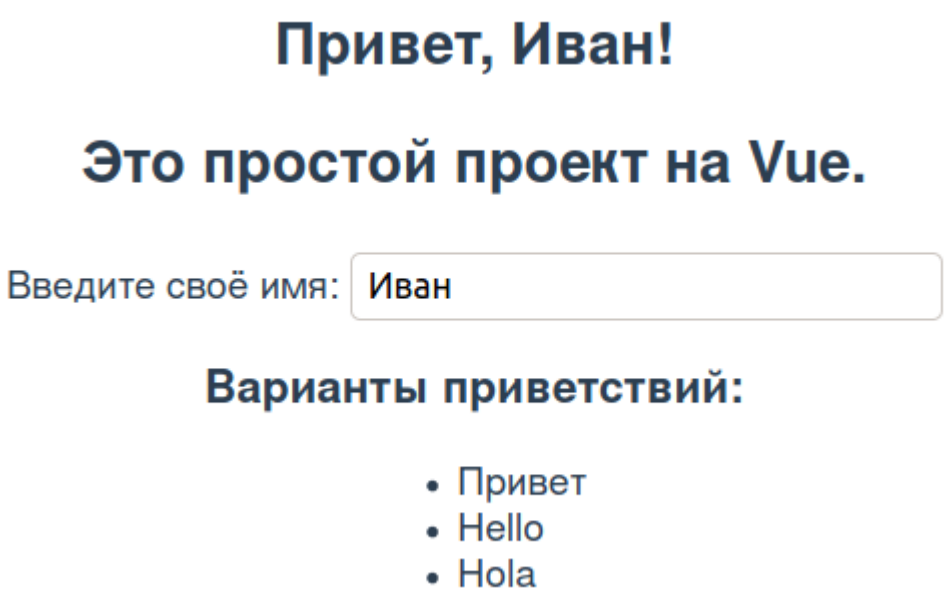
Добавление стилей в представление:

```
<style>
.greeting-list-wrapper {
 display: flex;
}

.greeting-list-wrapper .greeting-list {
 margin: auto
}
</style>
```

И в саму компоненту:

```
<style>
.greeting-list li {
 text-align: left;
}
</style>
```

Результат:



Таким же образом можно выводить любую информацию, получаемую с сервера. А с помощью v-if можно определять, что стоит вывести пользователю.

3. Подключение UI-фреймворка (vuetify)

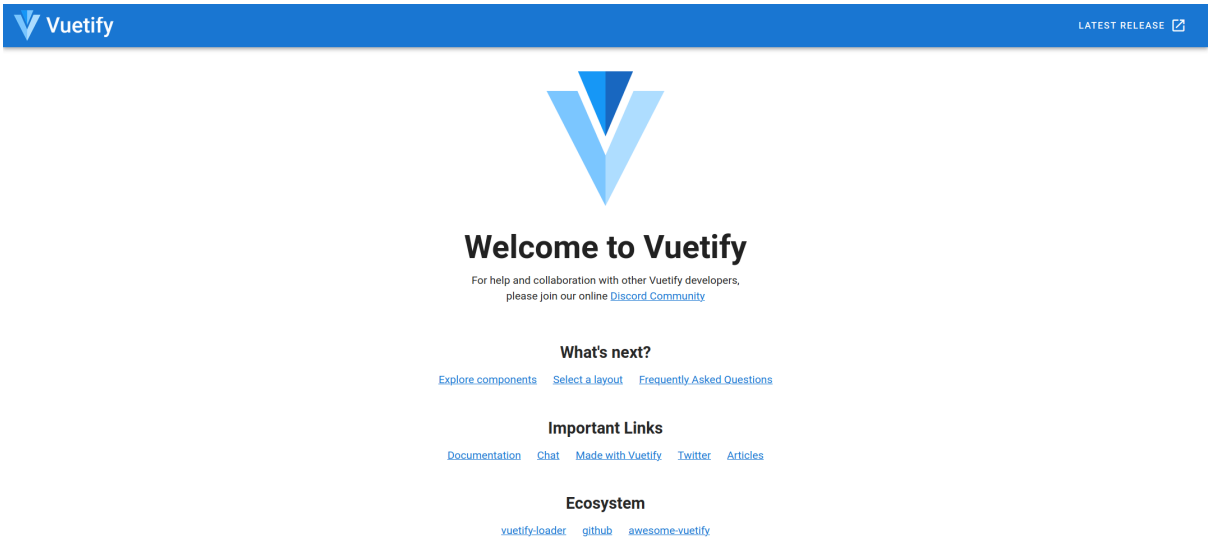
Использование UI-фреймворков позволяет ускорить процесс разработки за счёт большого количества готовых компонентов. Наиболее простой UI-фреймворк, для начала, это Vuetify. Документацию по нему можно найти здесь.

Установить vuetify можно через vue-cli следующей командой:

```
$ vue add vuetify
```

Она автоматически добавит все необходимые файла для работы.

После успешной установки главная страница будет иметь следующий вид:



Изменим содержимое главной компоненты:

```
<template>
 <v-app>
   <v-app-bar
     app
     color="primary"
     dark
   >
     <div class="d-flex align-center">
       <h1 class="headline">Простой проект на Vue</h1>
     </div>
   </v-app-bar>

   <v-main class="d-flex align-center text-center">
     <router-view />
   </v-main>
 </v-app>
</template>

<script>
export default {
 name: 'App',
```

```
 data: () => ({
   //
 })
}
</script>
```

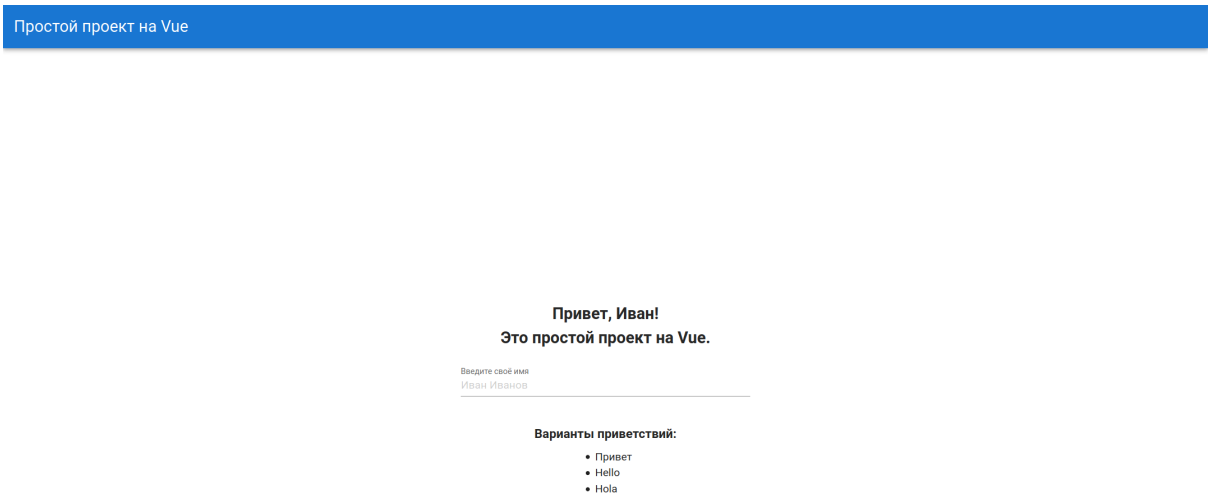
И содержимое представления Greeting:

```
<template>
 <section>
   <greeting-card :username="savedUsername" />

   <h2>Это простой проект на Vue.</h2>

   <v-form
     @submit.prevent="submitUsername"
     ref="form"
     class="my-2"
   >
     <v-row>
       <v-col cols="3" class="mx-auto">
         <v-text-field
           label="Введите своё имя"
           v-model="username"
           name="username"
           placeholder="Иван Иванов"
         />
       </v-col>
     </v-row>

   </v-form>

   <h3>Варианты приветствий:</h3>

   <div class="my-2 greeting-list-wrapper">
     <greeting-list :greetings="greetings" />
   </div>
 </section>
</template>

<script>
```

```
import GreetingCard from '@/components/GreetingCard'
import GreetingList from '@/components/GreetingList'

export default {
 name: 'Greeting',

 components: {
   GreetingCard,
   GreetingList
 },

 data: () => ({
   username: '',

   savedUsername: '',

   greetings: [
     { id: 1, text: 'Привет' },
     { id: 2, text: 'Hello' },
     { id: 3, text: 'Hola' }
   ]
 }),

 methods: {
   submitUsername () {
     // сохраним username в localStorage
     localStorage.setItem('username', this.username)

     // сохраним его в отдельную переменную,
     // для дальнейшей передачи в компоненту
     // GreetingCard
     this.savedUsername = this.username

     // очистим форму
     this.$refs.form.reset()
   }
 },

 created () {
   // если localStorage содержит значение по ключу
   // username, то запишем его в наши переменные
   if (localStorage.getItem('username')) {
     this.savedUsername = localStorage.getItem('username')
```

```
      this.username = this.savedUsername
    }
  }
}
</script>

<style>
.greeting-list-wrapper {
 display: flex;
}

.greeting-list-wrapper .greeting-list {
 margin: auto
}
</style>
```

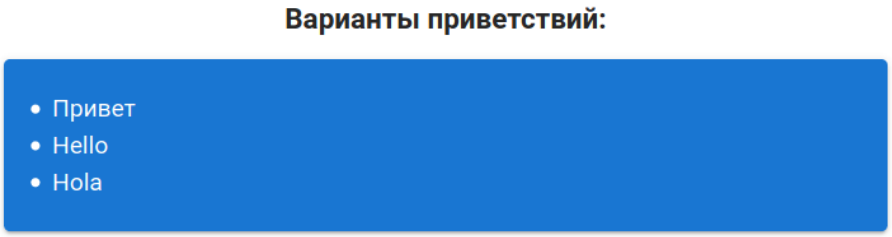
Результат:



Теперь заменим элементы сделаем карточку из пункта “Варианты приветствия”:

```
<v-row>
  <v-col cols="4" class="mx-auto">
    <v-card
      elevation="2"
      class="px-2 py-5"
      color="primary"
      dark
    >
      <greeting-list :greetings="greetings" />
    </v-card>
  </v-col>
</v-row>
```

Результат:



Во Vuetify используется колоночная вёрстка, что позволяет условно делить рабочую область на 12 равных частей и, в зависимости, от ширины того или иного элемента выдавать ему то или иное количество колонок. Например, количество колонок 4, отведённое под определённый элемент, будет означать, что его ширина будет равна 4 колонкам из 12, то есть 33% от рабочей области.

Все компоненты vuetify можно посмотреть здесь.

4. Запросы к API через axios

Запросы к API это важная часть работы SPA, можно даже сказать, основная. Поскольку вся работа SPA строится на асинхронных запросах. Для выполнения запросов будет использовать библиотека axios.

Установить библиотеку можно следующей командой:

```
$ npm install axios vue-axios --save-dev
```

После чего необходимо добавить эти зависимости в src/main.js:

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import vuetify from './plugins/vuetify'
import axios from 'axios'
import VueAxios from 'vue-axios'

Vue.config.productionTip = false
Vue.use(VueAxios, axios)

new Vue({
 router,
 vuetify,
 axios,
 render: h => h(App)
}).$mount('#app')
```

Работать будем с открытым API NASA: https://api.nasa.gov/.

Создадим компоненту EpicItemCard:

```
<template>
 <v-card
   elevation="2"
   class="px-2 py-5"
 >
 <v-row>
   <v-col cols="4">
     <v-img :src="epicItem.image" />
   </v-col>

   <v-col cols="6">
     <v-card-text>
       {{ epicItem.caption }}
```

```
      </v-card-text>
    </v-col>
  </v-row>
</v-card>
</template>
<script>
export default {
 name: 'EpicItemCard',

 props: {
   epicItem: Object
 }
}
</script>
```

И представление Nasa:

```
<template>
 <section>
   <v-row>
     <v-col cols="6" class="mx-auto">
       <epic-item-card
         v-for="epicItem in epicItems"
         :key="epicItem.identifier"
         :epic-item="epicItem"
         class="my-2"
       />
     </v-col>
   </v-row>
 </section>
</template>

<script>
import EpicItemCard from '@/components/EpicItemCard.vue'
const apiKey = 'pv7mgIb4MCWKU5a8q0Zm8phM3KZiCaKdOXA0Hmth'
const apiUrl = 'https://api.nasa.gov/EPIC'

export default {
 components: { EpicItemCard },
 name: 'Greeting',
```

```
data: () => ({
  epicItems: []
}),

methods: {
  async getEpicItems () {
    try {
      // выполним запрос на получение списка всех
      // фотографий земли
      const response = await this.axios
        .get(`${apiUrl}/api/natural?api_key=${apiKey}`)

      // если статус ответа не 200, то выкинем
      // ошибку
      if (response.status !== 200) {
        throw new Error(response.error)
      }

      // пройдёмся по полученному массиву данных и
      // приведём его к тому виду, с которым будет
      // удобно работать
      const epicItems = response.data.map((epicItem) => {
        const date = new Date(epicItem.date)

        // собираем дату в нужном виде
        const year = date.getFullYear()

        const month = String(date.getMonth() + 1).length > 1
          ? date.getMonth() + 1
          : `0${date.getMonth() + 1}`

        const day = String(date.getDate()).length > 1
          ? date.getDate()
          : `0${date.getDate()}`

        epicItem.date = `${year}/${month}/${day}`

        // получаем картинку
        epicItem.image =

`${apiUrl}/archive/natural/${epicItem.date}/png/${epicItem.image}.png?a
pi_key=${apiKey}`
```

```
          return epicItem
        })

        this.epicItems = epicItems
      } catch (e) {
        console.error('AN API ERROR', e)
      }
    }
  },

  created () {
    this.getEpicItems()
  }
}
</script>

<style>
</style>
```

Константы apiKey и apiUrl отвечают за ключ, который выдаётся после регистрации и адрес, на который мы будем слать запросы. После успешного выполнения запроса необходимо обработать полученный результат, чтобы вывести данные в том виде, в котором их выводить удобно, а именно: получить адрес картинки, которая будет показываться в карточке.

Добавим представление в src/router/index.js:

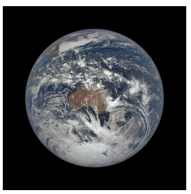
```
import Vue from 'vue'
import VueRouter from 'vue-router'

// импортируем все представления, по
// которым будем навигировать пользователя
import Greeting from '@/views/Greeting.vue'
import Nasa from '@/views/Nasa.vue'

Vue.use(VueRouter)

// заводим массив с роутами
const routes = [
  {
    path: '/',
```

```
    name: 'Greeting',
    component: Greeting
  },

  {
    path: '/nasa',
    name: 'Nasa',
    component: Nasa
  }
]

// создаём новый экземпляр класса
// VueRouter, с необходимыми параметрами
const router = new VueRouter({
  mode: 'history',
  base: process.env.BASE_URL,
  routes
})

// экспортируем сконфигурированный роутер
export default router
```

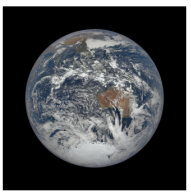
Результат:

Простой проект на Vue

This image was taken by NASA's EPIC camera onboard the NOAA DSCOVR spacecraft



This image was taken by NASA's EPIC camera onboard the NOAA DSCOVR spacecraft



This image was taken by NASA's EPIC camera onboard the NOAA DSCOVR spacecraft