▧ TUẦN 3: UI FRAMEWORK (VUETIFY) - HƯỚNG DẪN THỰC HÀNH CHI TIẾT

🎯 Mục tiêu

- Cài đặt và cấu hình Vuetify chuẩn cho dự án Vue 3.
- Nắm vững Grid System, layout responsive và theme.
- Thực hành xây dựng Dashboard, Data Table, Form, Dialog, Navigation Drawer.
- Tổ chức component, tối ưu UX và xử lý các lỗi thường gặp.

🧭 1. Tổng quan nhanh về Vuetify

Vuetify là UI framework theo Material Design cho Vue. Ưu điểm:

- Nhiều component sẵn, nhất quán UX/UI.
- Hệ thống lưới mạnh, responsive chuẩn.
- Hỗ trợ theme, icon, form validation, data table, dialog...

🛠️ 2. Cài đặt Vuetify (Vue 3 + Vite)

2.1. Cài thư viện

```
npm add vuetify
npm add @mdi/font
```

2.2. Cấu hình `main.js`

```
import { createApp } from 'vue'
import App from './App.vue'

import 'vuetify/styles'
import { createVuetify } from 'vuetify'
import * as components from 'vuetify/components'
import * as directives from 'vuetify/directives'
import '@mdi/font/css/materialdesignicons.css'

const vuetify = createVuetify({
  components,
  directives,
  theme: {
    defaultTheme: 'light',
    themes: {
      light: {
        colors: {
```

```
            primary: '#1976D2',
            secondary: '#424242',
            success: '#2E7D32',
            warning: '#ED6C02',
            error: '#D32F2F',
            info: '#0288D1',
          },
        },
      },
    },
  })

  createApp(App).use(vuetify).mount('#app')
```

2.3. Kiểm tra nhanh

```
<template>
  <v-app>
    <v-main class="pa-4">
      <v-btn color="primary">Vuetify OK</v-btn>
    </v-main>
  </v-app>
</template>
```

📐 3. Grid System và Layout chuẩn responsive

3.1. Quy tắc cơ bản

- Grid 12 cột: `v-container` > `v-row` > `v-col`.
- `cols="12"` cho mobile, `md="6"` cho desktop.

```
<v-container>
  <v-row>
    <v-col cols="12" md="6">
      <v-card class="pa-4">Cột trái</v-card>
    </v-col>
    <v-col cols="12" md="6">
      <v-card class="pa-4">Cột phải</v-card>
    </v-col>
  </v-row>
</v-container>
```

3.2. Breakpoints thường dùng

- `xs` (mặc định), `sm`, `md`, `lg`, `xl`
- Ví dụ: 4 card hiển thị 1 cột ở mobile, 2 ở tablet, 4 ở desktop.

```
<v-col cols="12" sm="6" md="3">...</v-col>
```

---

4. Thực hành 1: Xây dựng Dashboard

4.1. Yêu cầu

- 4 stat cards: Doanh thu, Đơn hàng, Khách hàng, Sản phẩm.
- Layout responsive: mobile 1 cột, tablet 2 cột, desktop 4 cột.
- Mỗi card có icon + số liệu + tiêu đề.

4.2. Gợi ý code

```
<script setup>
const stats = [
  { title: 'Doanh thu', value: '125.000.000', icon: 'mdi-cash', color: 'success'
},
  { title: 'Đơn hàng', value: '45', icon: 'mdi-cart', color: 'primary' },
  { title: 'Khách hàng', value: '120', icon: 'mdi-account-group', color: 'info' },
  { title: 'Sản phẩm', value: '89', icon: 'mdi-package-variant', color: 'warning'
},
]
</script>

<template>
  <v-container>
    <h1 class="mb-4">Dashboard</h1>
    <v-row>
      <v-col
        v-for="stat in stats"
        :key="stat.title"
        cols="12"
        sm="6"
        md="3"
      >
        <v-card class="pa-4">
          <div class="d-flex align-center">
            <v-icon :color="stat.color" size="40" class="mr-4">
              {{ stat.icon }}
            </v-icon>
            <div>
              <div class="text-h6">{{ stat.value }}</div>
              <div class="text-caption">{{ stat.title }}</div>
            </div>
          </div>
        </v-card>
      </v-col>
    </v-row>
  </v-container>
</template>
```

5. Thực hành 2: Data Table + Search + Actions

5.1. Mục tiêu

- Hiển thị danh sách sản phẩm.
- Có ô tìm kiếm và nút "Thêm mới".
- Cột hành động: sửa / xoá.

5.2. Gợi ý code

```
<script setup>
import { ref, computed } from 'vue'

const search = ref('')

const headers = [
  { title: 'Tên sản phẩm', key: 'name' },
  { title: 'Giá', key: 'price' },
  { title: 'Danh mục', key: 'category' },
  { title: 'Hành động', key: 'actions', sortable: false },
]

const products = ref([
  { id: 1, name: 'iPhone 15', price: 20000000, category: 'Điện thoại' },
  { id: 2, name: 'Samsung S24', price: 18000000, category: 'Điện thoại' },
  { id: 3, name: 'Macbook Air', price: 25000000, category: 'Laptop' },
])

const filteredProducts = computed(() =>
  products.value.filter((p) =>
    p.name.toLowerCase().includes(search.value.toLowerCase())
  )
)
</script>

<template>
  <v-container>
    <div class="d-flex align-center justify-space-between mb-4">
      <v-text-field
        v-model="search"
        label="Tìm kiếm sản phẩm"
        variant="outlined"
        density="compact"
        style="max-width: 300px;"
      />
      <v-btn color="primary" prepend-icon="mdi-plus">Thêm mới</v-btn>
    </div>

    <v-data-table
```

```
      :headers="headers"
      :items="filteredProducts"
      item-value="id"
    >
      <template v-slot:item.price="{ item }">
        {{ item.price.toLocaleString() }} đ
      </template>
      <template v-slot:item.actions="{ item }">
        <v-btn icon="mdi-pencil" variant="text" color="primary" />
        <v-btn icon="mdi-delete" variant="text" color="error" />
      </template>
    </v-data-table>
  </v-container>
</template>
```

🧩 6. Thực hành 3: Form + Validation

6.1. Mục tiêu

- Tạo form thêm sản phẩm.
- Kiểm tra required + min length + số dương.

6.2. Gợi ý code

```
<script setup>
import { ref } from 'vue'

const form = ref(null)
const model = ref({
  name: '',
  price: null,
  category: '',
})

const rules = {
  required: (v) => !!v || 'Bắt buộc nhập',
  min3: (v) => (v && v.length >= 3) || 'Ít nhất 3 ký tự',
  positive: (v) => (v > 0) || 'Giá phải > 0',
}

const submit = async () => {
  const { valid } = await form.value.validate()
  if (valid) {
    // TODO: xử lý submit
  }
}
</script>

<template>
  <v-form ref="form">
```

```
      <v-text-field
        v-model="model.name"
        label="Tên sản phẩm"
        :rules="[rules.required, rules.min3]"
        variant="outlined"
      />
      <v-text-field
        v-model="model.price"
        label="Giá"
        type="number"
        :rules="[rules.required, rules.positive]"
        variant="outlined"
      />
      <v-select
        v-model="model.category"
        :items="['Điện thoại', 'Laptop', 'Phụ kiện']"
        label="Danh mục"
        :rules="[rules.required]"
        variant="outlined"
      />
      <v-btn color="primary" @click="submit">Lưu</v-btn>
    </v-form>
  </template>
```

---

7. Thực hành 4: Dialog (Modal) thêm/sửa

7.1. Mục tiêu

- Bấm "Thêm mới" mở dialog.
- Form nằm trong dialog.

7.2. Gợi ý code

```
<script setup>
import { ref } from 'vue'

const open = ref(false)
</script>

<template>
  <v-btn color="primary" @click="open = true">Thêm mới</v-btn>

  <v-dialog v-model="open" width="500">
    <v-card>
      <v-card-title>Thêm sản phẩm</v-card-title>
      <v-card-text>
        <!-- Form ở đây -->
      </v-card-text>
      <v-card-actions class="justify-end">
        <v-btn variant="text" @click="open = false">Hủy</v-btn>
```

```
        <v-btn color="primary">Lưu</v-btn>
      </v-card-actions>
    </v-card>
  </v-dialog>
</template>
```

🧩 8. Thực hành 5: Navigation Drawer + App Bar

8.1. Mục tiêu

- Tạo sidebar có menu.
- App Bar có nút toggle.

8.2. Gợi ý code

```
<script setup>
import { ref } from 'vue'
const drawer = ref(true)
</script>

<template>
  <v-app>
    <v-navigation-drawer v-model="drawer">
      <v-list>
        <v-list-item prepend-icon="mdi-view-dashboard" title="Dashboard" />
        <v-list-item prepend-icon="mdi-package-variant" title="Sản phẩm" />
        <v-list-item prepend-icon="mdi-account-group" title="Khách hàng" />
      </v-list>
    </v-navigation-drawer>

    <v-app-bar>
      <v-app-bar-nav-icon @click="drawer = !drawer" />
      <v-app-bar-title>Admin</v-app-bar-title>
    </v-app-bar>

    <v-main class="pa-4">
      <router-view />
    </v-main>
  </v-app>
</template>
```

🧩 9. Dự án CRUD hoàn chỉnh với Vuetify (Mock API)

9.1. Mục tiêu dự án

- Xây trang quản trị sản phẩm (CRUD).
- Dữ liệu lấy từ Mock API (hoặc REST API thật).

- UI hoàn chỉnh: list + search + pagination + form dialog.

9.2. Tạo project và cài thư viện

```
npm create vite@latest vue-vuetify-crud -- --template vue
cd vue-vuetify-crud
npm install
npm add vuetify @mdi/font axios pinia vue-router
```

9.3. Cấu trúc thư mục gợi ý

```
src/
  main.js
  plugins/vuetify.js
  router/index.js
  stores/productStore.js
  services/productApi.js
  views/
    DashboardPage.vue
    ProductsPage.vue
  components/
    ProductTable.vue
    ProductForm.vue
```

9.4. Tách cấu hình Vuetify

src/plugins/vuetify.js

```
import 'vuetify/styles'
import { createVuetify } from 'vuetify'
import * as components from 'vuetify/components'
import * as directives from 'vuetify/directives'
import '@mdi/font/css/materialdesignicons.css'

export default createVuetify({
  components,
  directives,
  theme: {
    defaultTheme: 'light',
    themes: {
      light: {
        colors: {
          primary: '#1976D2',
          success: '#2E7D32',
          warning: '#ED6C02',
          error: '#D32F2F',
        },
      },
```

```
    },
  },
})
```

src/main.js

```
import { createApp } from 'vue'
import { createPinia } from 'pinia'
import App from './App.vue'
import router from './router'
import vuetify from './plugins/vuetify'

createApp(App).use(createPinia()).use(router).use(vuetify).mount('#app')
```

9.5. Router cơ bản

src/router/index.js

```
import { createRouter, createWebHistory } from 'vue-router'
import DashboardPage from '../views/DashboardPage.vue'
import ProductsPage from '../views/ProductsPage.vue'

const routes = [
  { path: '/', component: DashboardPage },
  { path: '/products', component: ProductsPage },
]

export default createRouter({
  history: createWebHistory(),
  routes,
})
```

9.6. Service gọi API (MockAPI)

Tạo project ở https://mockapi.io và lấy base URL ví dụ: https://67890abcd.mockapi.io/api/v1

src/services/productApi.js

```
import axios from 'axios'

const api = axios.create({
  baseURL: 'https://YOUR_ID.mockapi.io/api/v1',
})

export const getProducts = (params) => api.get('/products', { params })
export const createProduct = (data) => api.post('/products', data)
```

```
export const updateProduct = (id, data) => api.put(`/products/${id}`, data)
export const deleteProduct = (id) => api.delete(`/products/${id}`)
```

9.7. Pinia store quản lý state

src/stores/productStore.js

```
import { defineStore } from 'pinia'
import { getProducts, createProduct, updateProduct, deleteProduct } from
'../services/productApi'

export const useProductStore = defineStore('products', {
  state: () => ({
    items: [],
    loading: false,
    error: null,
  }),
  actions: {
    async fetch(params = {}) {
      this.loading = true
      this.error = null
      try {
        const { data } = await getProducts(params)
        this.items = data
      } catch (err) {
        this.error = 'Không tải được dữ liệu'
      } finally {
        this.loading = false
      }
    },
    async add(payload) {
      const { data } = await createProduct(payload)
      this.items.unshift(data)
    },
    async edit(id, payload) {
      const { data } = await updateProduct(id, payload)
      const idx = this.items.findIndex((p) => p.id === id)
      if (idx !== -1) this.items[idx] = data
    },
    async remove(id) {
      await deleteProduct(id)
      this.items = this.items.filter((p) => p.id !== id)
    },
  },
})
```

9.8. ProductsPage: ghép Table + Form + Dialog

src/views/ProductsPage.vue

```vue
<script setup>
import { ref, computed, onMounted } from 'vue'
import { useProductStore } from '../stores/productStore'
import ProductTable from '../components/ProductTable.vue'
import ProductForm from '../components/ProductForm.vue'

const store = useProductStore()
const open = ref(false)
const editing = ref(null)
const search = ref('')

const filtered = computed(() =>
  store.items.filter((p) =>
    p.name.toLowerCase().includes(search.value.toLowerCase())
  )
)

onMounted(() => store.fetch())

const openCreate = () => {
  editing.value = null
  open.value = true
}

const openEdit = (item) => {
  editing.value = { ...item }
  open.value = true
}

const save = async (payload) => {
  if (editing.value) await store.edit(editing.value.id, payload)
  else await store.add(payload)
  open.value = false
}
</script>

<template>
  <v-container>
    <div class="d-flex align-center justify-space-between mb-4">
      <v-text-field
        v-model="search"
        label="Tìm kiếm"
        density="compact"
        variant="outlined"
        style="max-width: 300px;"
      />
      <v-btn color="primary" prepend-icon="mdi-plus" @click="openCreate">
        Thêm mới
      </v-btn>
    </div>

    <product-table
      :items="filtered"
```

```
      :loading="store.loading"
      @edit="openEdit"
      @remove="store.remove"
    />

    <v-dialog v-model="open" width="520">
      <v-card>
        <v-card-title>{{ editing ? 'Sửa sản phẩm' : 'Thêm sản phẩm' }}</v-card-
title>
        <v-card-text>
          <product-form :model="editing" @submit="save" />
        </v-card-text>
      </v-card>
    </v-dialog>
  </v-container>
</template>
```

9.9. ProductTable component

src/components/ProductTable.vue

```
<script setup>
const props = defineProps({
  items: { type: Array, default: () => [] },
  loading: { type: Boolean, default: false },
})
const emit = defineEmits(['edit', 'remove'])

const headers = [
  { title: 'Tên sản phẩm', key: 'name' },
  { title: 'Giá', key: 'price' },
  { title: 'Danh mục', key: 'category' },
  { title: 'Hành động', key: 'actions', sortable: false },
]
</script>

<template>
  <v-data-table :headers="headers" :items="items" :loading="loading">
    <template v-slot:item.price="{ item }">
      {{ item.price.toLocaleString() }} đ
    </template>
    <template v-slot:item.actions="{ item }">
      <v-btn icon="mdi-pencil" variant="text" @click="emit('edit', item)" />
      <v-btn icon="mdi-delete" variant="text" color="error" @click="emit('remove',
item.id)" />
    </template>
  </v-data-table>
</template>
```

9.10. ProductForm component

src/components/ProductForm.vue

```
<script setup>
import { ref, watch } from 'vue'

const props = defineProps({ model: Object })
const emit = defineEmits(['submit'])

const form = ref(null)
const payload = ref({ name: '', price: null, category: '' })

watch(
  () => props.model,
  (val) => {
    payload.value = val
      ? { name: val.name, price: val.price, category: val.category }
      : { name: '', price: null, category: '' }
  },
  { immediate: true }
)

const rules = {
  required: (v) => !!v || 'Bắt buộc nhập',
  min3: (v) => (v && v.length >= 3) || 'Ít nhất 3 ký tự',
  positive: (v) => (v > 0) || 'Giá phải > 0',
}

const submit = async () => {
  const { valid } = await form.value.validate()
  if (valid) emit('submit', { ...payload.value })
}
</script>

<template>
  <v-form ref="form">
    <v-text-field
      v-model="payload.name"
      label="Tên sản phẩm"
      :rules="[rules.required, rules.min3]"
      variant="outlined"
    />
    <v-text-field
      v-model="payload.price"
      label="Giá"
      type="number"
      :rules="[rules.required, rules.positive]"
      variant="outlined"
    />
    <v-select
      v-model="payload.category"
      :items="['Điện thoại', 'Laptop', 'Phụ kiện']"
      label="Danh mục"
      :rules="[rules.required]"
```

```
        variant="outlined"
      />
      <v-btn color="primary" @click="submit">Lưu</v-btn>
    </v-form>
  </template>
```

🔥 10. Firestore: tạo CSDL quan hệ + Public API CRUD (không dùng Cloud Storage)

> Phần này viết cho người mới bắt đầu. Chỉ cần làm từng bước là chạy được. Không dùng Cloud Storage. Ảnh (nếu có) chỉ lưu **URL string**.

10.0. Khái niệm cực nhanh (đọc 2 phút)

- **Collection** = bảng trong CSDL quan hệ.
- **Document** = một dòng dữ liệu trong bảng.
- **Field** = cột dữ liệu.
- **Firestore không có JOIN** → ta lưu "khóa ngoại" thủ công bằng `id` (string).

10.1. Tạo Firebase project + bật Firestore (từng bước)

1. Mở https://console.firebase.google.com và đăng nhập Gmail.
2. Bấm **Add project** → đặt tên (vd: `vuetify-crud`).
3. Chọn **Continue** → (tuỳ chọn) bỏ chọn Analytics → **Create project**.
4. Khi tạo xong, bấm **Build** → **Firestore Database** ở menu trái.
5. Chọn **Create database** → **Start in test mode** → chọn khu vực (asia-southeast1) → **Enable**.

> Test mode giúp dùng được REST public. Sau 30 ngày sẽ hết hiệu lực, có thể chỉnh rules.

10.2. Tạo Web App và lấy thông tin kết nối

1. Ở Project Overview, bấm biểu tượng **</>** (Web).
2. Đặt tên app (vd: `vuetify-web`) → **Register app**.
3. Copy **firebaseConfig** (dùng khi cần SDK).
4. Kéo xuống **Your apps** → tìm **Web API Key** (dùng cho REST public).

10.3. Thiết kế CSDL: so sánh với CSDL quan hệ

CSDL quan hệ (ví dụ):

- `users(id, name, email)`
- `categories(id, name)`
- `products(id, name, price, category_id, created_by)`

Firestore tương ứng:

- Collection `users`: `{ name, email }`
- Collection `categories`: `{ name }`
- Collection `products`: `{ name, price, categoryId, createdBy }`

Quan hệ (mapping):

- `products.categoryId` ~ khóa ngoại đến `categories.id`
- `products.createdBy` ~ khóa ngoại đến `users.id`

> Vì Firestore không JOIN, ta lưu `categoryId` và `createdBy` trực tiếp trong document `products`.

10.4. Tạo dữ liệu mẫu trong Firestore Console (tỉ mỉ)

1. Trong Firestore Database → tab **Data** → **Start collection**.
2. Collection ID nhập `users` → **Next**.
3. Document ID nhập `u1`, thêm field:
   - `name` (string) = `Admin`
   - `email` (string) = `admin@demo.com`
4. Bấm **Save**.
5. Tạo tiếp collection `categories`:
   - Document ID: `c1`
   - Field: `name` (string) = `Điện thoại`
6. Tạo tiếp collection `products`:
   - Document ID: `p1`
   - Field `name` (string) = `iPhone 15`
   - Field `price` (number) = `20000000`
   - Field `categoryId` (string) = `c1`
   - Field `createdBy` (string) = `u1`

> Đến đây là bạn đã có 3 collection có quan hệ với nhau.

10.5. Cấu hình Firebase (tuỳ chọn nếu dùng SDK)

`src/firebase.js`

```
import { initializeApp } from 'firebase/app'
import { getFirestore } from 'firebase/firestore'

const firebaseConfig = {
  apiKey: 'YOUR_KEY',
  authDomain: 'YOUR_DOMAIN',
  projectId: 'YOUR_PROJECT_ID',
  messagingSenderId: 'YOUR_SENDER_ID',
  appId: 'YOUR_APP_ID',
}

const app = initializeApp(firebaseConfig)
export const db = getFirestore(app)
```

> Nếu bạn chỉ dùng REST API, không cần file này.

10.6. Public API CRUD với Firestore REST (cho người mới)

Base URL:

```
https://firestore.googleapis.com/v1/projects/PROJECT_ID/databases/(default)/documents
```

API Key: vào Firebase console → Project settings → General → Web API Key.

> REST API của Firestore yêu cầu payload theo chuẩn `fields`.

Helper chuyển object → fields:

```
export const toFields = (obj) =>
  Object.fromEntries(
    Object.entries(obj).map(([k, v]) => {
      if (typeof v === 'number') {
        return [k, Number.isInteger(v) ? { integerValue: String(v) } : { doubleValue: v }]
      }
      if (typeof v === 'boolean') return [k, { booleanValue: v }]
      return [k, { stringValue: String(v) }]
    })
  )
```

Kiểm tra nhanh bằng trình duyệt (GET list):

```
https://firestore.googleapis.com/v1/projects/PROJECT_ID/databases/(default)/documents/products?key=YOUR_API_KEY
```

Ví dụ CRUD bằng axios:

```
import axios from 'axios'
import { toFields } from './firestoreHelpers'

const BASE =
'https://firestore.googleapis.com/v1/projects/PROJECT_ID/databases/(default)/documents'
const KEY = 'YOUR_API_KEY'

export const getProductsPublic = async () => {
  const { data } = await axios.get(`${BASE}/products?key=${KEY}`)
  return (data.documents || []).map((doc) => ({
    id: doc.name.split('/').pop(),
    name: doc.fields.name?.stringValue || '',
    price: Number(doc.fields.price?.integerValue || 0),
    categoryId: doc.fields.categoryId?.stringValue || '',
    createdBy: doc.fields.createdBy?.stringValue || '',
```

```
  }))
}

export const createProductPublic = (payload) =>
  axios.post(`${BASE}/products?key=${KEY}`, { fields: toFields(payload) })

export const updateProductPublic = (id, payload) =>
  axios.patch(`${BASE}/products/${id}?key=${KEY}`, { fields: toFields(payload) })

export const deleteProductPublic = (id) =>
  axios.delete(`${BASE}/products/${id}?key=${KEY}`)
```

> Khi tạo sản phẩm mới, chỉ cần truyền:
> `{ name: 'AirPods', price: 3500000, categoryId: 'c1', createdBy: 'u1' }`

10.7. Firestore Rules (public demo)

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{id} {
      allow read, write: if true;
    }
    match /categories/{id} {
      allow read, write: if true;
    }
    match /products/{id} {
      allow read, write: if true;
    }
  }
}
```

> Lưu ý: Public API chỉ dùng cho demo. Khi làm thật, cần Auth và rules chặt.

☑ 11. Checklist hoàn thiện UI + CRUD

- Theme, icon, layout đã cấu hình đúng.
- List có search + pagination + actions.
- Form có validation và reset sau submit.
- Dialog mở/đóng ổn định, không mất state.
- CRUD Mock API hoặc Firebase chạy ổn định.

❌ 12. Lỗi thường gặp và cách xử lý

Lỗi 1: Quên import Vuetify styles

❌ Vấn đề:

```
import { createVuetify } from 'vuetify'
// ❌ Quên import styles
```

☑ **Giải pháp:**

```
import 'vuetify/styles'
import { createVuetify } from 'vuetify'
```

Lỗi 2: Grid không responsive

❌ **Vấn đề:**

```
<v-col cols="6">
```

☑ **Giải pháp:**

```
<v-col cols="12" sm="6" md="4" lg="3">
```

Lỗi 3: Data Table không hiển thị

❌ **Vấn đề:**

```
<v-data-table :items="products">
```

☑ **Giải pháp:**

```
<v-data-table :headers="headers" :items="products">
```

Lỗi 4: Firebase không load dữ liệu

❌ **Vấn đề:** quên cấu hình `firebaseConfig` hoặc chưa bật Firestore. ☑ **Giải pháp:** kiểm tra config + bật Firestore + rules.

---

13. Best Practices

- Tách component nhỏ, tái sử dụng.
- Dùng slots để tùy biến UI.
- Test responsive trên nhiều kích thước.

- Dùng `density="compact"` cho bảng, form để tiết kiệm không gian.
- Dùng Pinia để tách data và UI.
- Lazy load trang admin nếu app lớn.

---

14. Bài tập thực hành cuối buổi

1. Tạo trang Dashboard có 6 stat cards, responsive đủ 3 mức.
2. Tạo trang Products có Data Table + Search + Pagination.
3. Tạo Dialog thêm/sửa sản phẩm (validation đầy đủ).
4. Thêm filter theo danh mục + khoảng giá.
5. Tạo phiên bản Firebase cho CRUD.

---

15. Mini Test

Câu 1: Vuetify dùng Grid System bao nhiêu cột?

▶ Xem đáp án

Câu 2: Breakpoints trong Vuetify gồm những gì?

▶ Xem đáp án

Câu 3: Làm sao validate form trong Vuetify?

▶ Xem đáp án

Câu 4: Firebase CRUD lưu ở đâu?

▶ Xem đáp án

---

16. Quick Notes

- `v-container` + `v-row` + `v-col` để layout.
- `v-btn`, `v-card`, `v-text-field`, `v-data-table` là bộ cơ bản.
- `v-dialog`, `v-navigation-drawer`, `v-app-bar` cho admin layout.
- Mock API giúp test CRUD nhanh.
- Firebase phù hợp demo fullstack nhanh.
- Tham khảo nhanh: https://vuetifyjs.com/en/components/all/