CIS 371 Web Application Programming

Final Review

Lecturer: Dr. Yong Zhuang

Based on the original version by Professor Hans Dulimarta.

Firebase Cloud Firestore

SQL vs. noSQL

- Relational model
- Schema: relationship between tables and fields
- Popular examples
  - Oracle
  - DB2
  - MySQL
  - PostGreSQL

- Non-relational
- Schemaless Datastore
- Cloud Computing and Cloud Storage
- Rapid Development
- Popular examples
  - MongoDB
  - CouchDB
  - BigTable
  - Firebase Realtime DB
  - Firebase Cloud Firestore

Data Model: Hierarchy of Collections-Documents

- Hierarchical structure
  - The “root” holds one or more collections
  - A collection consists of one or more documents
  - A document is one or more key-value pairs
  - A value in a document may refer to a subcollection (1-to-many relationships)
- Data Types in a document
  - string, number, boolean, array, timestamp, map (kv-pairs), geolocation
  - Reference to a subcollection

SQL	Cloud Firestore
Tables	Collections
Rows	Documents
Primary Key	Document ID
Fields	key-value pairs

CRUD Operations: Create Doc (own Doc ID)

SQL

```
// Use “AK” as the primary key for the tuple
INSERT INTO states (abbrev, name, capital) VALUES(“AK”, “Alaska”, “Juneau”)
```

states (SQL table)

Abbrev (PK)	Name	Capital
AK	Alaska	Juneau
AL	Alabama	Montgomery
FL	Florida	Tallahassee

```
import { DocumentReference, setDoc, doc } from "firebase/firestore";
// Option #1: Use file name syntax for doc path
// Primary key "AK" becomes doc id
const doc1: DocumentReference = doc(db, "states", "AK");
setDoc(doc1, { name: "Alaska", capital: "Juneau" })
  .then(() => {
    console.log("New doc added");
  })
  .catch((err: any) => {
    /* your code here */
  });
```

Firestore in TS

CRUD Operations: Create Doc (automatic Doc ID)

```
INSERT INTO states (name, capital) VALUES(“Alaska”, “Juneau”)
```
SQL

states (SQL table)

Name	Capital
Alaska	Juneau
Alabama	Montgomery
Florida	Tallahassee

```
import { CollectionReference, addDoc, doc } from "firebase/firestore";

const myColl: CollectionReference = collection(db, "states");
addDoc(myColl, { name: "Alaska", capital: "Juneau" })
  .then(() => {
    console.log("New doc added");
  })
  .catch((err: any) => {
    /* your code here */
  });
```

Firestore in TS

CRUD Operations: Create Docs from Array

Firestore in TS

```
import {
  DocumentReference,
  setDoc,
  doc,
  collection,
  addDoc,
} from "firebase/firestore";
const stateArr = [
  { abbrev: "CA", name: "California", capital: "Sacramento" },
  { abbrev: "CO", name: "Colorado", capital: "Denver" },
  // more data here
];
// Option 1: Use state abbreviation as document ID
stateArr.forEach(async (st: any) => {
  const stateDoc = doc(db, "states", st.abbrev); // Us Abbreviation as document ID
  await setDoc(stateDoc, { name: st.name, capital: st.capital });
});
// Option 2: Let Firestore generates automatic
const myStateColl = collection(db, "states"); // Do this outside .forEach
stateArr.forEach(async (st: any) => {
  await addDoc(myStateColl, { name: st.name, capital: st.capital });
});
```

await vs. .then()

CRUD Operations: Read All Documents

```
SELECT * FROM states
```
SQL

states (SQL table)

Abbrev (PK)	Name	Capital
AK	Alaska	Juneau
AL	Alabama	Montgomery
FL	Florida	Tallahassee

```
// Assume saved data has the
// following structure
type StateType = {
  abbrev: string;
  name: string;
  capital: string;
};
```

```
import {
  CollectionReference,
  collection,
  QuerySnapshot,
  QueryDocumentSnapshot,
  getDocs,
} from "firebase/firestore";

const myStateColl: CollectionReference = collection(db, "states");

getDocs(myStateColl).then((qs: QuerySnapshot) => {
  qs.forEach((qd: QueryDocumentSnapshot) => {
    const stateData = qd.data() as StateType;
    const docId = qd.id; // Fixed 'cost' to 'const'
    // More code here to manipulate stateData
  });
});
```

Firestore in TS

CRUD Operations: Read A Specific Document

SQL

```
// Select a tuple with a known primary key
SELECT * FROM states WHERE abbrev = “FL”
```

states (SQL table)

Abbrev (PK)	Name	Capital
AK	Alaska	Juneau
AL	Alabama	Montgomery
FL	Florida	Tallahassee

```
// Assume saved data has the
// following structure
type StateType = {
  abbrev: string;
  name: string;
  capital: string;
};
```

```
import {
  DocumentReference,
  doc,
  DocumentSnapshot,
  getDoc,
} from "firebase/firestore";
// FL is a document ID
const myDoc: DocumentReference = doc(db, "states/FL");
getDoc(myDoc).then((qd: DocumentSnapshot) => {
  if (qd.exists()) {
    const stateData = qd.data() as StateType;
    // More code here to manipulate stateData
  }
});
```

Firestore in TS

CRUD Operations: Fetch Document(s) Where...

SQL

```
// Select tuples satisfying some conditions
SELECT * FROM states WHERE name = “Florida”
```

states (SQL table)

Abbrev (PK)	Name	Capital
AK	Alaska	Juneau
AL	Alabama	Montgomery
FL	Florida	Tallahassee

```ts
// Assume saved data has the
// following structure
type StateType = {
  abbrev: string;
  name: string;
  capital: string;
};
```

```ts
import {
  Query,
  getDocs,
  collection,
  where,
  query,
  QuerySnapshot,
  QueryDocumentSnapshot,
} from "firebase/firestore";
const getFL: Query = query(
  collection(db, "states"),
  where("name", "==", "Florida")
);
getDocs(getFL).then((qs: QuerySnapshot) => {
  qs.forEach((qd: QueryDocumentSnapshot) => {
    const stateData = qd.data() as StateType;
    // More code here to manipulate stateData
  });
});
```

Firestore in TS

CRUD Operations: Fetch Document(s) Where...

SQL

```
// Select tuples satisfying some conditions
SELECT * FROM states WHERE population > 10_000_000
```

states (SQL table)

Name	Capital	Population
California	Sacramento	39_123_612
Michigan	Lansing	8_432_911
Florida	Tallahassee	26_222_943

```ts
// Assume saved data has the
// following structure
type StateType = {
  name: string;
  capital: string;
  population: number;
};
```

```ts
import {
  Query,
  getDocs,
  collection,
  query,
  where,
  QuerySnapshot,
  QueryDocumentSnapshot,
} from "firebase/firestore";
const aboveTenMil: Query = query(
  collection(db, "states"),
  where("population", ">", 10_000_000)
);
getDocs(aboveTenMil).then((qs: QuerySnapshot) => {
  qs.forEach((qd: QueryDocumentSnapshot) => {
    const stateData = qd.data() as StateType;
    // More code here to manipulate stateData
  });
});
```

Firestore in TS

CRUD Operations: Fetch Document(s) Where...

SQL

```
// Select tuples satisfying some conditions
SELECT * FROM states WHERE population > 10_000_000
AND population < 15_000_000
```

states (SQL table)

Name	Capital	Population
California	Sacramento	39_123_612
Michigan	Lansing	8_432_911
Florida	Tallahassee	26_222_943

```ts
// Assume saved data has the
// following structure
type StateType = {
  name: string;
  capital: string;
  population: number;
};
```

```ts
import {
  Query,
  getDocs,
  collection,
  query,
  where,
  QuerySnapshot,
  QueryDocumentSnapshot,
} from "firebase/firestore";
const aboveTenMil: Query = query(
  collection(db, "states"),
  where("population", ">", 10_000_000),
  where("population", "<", 15_000_000)
);
getDocs(aboveTenMil).then((qs: QuerySnapshot) => {
  qs.forEach((qd: QueryDocumentSnapshot) => {
    const stateData = qd.data() as StateType;
    // More code here to manipulate stateData
  });
});
```

Firestore in TS

Available Query Where Operators

Operator	Example	SQL Equivalent
<, <=, ==, >=, >	where("population", ">", 20_000_000)	WHERE population > 20000000
!=	where("name", "!=", "Andy")	WHERE name != "Andy"
in	where("city", "in", ["Ada", "Flint"])	WHERE city == "Ada" OR city == "Flint"
not-in	where("city", "not-in", ["Ada", "Flint"])	WHERE city != "Ada" AND city != "Flint"

Operator	Example (courses must be an ARRAY)
array-contains	// Has this student taken MTH200? where("courses", "array-contains", "MTH200")
array-contains-any	// Has this student taken either MTH200 or STA215? where("courses", "array-contains-any", ["MTH200", "STA215"])

Query Limitations

```
// Multiple .where() on the same field
const q = query(
  collection(__, "states"),
  where("population", ">=", 5_000_000),
  where("population", "<=", 10_000_000)
);
getDocs(q).then(() => {
  /* code */
});
```

OK

```
// Can"t use inequalities on two different fields
query(
  collection(__, "states"),
  where("population", ">=", 5_000_000),
  where("area", "<=", 200_000)
);
```

Not OK

```
// Multiple .where on different fields
// require a composite index on both fields
// At most one inequality comparison!!
const q = query(
  collection(__, "students"),
  where("major", "==", "MATH"),
  where("gpa", ">=", 3.0)
);
getDocs(q).then(/* more code */);
```

OK

CRUD: Update Doc (change a simple field)

```
// Update a record with a known primary key
UPDATE students SET phone = "616-616-6161" WHERE gnumber = "G71884"
```

SQL

SQL table

Gnumber	Name	Phone
G81291	Abby	517-123-4567
G71884	Ally	269-333-4444 → 616-616-6161
G53181	Annie	616-777-3332

```
// Assume saved data has the
// following structure
type StudentType = {
  gnumber: string; // PrimaryKey
  name: string;
  phone: string;
};
```

```
import { doc, updateDoc, DocumentReference } from "firebase/firestore";
// After initialization
const docRef: DocumentReference = doc(db, "students/G71884");
// add a new simple data
updateDoc(docRef, { phone: "616-616-6161" }).then(() => {
  console.debug("Update successful");
});
```

Firestore in TS

CRUD: Update Doc (change a simple field)

```
// Update a record when primary key is UNKNOWN
UPDATE students SET phone = "616-616-6161" WHERE name = "Abby"
```

SQL

SQL table

Gnumber	Name	Phone
G81291	Abby	517-123-4567
G71884	Ally	269-333-4444
G53181	Annie	616-777-3332

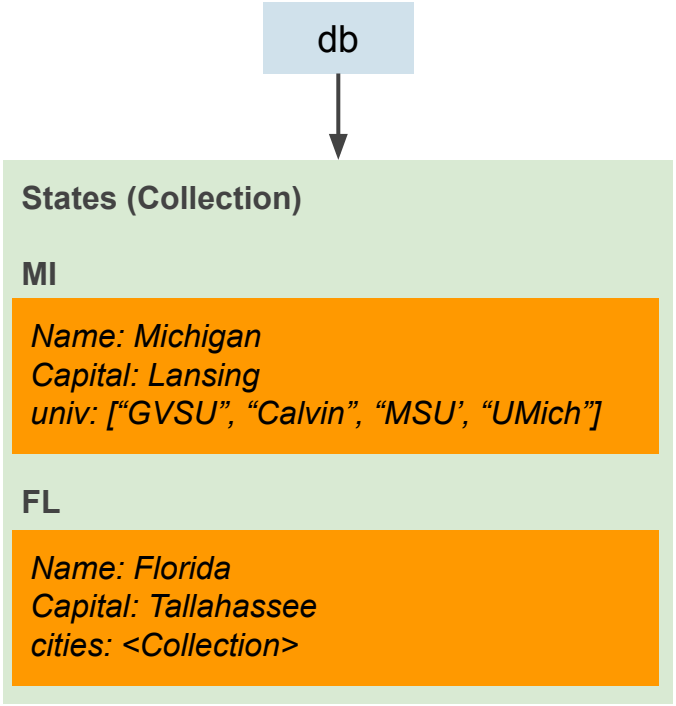
616-616-6161

```
// Assume saved data has the
// following structure
type StudentType = {
  gnumber: string; // PrimaryKey
  name: string;
  phone: string;
};
```

```
import {
  collection, CollectionReference, doc, getDocs, QueryDocumentSnapshot,
  QuerySnapshot, query, updateDoc, where,
} from "firebase/firestore";
const myCol: CollectionReference = collection(db, "students");
const qr = query(myCol, where("name", "==", "Abby"));
getDocs(qr).then((qs: QuerySnapshot) => {
  qs.forEach(async (qd: QueryDocumentSnapshot) => {
    const myDoc = doc(db, qd.id);
    await updateDoc(myDoc, { phone: "616-616-6161" });
  });
});
```

Firestore in TS

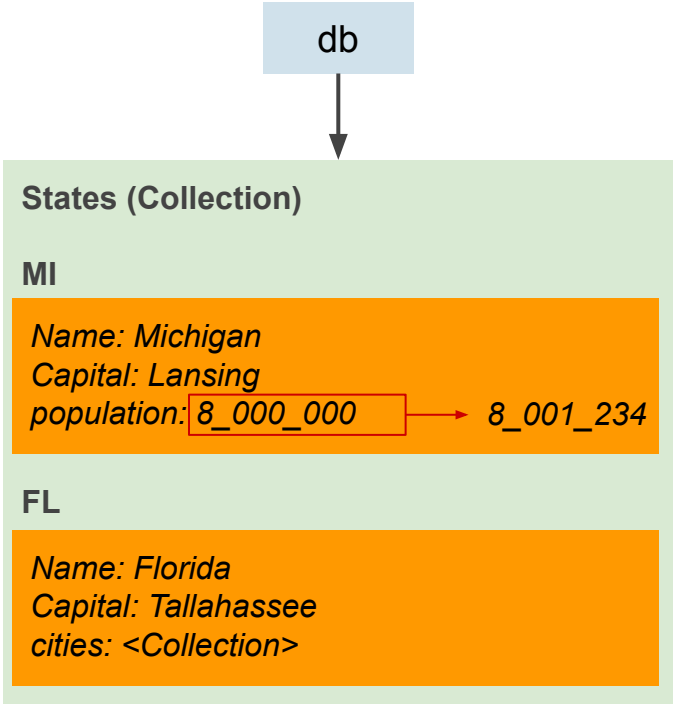
CRUD: Update array field in a Document




```
// After initialization
import {
  updateDoc,
  arrayRemove,
  arrayUnion,
  DocumentReference,
  doc,
} from "firebase/firestore";
const mich: DocumentReference = doc(db, "states/MI");
// add a new JS/TS array
updateDoc(mich, { universities: ["GVSU", "Calvin", "XYZ"] }).then(() => {
  console.debug("Update successful");
});
// updated erroneous entry in the array
updateDoc(mich, {
  universities: arrayRemove("XYZ"),
}).then(() => {
  console.debug("Update successful");
});
// Add more entries in the array
updateDoc(mich, {
  universities: arrayUnion("MSU", "UMich"),
}).then(() => {
  console.debug("Update successful");
});
```

Firestore in TS

CRUD: Update array field in a Document




```
import {
  updateDoc,
  increment,
  DocumentReference,
  doc,
} from "firebase/firestore";
const mich: DocumentReference = doc(db, "states/MI");
updateDoc(mich, {
  // Add 1234 to the current population
  population: increment(1234),
}).then(() => {
  console.debug("Update successful");
});
```

Firestore in TS

CRUD: Delete one Document

```
// Delete a record with a known primary key
DELETE FROM students WHERE gnumber = "G71884"
```
SQL

SQL table

Gnumber	Name	Phone
G81291	Abby	517-123-4567
G71884	Ally	269-333-4444
G53181	Annie	616-777-3332

```
// Assume saved data has the
// following structure
type StudentType = {
  gnumber: string; // PrimaryKey
  name: string;
  phone: string;
};
```

```
import { deleteDoc, doc } from "firebase/firestore";
// Delete the entire document
const toRemove = doc(db, "students/G71884");
deleteDoc(toRemove).then(() => {
  console.debug("Student G71884 removed");
});
```
Firestore in TS

CRUD: Delete one Document (unknown Doc ID)

```
// Update a record when primary key is UNKNOWN
DELETE FROM students WHERE name = "Abby"
```

SQL

SQL table

Gnumber	Name	Phone
G81291	Abby	517-123-4567
G71884	Ally	269-333-4444
G53181	Annie	616-777-3332

```ts
// Assume saved data has the
// following structure
type StudentType = {
  gnumber: string; // PrimaryKey
  name: string;
  phone: string;
};
```

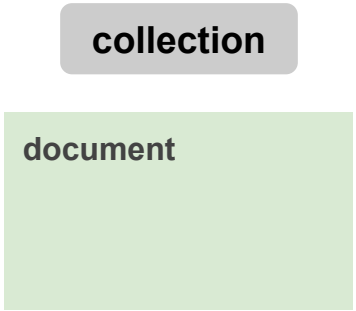
```ts
import { deleteDoc, doc, collection, CollectionReference,
  QueryDocumentSnapshot, QuerySnapshot, query, where, getDocs,
} from "firebase/firestore";
const myCol: CollectionReference = collection(db, "students");
const qr = query(myCol, where("name", "==", "Abby"));
getDocs(qr).then((qs: QuerySnapshot) => {
  qs.forEach(async (qd: QueryDocumentSnapshot) => {
    const myDoc = doc(db, qd.id);
    await deleteDoc(myDoc);
  });
});
```

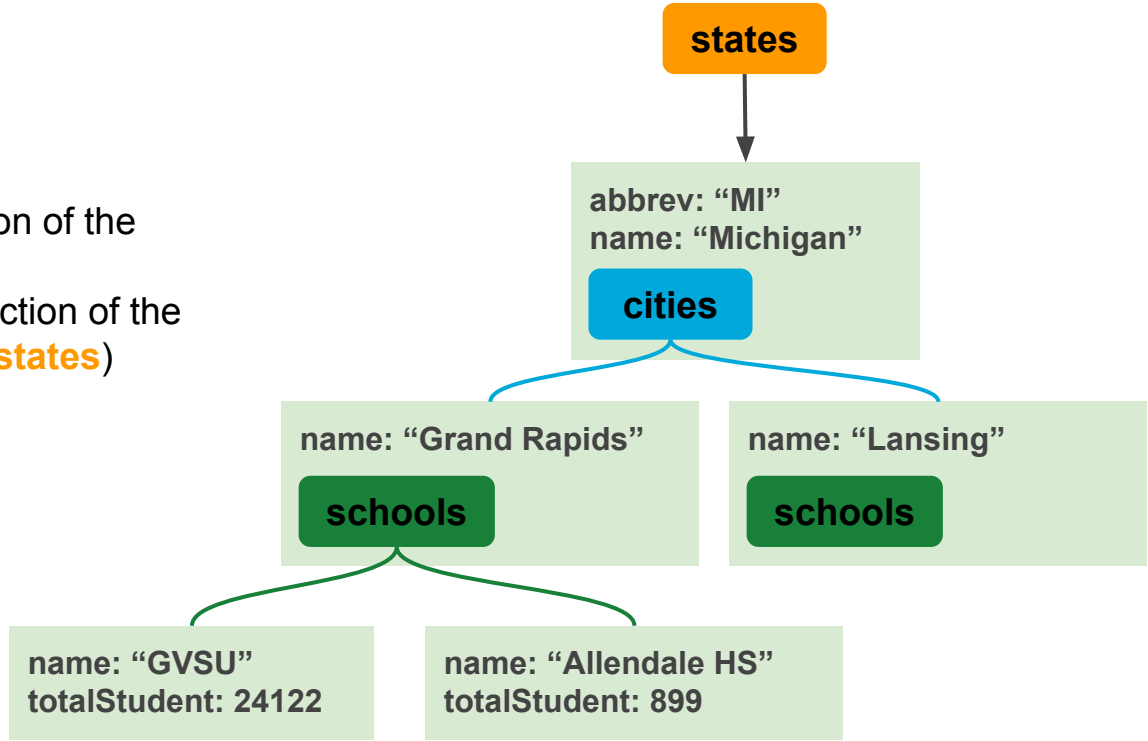
Firestore in TS

SubCollections: One-to-Many Relationship

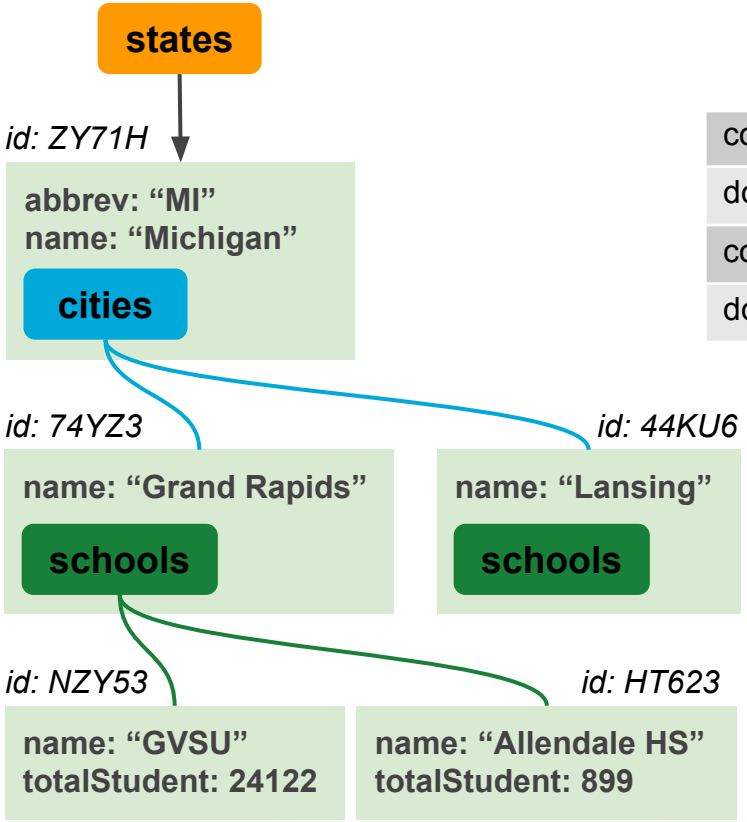
One-to-Many Relationships

- One **state** has many **cities**
- One **city** has many **schools**
- In Firebase Firestore
  - **cities** becomes a subcollection of the **states** collection
  - **schools** becomes a subcollection of the **cities** (sub)collection (under **states**)





Sub-Collections



collection(db, “states/ZY71H/cities”)	Cities in Michigan
doc(db, “states/Z71H/cities/74Y23”)	City of Grand Rapids
collection(db, “states/ZY71H/cities/74Y23/schools”)	Schools in GR
doc(db, “states/ZY71H/cities/74Y23/schools/NZY53”	GVSU

Operations on SubCollections

collection(db, “states/ZY71H/cities”)	Cities in Michigan
doc(db, “states/Z71H/cities/74Y23”)	City of Grand Rapids
collection(db, “states/ZY71H/cities/74Y23/schools”)	Schools in GR
doc(db, “states/ZY71H/cities/74Y23/schools/NZY53”	GVSU

```
// Add a new city in Michigan
const miCities = collection(db, "states/ZY71H/cities");
await addDoc(miCities, {
  name: "Holland",
  /* more details on Holland */
});
```

```
// Update Grand Rapids details
const grDoc = doc(db, "states/Z71H/cities/74Y23");
await updateDoc(grDoc, { subwayAvailable: false });
```

```
// Get all schools in Grand Rapids
const grSchools = collection(db, "states/ZY71H/cities/74Y23/schools");
getDocs(grSchools).then((qs: QuerySnapshot) => {
  qs.forEach((qd: QueryDocumentSnapshot) => {
    const skool = qd.data() as SchoolType;
  });
});
```

Listening to Field Updates on a SINGLE doc

db

buildings

MAK

name: Mackinac Hall
location: Allendale
depts: [“CIS”, “Math”]
numStaffs: 134
rooms: <SubCollection>

PAD

name: Padnos Hall
location: Allendale

DEV

name: DeVos Hall
location: Pew

```
import { doc, onSnapshot, DocumentSnapshot } from "firebase/firestore";
const mac = doc(db, "buildings/MAK");
// Listen to updates on a single document
const unsubscribe = onSnapshot(mac, (snapshot: DocumentSnapshot) => {
  const newData = snapshot.data();
  console.log("Document has been updated to", newData);
});

// Later ...

// Stop listening to changes
unsubscribe();
```

Firestore in TS

Will NOT receive notifications on updates of the doc subcollections (rooms in the example)

Listening to Updates on a Collection of Docs

db

buildings

MAK

name: Mackinac Hall
location: Allendale
depts: [“CIS”, “Math”]
numStaffs: 134
rooms: <SubCollection>

PAD

name: Padnos Hall
location: Allendale

DEV

name: DeVos Hall
location: Pew

```
import { collection, onSnapshot, QuerySnapshot } from "firebase/firestore";
const bldColl = collection(db, "buildings");
const unsubscribe = onSnapshot(bldColl, (s: QuerySnapshot) => {
  for (let chg of s.docChanges()) {
    const newData = chg.doc.data();
    const updateAction = chg.type; // "added", "modified", "removed"
    console.log(chg.doc.id, "has been", updateAction, newData);
  }
});

// Later ...

// Stop listening to changes
unsubscribe();
```

Firestore in TS

Handle listen errors

db

buildings

MAK

name: Mackinac Hall
location: Allendale
depts: [“CIS”, “Math”]
numStaffs: 134
rooms: <SubCollection>

PAD

name: Padnos Hall
location: Allendale

DEV

name: DeVos Hall
location: Pew

Detach listeners when leaving a page or unmounting a component.

```
import { doc, onSnapshot, DocumentSnapshot } from "firebase/firestore";
const mac = doc(db, "buildings/MAK");
// Listen to updates on a single document
const unsubscribe = onSnapshot(
  mac,
  (snapshot: DocumentSnapshot) => {
    const newData = snapshot.data();
    console.log("Document has been updated to", newData);
  },
  (error) => {
    // ...
  }
);
// Later ...

// Stop listening to changes
unsubscribe();
```

Firestore in TS

Firebase Auth: Create A New Account

```
import {
  getAuth,
  createUserWithEmailAndPassword,
  UserCredential,
} from "firebase/auth";
const auth = getAuth();

createUserWithEmailAndPassword(auth, "me@sample.com", "1q2w3e4r5")
  .then((cred: UserCredential) => {
    sendEmailVerification(cred.user);
    console.log("Verification email has been sent to", cred.user?.email);
    auth.signOut();
  })
  .catch((err: any) => {
    console.error("Oops", err);
  });
```

Firebase Auth: Signin With Email

```
import {
  getAuth,
  signInWithEmailAndPassword,
  UserCredential,
} from "firebase/auth";
const auth = getAuth();

signInWithUserWithEmailAndPassword(auth, "me@sample.com", "1q2w3e4r5")
  .then((cred: UserCredential) => {
    if (cred.user?.emailVerified) console.log("Signed in as", cred.user?.email);
    else {
      console.log("Please verify your email first");
      auth.signOut();
    }
  })
  .catch((err: any) => {
    console.error("Oops", err);
  });
```

Firebase Auth: Sign In With Providers

```
import { getAuth, GoogleAuthProvider, signInWithPopup } from "firebase/auth";
const auth = getAuth();

const provider = new GoogleAuthProvider();

signInWithPopup(auth, provider)
  .then((result) => {
    const cred = GoogleAuthProvider.credentialFromResult(result);
    console.log("Signed in as", cred.user?.email);
  })
  .catch((err: any) => {
    console.error("Oops", err);
  });
```

Make sure you have added the login provider in the Authentication Dashboard

JS|TS Modules

Multiple Scripts: name conflicts

```html
<html>
  <body>
    <script src="one.js"></script>
    <script src="two.js"></script>
    <script>
      let msg = "Here";
    </script>
    <script>
      let msg = "I'm here";
    </script>
    <script>
      console.debug(msg);
    </script>
  </body>
</html>
```

```
// one.js
let msg = "Hello";
```

```
// two.js
let msg = "Hello World";
```

- *JS Error: “msg” is already defined*
- *Variables defined in each <script> block are globally visible*

Solution: use Modules (ES6)

```
<html>
  <body>
    <script type="module">
      let msg = "Here";
    </script>
    <script type="module">
      let msg = "I'm here";
    </script>
    <script type="module">
      // does NOT work
      console.debug(msg);
    </script>
  </body>
</html>
```

- *"msg" is undefined*
- *Variables defined in each <script> block are visible only within that block (local)*

```
// one.js
let msg = "Hello";
export { msg }
```

```
// two.js
let msg = "Hello World";
export { msg }
```

```
<html>
  <body>
    <!-- Because the two JS files are imported,
the following two lines are NOT needed
    <script src="one.js" type="module"></script>
    <script src="two.js" type="module"></script> -->
    <script type="module">
      import { msg } from "./one.js";
      import { msg as msg2 } from "./two.js";
      console.debug(msg);
      console.debug(msg2);
    </script>
  </body>
</html>
```

Choices of Modules

- CommonJS (2009): require("module-name") and module.exports = { }
  - Use by NodeJS
- ES6 Modules (2015)
  - import and export (shown as examples in earlier slides)
- AMD: Asynchronous Module Definition
  - RequireJS (supplement to AMD)
- UMD: Universal Module Definition
  - enable apps to use CommonJS and AMD together

Less popular

JS|TS Promise

Promise settlement: resolve() or reject()

```
function nthPrime(nth: number): Promise<number> {
  if (nth < 100_000) {
    // assume prime calculation takes 10 seconds
    return Promise.resolve(a_prime_number_here);
  } else return Promise.reject("Can't compute prime");
}
```

```
console.log("Start");
nthPrime(500).then((pr: number) => {
  console.log("Prime is", pr);
});
.catch((err:any) => {
console.log("Rejected", err);
});
console.log("Here");
```

```
# Watch for order of execution
Start
Here
# if the promise is resolved
# After 10 seconds ...
Prime is 3571
# if the promise is rejected
Rejected Can’t compute prime
```

Demo

JS Promise

- Basic methods: then(), catch(), finally()
- Basics static functions
  - Promise.resolve()
  - Promise.reject()
- Advanced (for handle multiple concurrent promises)
  - Promise.all(array): wait until all the promises in the array are resolved
  - Promise.allSettled(array): wait until all the promises in the array are either resolved or rejected
  - Promise.any(array): wait until ONE of the promises in the array is resolved
  - Promise.race(array): wait until ONE of the promises in the array is either resolved or rejected

Then and then and then and …

```
function nthPrime(nth: number): Promise<number> {
  // more code here
  return Promise.____;
}
```

```
function toRomanNumeral(inputNum: number): string {
  // conversion to Roman numberal
  return _______;
}
```

Return from a then() becomes a Promise to the next then() inline

```
nthPrime(500)
  .then((p: number): string => {
    return toRomanNumeral(p);
  })
  .then((rome: string) => {
    console.log(`Prime in roman numeral ${rome}`);
  });
```

```
// After 1-line return elimination
nthPrime(500)
  .then((p: number): string => toRomanNumeral(p))
  .then((rome: string) => {
    console.log(`Prime in roman numeral ${rome}`);
  });
```

Demo

Then and then and … (promise “unpacked”)

```
function nthPrime(nth: number): Promise<number> {
  // more code here
  return Promise.____;
}
```

```
nthPrime(500)
  .then((p: number): Promise<string> => promNum(p))
  .then((rome: string) => {
    // "unpacked"!!!
    console.log(`Prime in roman numeral ${rome}`);
  });
```

```
function promNum(inputNum: number): Promise<string> {
  // conversion to Roman numberal
  return Promise._______;
}
```

Demo

Promise: with finally

```
function nthPrime(nth: number): Promise<number> {
  // work takes 10 seconds
  return _______;
}
```

```
console.log("Start");
nthPrime(500).then((pr: number) => {
  console.log("Prime is", pr);
});
doMoreWork();
```

```
Start
Partial output of doMoreWork()
# After 10 seconds
Prime is 3571
More output from doMoreWork()
```

```
console.log("Start");
nthPrime(500)
  .then((pr: number) => {
    console.log("Prime is", pr);
  })
  .finally(() => {
    doMoreWork();
  });
```

```
Start
# After 10 seconds
Prime is 3571
Output of doMoreWork()
```

The finally method is used to specify a block of code that will run after the promise is settled, regardless of whether it was resolved or rejected.

async & await

Async functions

```
function nthPrime(nth: number): Promise<number> {
  let thePrime: number;
  // more code here
  return Promise.resolve(thePrime);
}
```

What is the difference between promise functions with and without the async keyword?

```
async function nthPrime(nth: number): Promise<number> {
  let thePrime: number;
  // more code here
  return thePrime; // Promise.resolve() is not required
}
```

The async keyword makes asynchronous functions look and behave more like synchronously way by

- *removing the need for explicit promise creation.*
- *using the await keyword to pause the async function execution*

```
const nthPrime = async (nth: number): Promise<number> => {
  let thePrime: number;
  // more code here
  return thePrime; // Promise.resolve() is not required
};
```

await: rewrite in synchronous style

```
orderPizza(____)
  .then((ord: PizzaOrder) => playWithPal(__, __))
  .then((proof: ProofOfPlay) => makePizza(____))
  .then((box: PizzaBox) => {
    console.log("Open the box and enjoy!");
  })
  .catch((err: any) => {
    console.error("Can't complete order");
  });
```

Await can only be used inside async functions

```
const doPizza = async (): Promise<void> => {
  try {
    const ord: PizzaOrder = await orderPizza(____);
    const proof: ProofOfPlay = await playWithPal(__, __);
    const box: PizzaBox = await makePizza(____);
    console.log("Open the box and enjoy!");
  } catch (err) {
    console.error("Can't complete order");
  }
};
```

Axios & Web Services

Example of Web Services

- Random Users
  - Documentation: https://randomuser.me/documentation
  - Service Endpoint: https://randomuser.me/api
- Random Quotes
  - Documentation: https://github.com/lukePeavey/quotable
  - Service Endpoint: https://api.quotables.io
- A gazillion more Web Services: https://github.com/public-apis/public-apis
  - Pick ones that allow CORS

Example #1: Random User

Browser https://randomuser.me/api

```
type RandUserData = {
  results: Array<RandomUser>;
  info: any;
};
type RandomUser = {
  name: {
    title: string;
    first: string;
    last: string;
  };
  email: string;
};
```

Important: *define these types to match the JSON structure of the API response.*

```
axios
  .request({
    method: "GET",
    url: "https://randomuser.me/api",
  })
  .then((resp: AxiosResponse) => resp.data)
  .then((incoming: RandUserData) => {
    console.log(incoming.info);
    for (let k = 0; k < incoming.results.length; k++) {
      console.log(incoming.results[k]);
    }
  });
```

Example #2: Random Quote

Browser http://api.quotable.io/random

```
type Quote = {
  tags: Array<string>;
  content: string;
  author: string;
  length: number;
};
```

```
axios
  .request({
    method: "GET",
    url: "http://api.quotable.io/random",
  })
  .then((resp: AxiosResponse) => resp.data)
  .then((q: Quote) => {
    console.log(`${q.content} [${q.author}]`);
  });
```

Example #3: Random User with Query Params

Browser https://randomuser.me/api?results=5&nat=gb,fr&inc=name,email,picture

```
type RandUserData = {
  results: Array<RandomUser>;
  info: any;
};
type RandomUser = {
  name: {
    title: string;
    first: string;
    last: string;
  };
  email: string;
};
```

```
axios
  .request({
    method: "GET",
    url: "https://randomuser.me/api",
    params: {
      results: 5,
      nat: "gb,fr",
      inc: "name,email,picture",
    },
  })
  .then((resp: AxiosResponse) => resp.data)
  .then((incoming: RandUserData) => {
    console.log(incoming.info);
    for (let k = 0; k < incoming.results.length; k++) {
      console.log(incoming.results[k]);
    }
  });
```

Example #4: Random Quotes with Query params

Browser http://api.quotable.io/quotes?limit=3

```
type QuoteResponse = {
  count: number;
  results: Array<Quote>;
};
type Quote = {
  tags: Array<string>;
  content: string;
  author: string;
  length: number;
};
```

```
axios
  .request({
    method: "GET",
    url: "http://api.quotable.io/quotes",
    params: {
      limit: 3,
    },
  })
  .then((resp: AxiosResponse) => resp.data)
  .then((qr: QuoteResponse) => {
    console.log(qr.count);
    for (let q of qr.results) {
      console.log(q);
    }
  });
```

Vuetify

Using Color Class Names

Color Usage	Class name	Example
Background Color	bg-{color}	...
Text Color	text-{color}	...
Darker	bg-{color}-darken-{n} text-{color}-darken-{n}	...
Lighter	bg-{color}-lighten-{n} text-{color}-lighten-{n}	...
Accent	bg-{color}-accent-{n} text-{color}-accent-{n}	...

1~4

1~4

1~5

Demo

Input, Selection, and Controls

User Inputs

- v-autocomplete / v-combobox / v-select
- v-checkbox / v-switch
- v-radio-group and v-radio
- v-slider / v-range-slider
- v-form & v-text-field & v-textarea
- v-date-picker
- v-time-picker
- v-menu

Important attribute: v-model

Input Field Validation

```
<template>
    <v-text-field label="Email" :rules="eRules" />
</template>
```

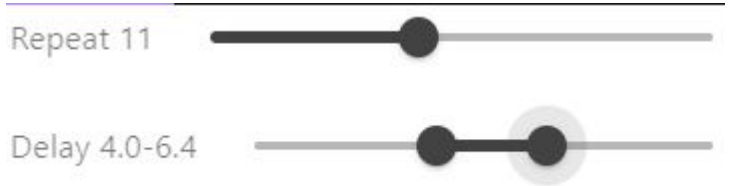
```
<script setup lang="ts">
const eRules = [
  function (x: string): boolean | string {
    if (x.indexOf('@') >= 1) return true
    else return 'Invalid email format'
  },
  function (x: string): boolean | string {
    if (x.endsWith('.edu') || x.endsWith('.org') || x.endsWith('.net')) return true
    else return 'Only .edu/.org/.net accepted'
  }
]
</script>
```

Demo

<v-slider> & <v-range-slider>

```
<template>
  <v-slider v-model="repeat" color="red" :label="`Repeat ${repeat}`" min="1" max="5" />
  <v-range-slider v-model="delay" :label="`Delay ${range}`" step="0.1" min="2" max="5" />
</template>
```

```
<script setup lang="ts">
import { ref, computed } from 'vue'
const repeat = ref(0)
const delay = ref([2.7, 4.3])
const range = computed(() => {
  return delay.value[0].toFixed(1) + '-' + delay.value[1].toFixed(1) + 'secs'
})
</script>
```



Demo

Date & Time Pickers

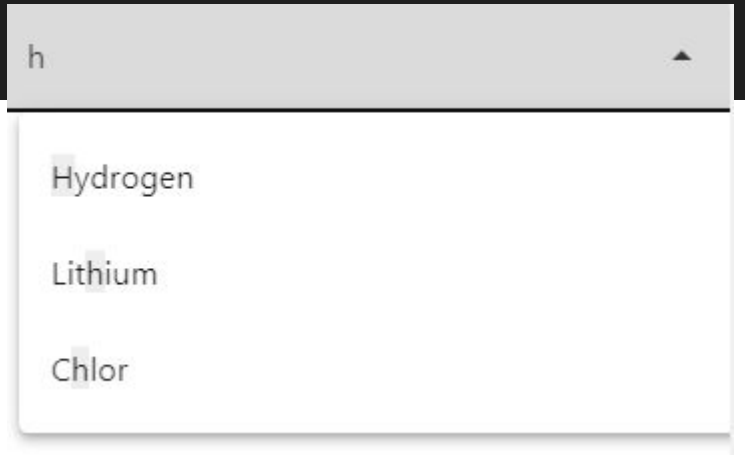
```
<template>
  <v-app>
    <v-main>
      <v-date-picker v-model="selDate" width="200px" />
    </v-main>
  </v-app>
</template>
```

```
<script setup lang="ts">
import { ref } from 'vue'
const selDate = ref(null) // For storing the selected date
</script>
```

Demo

<v-autocomplete> / <v-combobox> / <v-select>

```
<template>
  <v-app>
    <v-main>
      <p>Select your input below</p>
      <v-autocomplete :items="atoms" v-mode="myAtom" />
    </v-main>
  </v-app>
</template>
```




```
<script setup lang="ts">
import { ref } from 'vue'

const myAtom = ref('')
const atoms = ref([
  'Hydrogen',
  'Lithium',
  'Sodium',
  'Pottasium',
  'Rubidium',
  'Chlor',
  'Carbon',
  'Tin',
  'Lead',
  'Gold',
  'Copper'
])
</script>
```

Online playground

<v-radio-group> & <v-radio>

```vue
<template>
  <v-app>
    <v-main>
      <v-radio-group label="Tax Filing Status" v-model="taxStatus">
        <v-radio v-for="(s, opt) in taxFiling" :label="s" :value="opt" :key="s" />
      </v-radio-group>

      <p>Your selection is {{ taxStatus }}</p>
    </v-main>
  </v-app>
</template>
```

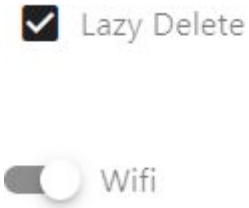
```vue
<script setup lang="ts">
import { ref } from 'vue'

const taxStatus = ref(-1)
const taxFiling = ref([
  'Single',
  'Married Filing Jointly',
  'Married Filing Separately',
  'Head of Household'
])
</script>
```

Demo

<v-checkbox> / <v-switch>

```
<template>
  <v-app>
    <v-main>
      <v-checkbox v-model="lazyDel" label="Lazy" Delete”> </v-checkbox>
      <v-switch v-model="useWifi" label="Wifi"> </v-switch>
    </v-main>
  </v-app>
</template>
```




```
<script setup lang="ts">
import { ref } from 'vue'

const lazyDel = ref(false)
const useWifi = ref(true)
</script>
```

Demo

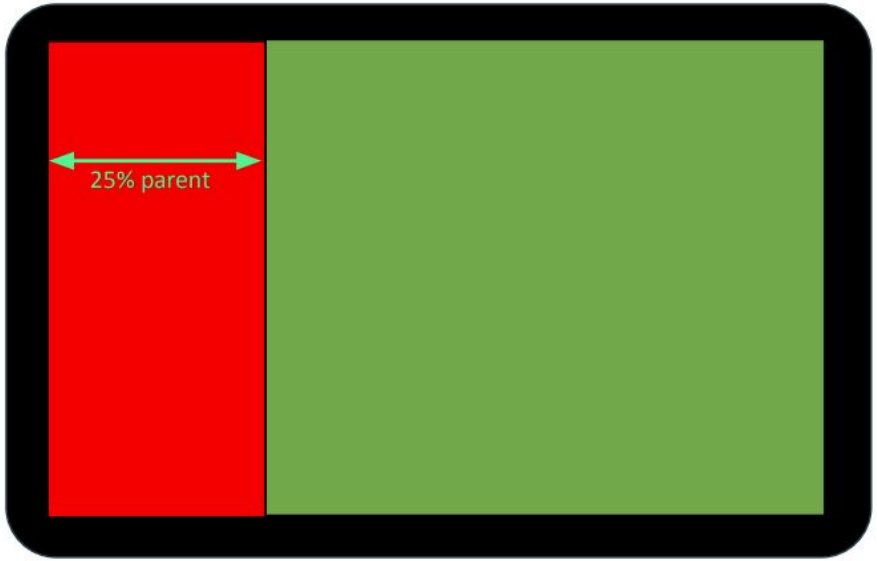
Vuetify Grid System

- Inspired by Bootstrap grid & CSS Flexbox
- Relevant elements: , , ,
  - Warning: row/col does not imply “horizontal”/”vertical” grouping of elements
- Vuetify 12-column layout?
  - Why 12? It is a relatively small number with many integer factors: 2, 3, 4, 6
  - It is easy to divide the screen into
    - Full width (100%)
    - 2 halves (each column is 50% of total screen width => 6/12 each)
    - 3 thirds (33% -> 4/12 each)
    - 4 quarters (25% => 3/12 each)

Online playground

Vuetify Grid Layout (12-column system)

```
<v-container>
  <v-row>
    <v-col cols="3" class="bg-red"> </v-col>
    <v-col class="bg-green"> </v-col>
  </v-row>
</v-container>
```



Size Code and Media breakpoints

Symbolic names for sizes based on 12-column grid layout

Code	Description	Media	Size Range
xs	eXtra Small	Small to large phone	up to 600 px
sm	Small	Small to medium tablet	up to 960 px
md	Medium	Large tablet to laptop	up to 1280 px
lg	Large	Laptop to desktop	up to 1920 px
xl	Extra Large	1080p to 1440p desktop	up to 2560 px
xxl	Extra extra Large	4k and ultra-wide	> 2560 px

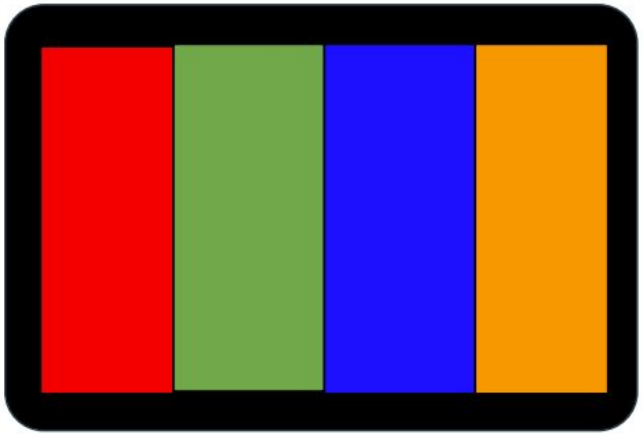
Examples:

- xs=3 ⇒ 25% (3/12) width on any media
- sm=6 ⇒ 50% (6/12) width on tablets or larger media
- md=12 ⇒ full width on desktops
- lg=9 ⇒ 75% (9/12) width on large desktops or above
- xl=2 ⇒ 16% (2/12) width on huge desktops

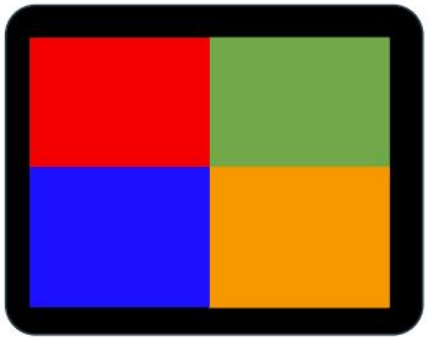
Medium vs. Small

```
<v-container>
  <v-row>
    <v-col md="3" class="red" sm="6"></v-col>
    <v-col md="3" class="green" sm="6"></v-col>
    <v-col md="3" class="blue" sm="6"></v-col>
    <v-col md="3" class="orange" sm="6"></v-col>
  </v-row>
</v-container>
```

On medium screens



On small screens



Size Code and Media breakpoints

```
<v-container>
  <v-row>
    <v-col v-for="url in imageURLs" lg="2" md="3" sm="4" xs="6">
      <img :src="url" />
    </v-col>
  </v-row>
</v-container>
```

Screen Size	Effective Width	# of images per line?
Large	2	
Medium	3	
Small	4	
Extra Small	6	

Online playground

Styling via Class Names

Vuetify provides predefined class names for styling the following properties:

- Material Color classes (background and foreground)
- Space classes (margin & padding)
- Media size breakpoints

Styling Margin/Padding via Class Names

- Vuetify provides predefined classnames (zz-n) as shortcuts for defining margins and paddings
- Regex for these class names:

[mpg] [tblrseaxyc] - [0-16 | n1-16 | auto]

margin / padding/gap

top / bottom / left / right /start / end/ all
x (left&right) / y (top&bottom) / column-gap

Styling Margin/Padding via Class Names
[mpg] [tblrseaxyc] - [0-16 | n1-16 | auto]

Code	Pixels
0	0
1	4px
2	8px
3	12px
4	16px
n1	-4px
n3	-12px

negative number only work for margin.

Vuetify CSS Class	Equivalent CSS Declaration
ma-2	
ml-3	
pt-0	
pb-5	
px-1	
me-n2	

Spacing Class Names

```
<!-- traditional HTML file --->
<span id="msg">Sample Text</span>

<style>
  #msg {
    margin-left: 24px;
  }
</style>
```

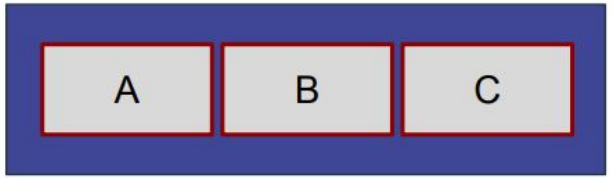
Traditional way

```
<span class="ml-4">Sample Text with left margin</span>
<p class="py-2">text with top and bottom padding…</p>
```

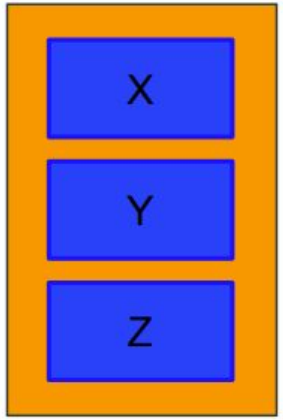
The vuetify way

Row- vs. Column-Oriented Box

```
<v-container>
  <v-row>
    <v-col>A</v-col>
    <v-col>B</v-col>
    <v-col>C</v-col>
  </v-row>
</v-container>
```




```
<v-container>
  <v-row class="flex-column">
    <v-col>X</v-col>
    <v-col>Y</v-col>
    <v-col>Z</v-col>
  </v-row>
</v-container>
```



v-row: justify-* and align-*

CSS3 FlexBox	CSS3 Flexbox Properties	Vuetify Attribute

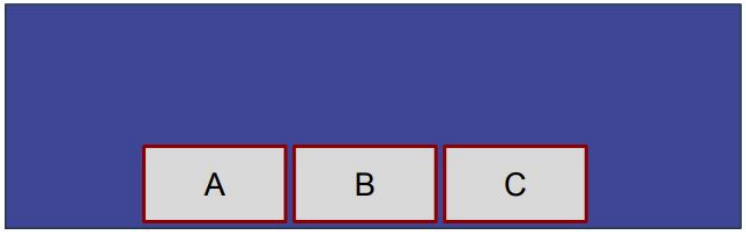
Main-axis ***justification***	`justify-content: flex-start;`
	`justify-content: flex-end;`
	`justify-content: flex-center;`
	`justify-content: flex-space-around;`
	`justify-content: flex-space-between;`
		`justify="start"`
		`justify="end"`
		`justify="center"`
		same as above

Cross-axis ***alignment***	`align-items: flex-start;`
	`align-items: flex-end;`
	`align-items: flex-center;`
	`align-items: flex-baseline;`
		`align="start"`
		`align="end"`
		same as above

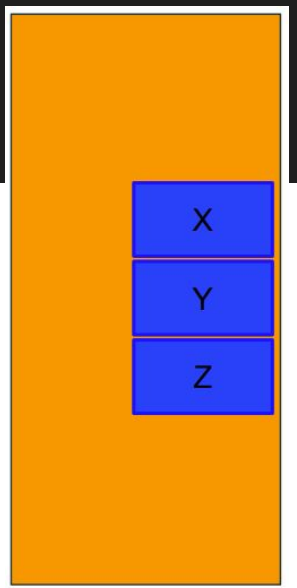
Vuetify accepts both align or align-content as valid attributes

Justification vs. Alignment

```
<v-container>
  <v-row justify="center" align="end">
    <v-col>A</v-col>
    <v-col>B</v-col>
    <v-col>C</v-col>
  </v-row>
</v-container>
```




```
<v-container>
  <v-row class="flex-column" justify="center" align="end">
    <v-col>X</v-col>
    <v-col>Y</v-col>
    <v-col>Z</v-col>
  </v-row>
</v-container>
```



Containment: Buttons

```
<v-btn icon="mdi-account" size="x-small"></v-btn>
```

The **size** property is used to control the size of the button and scales with density.

```
<v-btn density="default" icon="mdi-open-in-new"></v-btn>
```

The **density** prop is used to control the vertical space that the button takes up.

```
<v-btn append-icon="mdi-account" size="x-small">Tom</v-btn>
```

```
<v-btn prepend-icon="mdi-account" size="x-small">Tom</v-btn>
```

Props like prepend-icon and append-icon offer a straightforward approach to incorporate positioned icons,

```
<v-btn append-icon="mdi-account-circle" prepend-icon="mdi-check-circle">
  <template v-slot:prepend>
    <v-icon color="success"></v-icon>
  </template>
  Button
  <template v-slot:append>
    <v-icon color="warning"></v-icon>
  </template>
</v-btn>
```

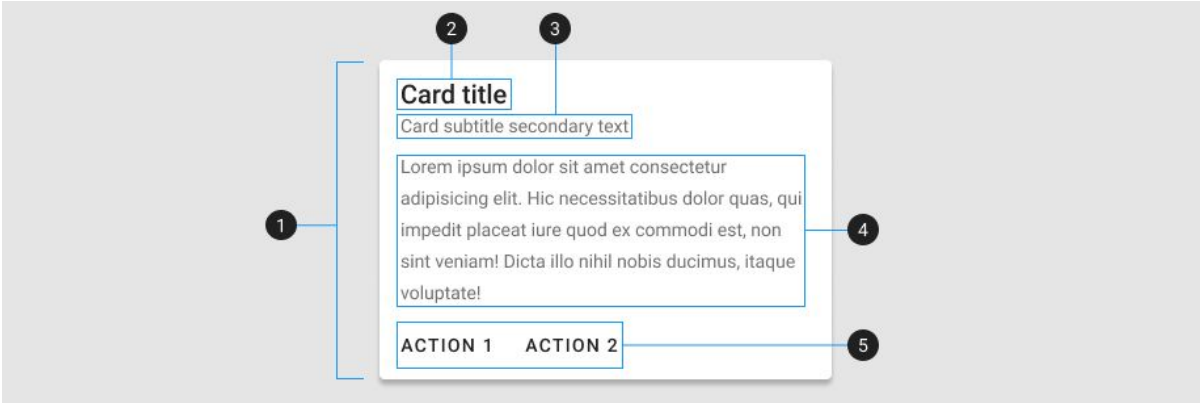
For the prepend-icon and append-icon, the corresponding slots are prepend and append

Demo

Variants

Value	Example
elevated	BUTTON
flat	BUTTON
tonal	BUTTON
outlined	BUTTON
text	BUTTON
plain	BUTTON

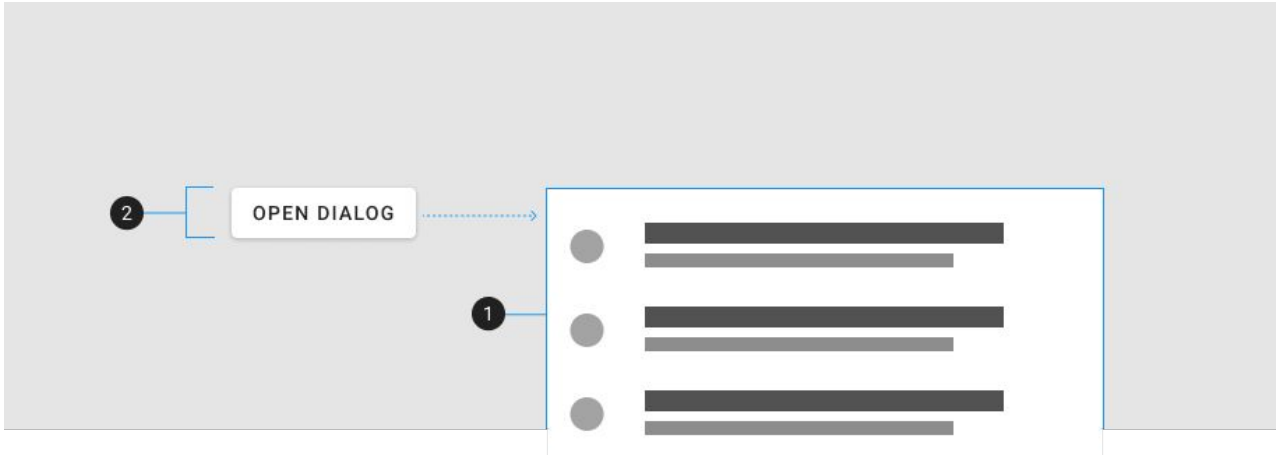
Containment: Cards



Element / Area	Description
1. Container	The Card container holds all `v-card` components. Composed of 3 major parts: `v-card-item`, `v-card-text`, and `v-card-actions`
2. Title (optional)	A heading with increased **font-size**
3. Subtitle (optional)	A subheading with a lower emphasis text color
4. Text (optional)	A content area with a lower emphasis text color
5. Actions (optional)	A content area that typically contains one or more v-btn components

Demo

Containment: Dialogs

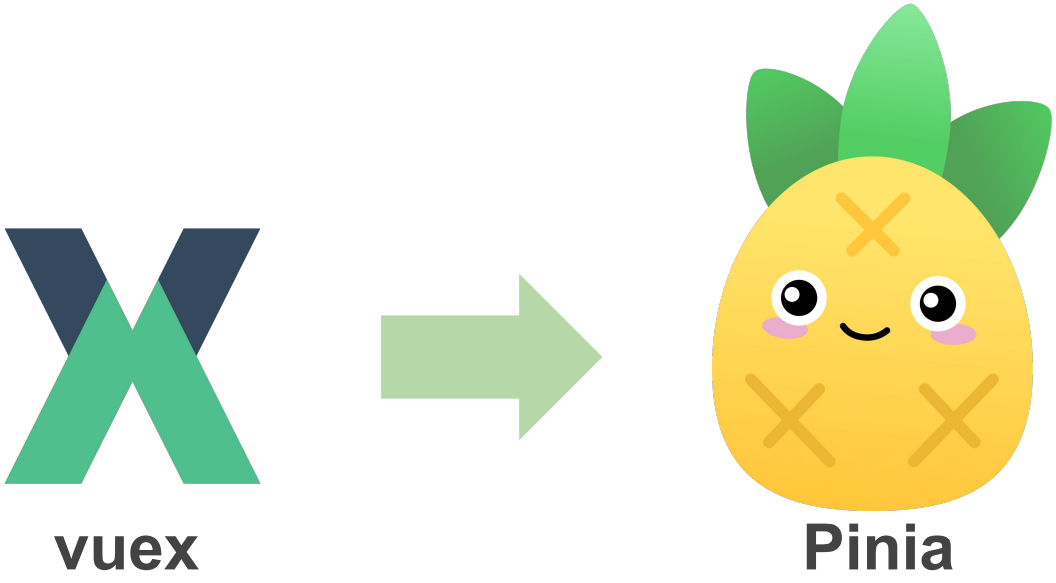


Element / Area	Description
1. Container	The dialog's content that animates from the activator
2. Activator	The element that activates the dialog

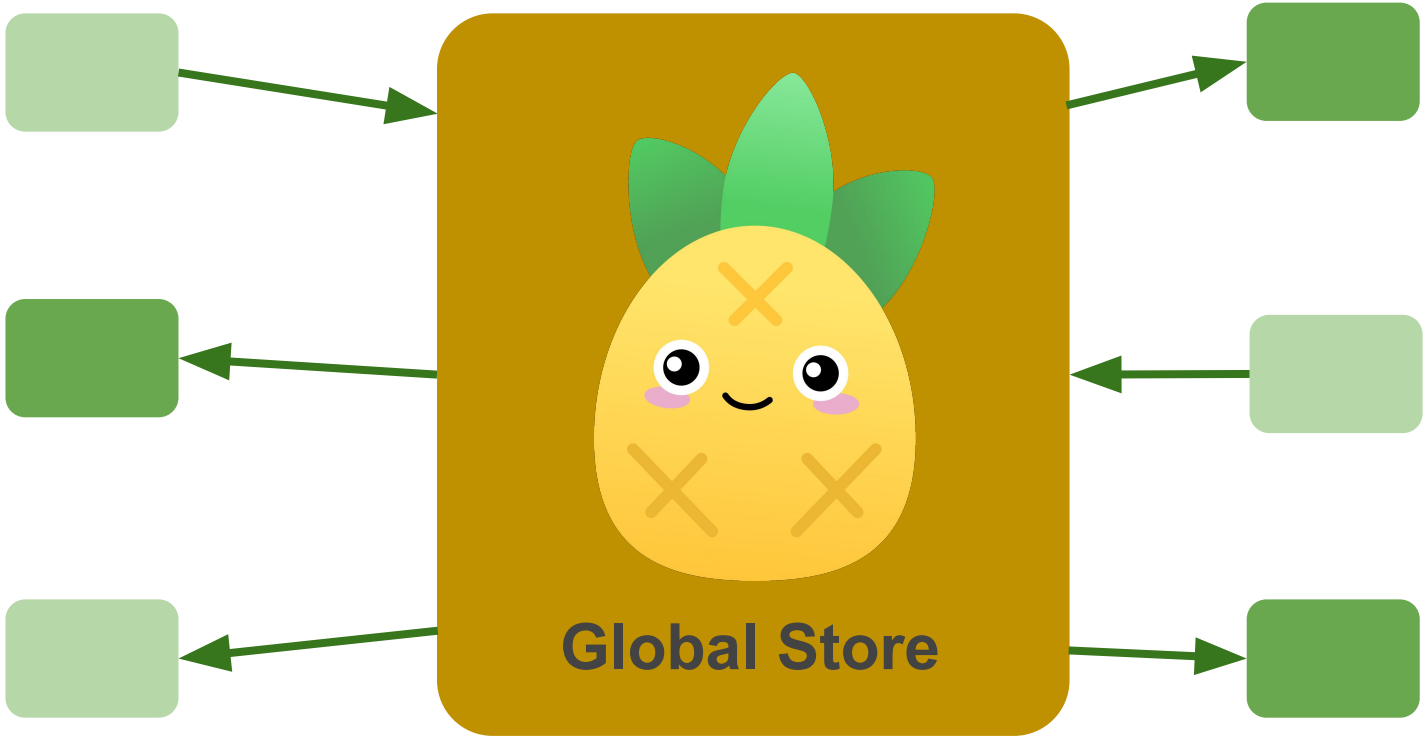
Demo

What is Pinia?

Pinia is a state management solution created by Vue core team member Eduardo San Martin Morote, who also created Vue Router.



Pinia: state management solution



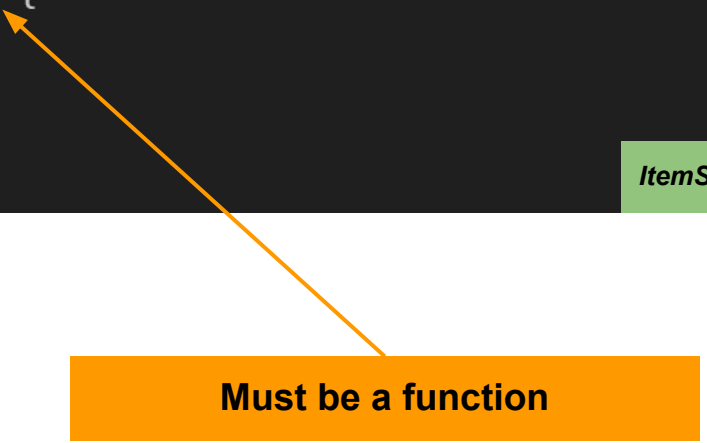
Pinia State

Define Pinia State

```
import { defineStore } from "pinia";

export const useItemStore = defineStore("ItemStore", {
  state: () => {
    return {};
  },
});
```

ItemStore.ts



Define Pinia State

```ts
export const useItemStore = defineStore("ItemStore", {
  state: () => {
    return {
      category: "Electronics",
      products: [
        "Gamer's Delight Laptop",
        "SmartTech Pro Phone",
        "Rapid Dual USB-C Charger",
      ],
      inStock: true,
    };
  },
});
```

ItemStore.ts

```vue
<script setup lang="ts">
import { useItemStore } from "./stores/ItemStore";

const itemStore = useItemStore();
console.log(itemStore.category);
</script>
```

AnyVueComponent.vue

Access Pinia State

AnyVueComponent.vue

```
<script setup lang="ts">
import { useItemStore } from "./stores/ItemStore";

const itemStore = useItemStore();
console.log(itemStore.ca);
</script>
```

```
category
$onAction
$patch
```

```
(property) category: string
```

State is TypeSafe

Access Pinia State

AnyVueComponent.vue

```vue
<script setup lang="ts">
import { useItemStore } from "./stores/useItemStore";

const { category } = useItemStore();
console.log(category);
</script>
```

Can de-structure state from store

*category is **no longer reactive***

Access Pinia State

AnyVueComponent.vue

```vue
<script setup>
import { useItemStore } from "./stores/useItemStore";
import { storeToRefs } from "pinia";

const { category } = storeToRefs(useItemStore());
console.log(category.value);
</script>
```

*convert store to **refs** to maintain reactivity*

Can de-structure state from store

Update Pinia State

AnyVueComponent.vue

```vue
<script setup>
import { useItemStore } from "./stores/useItemStore";
import { storeToRefs } from "pinia";

const { category } = storeToRefs(useItemStore());
category.value = "Groceries";
</script>
```

State can be directly updated

AnyVueComponent.vue

```vue
<script setup lang="ts">
import { useItemStore } from "./stores/useItemStore";

const itemStore = useItemStore();
itemStore.category = "Groceries";
</script>
```

If you don't de-structure, you don't need .value

Two-way Binding

```vue
<script setup lang="ts">
import { useItemStore } from "./stores/ItemStore";
import { storeToRefs } from "pinia";
const { category } = storeToRefs(useItemStore());
</script>

<template>
  <input v-model="category" type="text" />
</template>
```

AnyVueComponent.vue

Demo

v-for

ItemStore.ts

```
import { defineStore } from "pinia";
import products from "../data/products.json";

export const useItemStore = defineStore("ItemStore", {
  state: () => {
    return { products };
  }
});
```

AnyVueComponent.vue

```
<script setup lang="ts">
import { useItemStore } from "./stores/ItemStore";
const items = useItemStore();
</script>

<template>
  <ul v-for="(item, idx) in items.products" :key="idx" :item="item">
    <li>{{ item.name }}</li>
  </ul>
</template>
```

Demo

Pinia Actions

Pinia Actions

ItemStore.ts

```
import { defineStore } from "pinia";
import products from "../data/products.json";

export const useItemStore = defineStore("ItemStore", {
  state: () => {
    return { products };
  },

  actions: {
    fill() {},
  },
});
```

Define actions as methods on the actions option

Pinia Actions

UserStore.ts

```ts
import { defineStore } from "pinia";

export const useUserStore = defineStore("UserStore", {
  state: () => {
    return { users: [] };
  },

  actions: {
    async fill() {
      const res = await fetch(
        "https://randomuser.me/api?results=5&nat=gb,fr
      );
      this.users = await res.json();
    },
  },
});
```

```ts
  actions: {
    async fill() {
      const res = await fetch(
        "https://randomuser.me/api?results=5&nat=gb,fr&inc=name,email,picture"
      );
      this.users = await res.json();
      this.someOtherAction();
    },
    async someOtherAction() {},
  },
});
```

Access state with `this`

Access other actions with `this`

Pinia Actions

user.ts

```ts
type User = {
  name: {
    first: string;
    last: string;
    title: string;
  };
  email: string;
  picture: {
    large: string;
    medium: string;
    thumbnail: string;
  };
};
type RandomUser = {
  results: Array<User>;
};
export type { User, RandomUser };
```

UserStore.ts

```ts
import { defineStore } from "pinia";
import { RandomUser } from "../types/user";

export const useUserStore = defineStore("UserStore", {
  state: () => {
    return { users: { results: [] } as RandomUser };
  },

  actions: {
    async fill() {
      const res = await fetch(
        "https://randomuser.me/api?results=5&nat=gb,fr&inc=name,email,picture"
      );
      this.users = await res.json();
      console.log(this.users);
    },
  },
});
```

Pinia Actions

AnyVueComponent.vue

```
<script setup lang="ts">
import { useUserStore } from "./stores/UserStore";
const userStore = useUserStore();
userStore.fill();
</script>

<template>
  <ul v-for="(u, idx) in userStore.users.results" :key="idx" :u="u">
    <li>{{ u.name.first }}. {{ u.name.last }}</li>
  </ul>
</template>
```

Demo

Pinia Actions

```
<script setup lang="ts">
import { useUserStore } from "./stores/UserStore";
const { fill } = useUserStore();
fill();
</script>
```

de-structure

```
<script setup lang="ts">
import { useUserStore } from "./stores/UserStore";
import { storeToRefs } from "pinia";
const { fill } = storeToRefs(useUserStore());
fill();
</script>
```

won't work

Demo

Pinia Actions: Example

CartStore.ts

```ts
import { defineStore } from "pinia";

export const useCartStore = defineStore("CartStore", {
  state: () => {
    return {
      items: [],
    };
  },
  actions: {
    addItem(itemId, count) {
      // set the count for the proper item in the state above
    },
  },
});
```

Item.vue

```vue
<button @click="addItem(product.id, $event)">Add Product to Cart</button>
```

Useful for updating state based on user interaction

Pinia Getters

What are Pinia Getters?

Equivalent of computed props on a component

Must explicitly type the return

Access state with `this`

UserStore.ts

```ts
import { defineStore } from "pinia";
import { RandomUser } from "../types/user";

export const useUserStore = defineStore("UserStore", {
  state: () => {
    return { users: {} as RandomUser };
  },
  getters: {
    count(): number {
      return this.users.results.length;
    },
  },
  actions: {
    async fill() {
      const res = await fetch(
        "https://randomuser.me/api?results=5&nat=gb,fr&inc=name,email,picture"
      );
      this.users = await res.json();
    },
  },
});
```

Pinia Getters

Access state with `state`

```
getters: {
  count(state): number {
    return state.users.results.length;
  },
},
```

No need to explicitly type return

```
getters: {
  count: (state) => state.users.results.length,
},
```

single line arrow functions

Access other getters on `this`

```
getters: {
  count: (state) => state.users.results.length,
  doubleCount(): number {
    return this.count * 2;
  },

  findUserByFirstName: (state) => (first: string) => {
    return state.users.results.find(
      (user: User) => user.name.first === first
    );
  },
},
```

Accept arguments by returning a function

Access Getters

as a property

```
<script setup lang="ts">
import { useUserStore } from "./stores/UserStore";
const userStore = useUserStore();
console.log(userStore.count);
</script>
```

de-structure

```
<script setup lang="ts">
import { useUserStore } from "./stores/UserStore";
import { storeToRefs } from "pinia";
const { count } = storeToRefs(useUserStore());
console.log(count.value);
</script>
```

Can de-structure getters from store but must use `storeToRefs`

Demo

The 'Patch Object' in Pinia

```ts
import { defineStore } from "pinia";

export const useUserStore = defineStore("userStore", {
  state: () => ({
    name: "",
    age: 0,
    email: "",
  }),
  // actions, getters, etc.
});
```

UserStore.ts

```vue
<script setup lang="ts">
import { onMounted } from "vue";
import { useUserStore } from "./stores/UserStore";
const userStore = useUserStore();

onMounted(() => {
  userStore.$subscribe((mutation, state) => {
    if (mutation.type === "patch object") {
      console.log("Patch object:", mutation.payload);
    }
  });
  userStore.$patch({
    name: "John Doe",
    email: "john@example.com",
  });
});
</script>

<template>
  <ul>
    <li>{{ userStore.name }}</li>
    <li>{{ userStore.age }}</li>
    <li>{{ userStore.email }}</li>
  </ul>
</template>
```

AnyVueComponent.vue

Demo

Online Doc