

# IndexedDB with Dart

CRUD implementation in Dart

- [JS Object to Dart Object](#)
  - [jsfy](#)
  - [dartify](#)
- [dart2js project](#)
  - [Compilation Process](#)
- [foobar1 project](#)
  - [What Is idb\\_shim?](#)
  - [Name Conflicts in Dart](#)
- [HTML \(GUI\)](#)
  - [Connecting HTML and Dart](#)
- [Database Access](#)
  - [CRUD](#)
- [foobar2 project](#)
  - [Web Application Directories](#)
  - [webdev](#)
  - [Development Workflow](#)
- [Databases](#)
  - [Decision Framework](#)
  - [IndexedDB Limitations](#)

# JS Object to Dart Object

---

- From a web API or JS function, we get a JS Object.

```
var jsObject = {'name': 'John', 'age': 25, 'active': true};
```

- Dart has `Map<String, dynamic>` to represent a JSON object.

```
Map<String, dynamic> dartMap = {name: John, age: 25, active: true}
```

Dart supports `dartify` and `jsify` to convert from a JS Object to a Dart Map.

Function	Input Type	Output Type	Main Purpose
<code>dartify</code>	JavaScript Object/Array	Dart Map/List (dynamic)	Converts JavaScript objects to Dart (interop)
<code>jsify</code>	Dart Map, List, primitive types	JavaScript Object/Array	Converts Dart objects to JavaScript (interop)

It is essential to understand that they are not intended for serialization, as there are dedicated functions for this purpose.

Function	Input Type	Output Type	Main Purpose
<code>jsonDecode</code>	JSON-formatted String	Dart Map/List (dynamic)	Parses JSON string to Dart object
<code>jsonEncode</code>	Dart Map, List, primitive types	JSON-formatted String	Encodes Dart objects to JSON string

# jsfy

---

- **Purpose:** Converts Dart collections (Map, List) and primitives into JavaScript objects or arrays.
- **Usage:** Used when you need to interact with `JavaScript APIs` or code from Dart (especially in Flutter Web or Dart Web).
- **Conversion:** Deeply converts all nested Maps and Lists. Only supported types (Map, List, numbers, strings, booleans, and null) are handled.

## Example of jsify

```
import 'dart:js_util';

// Dart Map
Map<String, dynamic> dartData = {
  'status': 'success',
  'count': 10,
  'items': ['apple', 'banana', 'orange']
};

// Convert to JavaScript object
var jsData = jsify(dartData);

// Now can be passed to JavaScript functions
// callJavaScriptFunction(jsData);
```

## *Usage of jsonDecode and jsonEncode*

The `jsonDecode` decodes the JSON String, and the `jsonEncode` makes a Dart map from a string.

```
// jsonDecode example:  
import 'dart:convert';  
var jsonString = '{"foo":1,"bar":true}';  
// Produces Dart Map  
var dartMap2 = jsonDecode(jsonString);  
  
// jsonEncode example:  
import 'dart:convert';  
var dartMap3 = {'foo': 1, 'bar': true};  
// Produces String: '{"foo":1,"bar":true}'  
var jsonStr = jsonEncode(dartMap3);
```



# dartify

---

- **Purpose:** Converts JavaScript primitives, arrays, and objects into Dart types (`int`, `bool`, `String`, `List`, `Map`).
- **Usage:** Used when you receive data from JavaScript APIs or need to process JavaScript objects in Dart (especially in Flutter Web or Dart Web interop).
- **Conversion:** Deeply converts all nested arrays and objects into Dart collections.

## Example of dartify

```
import 'dart:js_util';

// JavaScript object (from web API or JS function)
var jsObject = {'name': 'John', 'age': 25, 'active': true};

// Convert to Dart Map
Map<String, dynamic> dartMap = dartify(jsObject);
print(dartMap); // {name: John, age: 25, active: true}

// Now you can use Dart operations
print(dartMap['name']); // John
dartMap['city'] = 'Seoul'; // Add new key-value pair
```

# dart2js project

---

- In this project, we discuss how to use Dart for the generation of JavaScript code.
- Examples are in the `indexeddb/dart2js` directory.

# *Three Steps to get JavaScript from Dart source*

1. Write Dart code.

```
// main.dart  
void main() {  
  print('Hello!');  
}
```



2. Compile the code "dart compile"

```
dart compile js main.dart
```



The compiler generates JavaScript (main.dart.js):

```
function main() {  
  console.log('Hello!');  
}  
// THE MAGIC LINE (added by  
// compiler at the end):  
main(); // ← THIS CALLS MAIN!
```



3. Load the JavaScript in the HTML.

```
<script src="main.dart.js"> </script>
```



The web browser executes the JavaScript code.

- Downloads main.dart.js
- Runs the JavaScript
- Reaches main(); at the end
- Your main() function executes

## *Dart JS program structure*

```
├── lib
├── pubspec.yaml
└── web
    ├── index.html
    ├── main.dart
    └── main.dart.js
```

- The `lib` directory contains Dart code.
- The `web` directory contains JavaScript code.

# Compilation Process

---

- Step 1: Get dependencies (one time)

```
dart pub get
```

- Step 2: Run compiler

```
dart compile js web/main.dart -o web/main.dart.js
```

- Step 3: Run local web server

# Options for Optimization

```
# Development (no optimization) - fastest compilation
dart compile js web/main.dart -o web/main.dart.js
# Optimized for production (default is -O1)
dart compile js -O2 web/main.dart -o web/main.dart.js
dart compile js -O3 web/main.dart -o web/main.dart.js # Maximum optimization
dart compile js -O4 web/main.dart -o web/main.dart.js # Aggressive optimization
```

# Options for debugging

```
# Generate source maps for debugging
dart compile js --source-map web/main.dart -o web/main.dart.js
# Include type checks (helps catch errors)
dart compile js --enable-asserts web/main.dart -o web/main.dart.js
# Verbose output to see what's happening
dart compile js --verbose web/main.dart -o web/main.dart.js
# Development mode (faster compilation, larger output)
dart compile js --no-minify web/main.dart -o web/main.dart.js
```



*Run local web server.*

- Use VSCode "Live Server" extension.
- Use Python: `python3 -m http.server`.
  - `python3 -m http.server 9000` for specifying the port to use.

*Open* `web/index.html`

- Do not open the HTML file directly from a web browser.

# foobar1 project




---






- In this project, we use `idb_shim` (indexedDB shim) and `web` package instead.
- The `dart:html` library was a core Dart library used for web-based applications and is now deprecated.

# What Is idb\_shim?

---

*A platform abstraction layer (shim or adjustment) for IndexedDB:*

-  Works in browser  
(`idb_browser.dart`)
-  Works in Dart VM using memory store (`idb_memory.dart`)
-  Unified API for storage

-  Dart-native IndexedDB API
-  Works in browser and Dart VM
-  Test-friendly (in-memory DB)
-  Clean Future-based async
-  No DOM or UI APIs

-  Raw bindings to **Web APIs**
- Access to:
  - `document`, `window`, `console`
  - DOM elements (`HTMLButtonElement`, etc.)
-  Not testable in Dart VM
-  Low-level, JS-like code

Use Case	idb_shim	package:web
Cross-platform DB logic	✓	✗
IndexedDB (clean/testable)	✓	✗
DOM manipulation	✗	✓
UI interaction	✗	✓

- `idb_shim`: High-level Dart Programming
- `package:web`: Low-Level JS Programming

## *dart:html vs idb\_shim*

Use Case	dart:html	idb_shim
Works in browser	✓	✓
Works in Dart VM	✗	✓
Works in unit tests (CI)	✗	✓
No JavaScript interop	✗	✓
Dart-native types & Futures	✗	✓

- Don't use `dart:html` in your project!

# Example

```
import 'package:idb_shim/idb.dart';
import 'package:idb_shim/idb_browser.dart';
import 'package:web/web.dart' hide Request;

// idb_shim - DB Logic
final IdbFactory idbFactory = getIdbFactory();
final db = await idbFactory.open('myDB', version: 1,
  onUpgradeNeeded: (e) {
    e.database.createObjectStore('store');
  });

// web - UI Logic
final outputDiv = document.querySelector('#output') as HTMLDivElement?;
```



# Name Conflicts in Dart

---

When using:

```
import 'package:web/web.dart';  
import 'package:idb_shim/idb.dart';
```

Dart sees **conflicting types**:

- Request and Event

Both packages define them.

## *Conflict Example*

Type	From web.dart	From idb_shim
Request	Fetch API / DOM Request	IndexedDB Request
Event	DOM Event (UI, network)	IndexedDB Transaction Event

- Dart **does not allow duplicate type names** in the same file.

## *The Fix: Use hide*

```
import 'package:web/web.dart' hide Request, Event as web;  
import 'package:idb_shim/idb.dart';
```

- Only `idb_shim`'s `Request` and `Event` are used
- `package:web` gives you IndexedDB, DOM, etc. – minus conflict
  - We can use `web:Request` to access `Request` in `web`.

# HTML (GUI)

---

```
<label for="key">Key:</label>
<input type="text" id="key" placeholder="Enter key (number)" />

<label for="foo">Foo:</label>
<input type="text" id="foo" placeholder="Enter foo" />

<label for="bar">Bar:</label>
<input type="number" id="bar" placeholder="Enter bar (number)" />

<div>
  <button id="create">Create</button>
  <button id="read">Read</button>
  <button id="update">Update</button>
  <button id="delete">Delete</button>
</div>

<div id="output" aria-live="polite" aria-atomic="true"></div>
```

# Connecting HTML and Dart

---

## *HTML*

```
<div id="output" aria-live="polite" aria-atomic="true"></div>
```

## *Accessing the HTML component*

```
void showOutput(String msg) {  
  final outputDiv = document.querySelector('#output');  
  if (outputDiv != null) {  
    final p = HTMLParagraphElement();  
    p.textContent = msg;  
    outputDiv.appendChild(p);  
  }  
}
```

## *JavaScript comparison*

```
function showOutput(message) {  
  const output = document.getElementById('output');  
  output.innerHTML  
    += `

${JSON.stringify(message, null, 2)}



---

`;  
}
```

- Dart: Uses typed DOM APIs ( `HTMLParagraphElement` ) for type-safe and clarity.
- JavaScript: Uses `innerHTML`, which is shorter but less safe.

## *Three steps for the connection*

- Step 1: Placeholder and buttons in the HTML

```
<label for="key">Key:</label>  
<input type="text" id="key"  
  placeholder="Enter key (number)" />
```

```
<button id="create">Create</button>
```

- Step 2: Get the HTML element in Dart and Use it

```
final createBtn = document.querySelector('#create');
```

- In this code, we use querySelector function to get an HTML element.
- We need to check the nullability, check the type using the is operator, and convert the type using the as operator.



```

final createBtn = document.querySelector('#create');

void handleCreate() async {
    if (inputFoo == null ||
        inputBar == null ||
        inputKey == null)
        return;
    final data = {
        'foo': inputFoo is HTMLInputElement ? inputFoo.value : '',
        'bar': int.tryParse(inputBar is HTMLInputElement
            ? inputBar.value : '0')
    };
    final key = await create(data);
    (inputKey as HTMLInputElement).value = key.toString();
}

```

- However the code is ugly with many tychecking and conversion.

*We can solve this issue by type conversion to Dart type.*

- we need to make sure  
HTMLButtonElement is nullable.

```
final createBtn = document.querySelector('#create') as HTMLButtonElement?;

void handleCreate() async {
  if (inputFoo == null || inputBar == null || inputKey == null) return;
  final data = {
    'foo': inputFoo.value,
    'bar': int.tryParse(inputBar.value) ?? 0,
  };
  final key = await create(data);
  inputKey.value = key.toString();
  showOutput('Created with key $key: $data');
}
```

- Step 3: Connect the Dart function with JavaScript event handler.

```
createBtn?.onclick = handleCreate.toJS;  
readBtn?.onclick = handleRead.toJS;  
updateBtn?.onclick = handleUpdate.toJS;  
deleteBtn?.onclick = handleDelete.toJS;
```

# Database Access

---

```
const String dbName = "myDB";
const String storeName = "myStore";

Future<idb.Database> openDb() async {
  final idbFactory = idbFactoryBrowser;
  return await
    idbFactory.open(dbName, version: 1, onUpgradeNeeded:
      (idb.VersionChangeEvent e) {
        final db = (e.target as idb.Request).result as idb.Database;
        if (!db.objectStoreNames.contains(storeName)) {
          db.createObjectStore(storeName, autoIncrement: true);
        } // If statement
      } // Lambda expression
    ); // idbFactor.open
}
```

# JavaScript comparison

```
let db;
const dbName = 'recordsDB';
const storeName = 'records';
// Initialize database
const request = indexedDB.open(dbName, 1);
request.onerror = (event) => {
  console.error("Database error:", event.target.error);
};
request.onsuccess = (event) => {
  db = event.target.result;
  console.log("Database opened successfully");
};
request.onupgradeneeded = (event) => {
  const db = event.target.result;
  if (!db.objectStoreNames.contains(storeName)) {
    db.createObjectStore(storeName, { keyPath: 'id', autoIncrement: true });
  }
};
```

# CRUD

---

Operation	Dart	JavaScript
<b>CREATE</b>	<code>store.add(data)</code>	<code>store.add(data)</code>
<b>READ</b>	<code>store.getObject(key)</code> (Dart-specific)	<code>store.get(key)</code>
<b>READ (ALL)</b>	<code>store.getAll()</code>	<code>store.getAll()</code>
<b>UPDATE</b>	<code>store.put(data, key)</code>	<code>store.put(data, key)</code>
<b>DELETE</b>	<code>store.delete(key)</code>	<code>store.delete(key)</code>

- Almost the same as both use Dart mimics the IndexedDB API.

# CREATE

```
Future<int> create(Map<String, dynamic> data) async {  
  final db = await openDb();  
  final txn = db.transaction(storeName, 'readwrite');  
  final store = txn.objectStore(storeName);  
  final key = await store.add(data);  
  await txn.completed;  
  return key as int;  
}
```

```
function addRecord() {  
  const transaction = db.transaction([storeName], 'readwrite');  
  const store = transaction.objectStore(storeName);  
  const request = store.add(data);  
  request.onsuccess = () => {  
    showOutput(data);  
  };  
}
```

# READ

```
Future<Map<String, dynamic>?> read(int key) async {  
    final db = await openDb();  
    final txn = db.transaction(storeName, 'readonly');  
    final store = txn.objectStore(storeName);  
    final result = await store.getObject(key);  
    await txn.completed;  
    if (result != null && result is Map) {  
        return Map<String, dynamic>.from(result as Map);  
    }  
    return null;  
}
```

```
function readData() {  
    const transaction = db.transaction([STORE_NAME], 'readonly');  
    const objectStore = transaction.objectStore(STORE_NAME);  
    const request = objectStore.get(ID);  
    ...  
}
```



# ReadAll

```
Future<List<Map<String, dynamic>>> readAll(Database db) async {  
  var txn = db.transaction(storeName, idbModeReadOnly);  
  var store = txn.objectStore(storeName);  
  var items = await store.getAll();  
  await txn.completed;  
  return List<Map<String, dynamic>>.from(items);  
}
```

```
function readAll() {  
  const transaction = db.transaction([STORE_NAME], 'readonly');  
  const objectStore = transaction.objectStore(STORE_NAME);  
  // Get all data and all keys simultaneously  
  const dataRequest = objectStore.getAll();  
  const keysRequest = objectStore.getAllKeys();  
}
```

# UPDATE

```
Future<void> update(int key, Map<String, dynamic> data) async {  
    final db = await openDb();  
    final txn = db.transaction(storeName, 'readwrite');  
    final store = txn.objectStore(storeName);  
    await store.put(data, key);  
    await txn.completed;  
}
```

```
function updateData() {  
    ...  
    // put() updates or creates  
    const request = objectStore.put(updatedData, updateId);  
    request.onsuccess = function (event) { ... };  
    request.onerror = function (event) { ... };  
}
```

# DELETE

```
Future<void> deleteRecord(int key) async {  
    final db = await openDb();  
    final txn = db.transaction(storeName, 'readwrite');  
    final store = txn.objectStore(storeName);  
    await store.delete(key);  
    await txn.completed;  
}
```

```
function deleteData() {  
    ...  
    const transaction = db.transaction([STORE_NAME], 'readwrite');  
    const objectStore = transaction.objectStore(STORE_NAME);  
    const request = objectStore.delete(deleteId);  
    request.onsuccess = function (event) { ... };  
    request.onerror = function (event) { ... };  
}
```

# foobar2 project

---

- In this project we adopt a modular directory system: create separate folders for different purposes.
  - lib for Dart code
  - web for HTML/UI code
- We introduce webdev for building and application server.

# Web Application Directories

---

*For Dart web application, we use this project structure*

```
├── doc
│   └── README.md
├── lib
│   └── foo_crud.dart
├── build.yaml
├── pubspec.yaml
├── run.sh
└── web
    ├── index.html
    └── main.dart
```

## *lib directory*

- The `lib/` directory maps to `package:foobar/` (based on your `pubspec.yaml`).
  - For example, `lib/foo_crud.dart` is imported as `package:foobar/foo_crud.dart`.
- Store all Dart source files here.

```
name: foobar
```

## *web directory*

- The `web/` directory contains all the UI related files.
  - `index.html` for the main HTML page.
  - `main.dart` for the main Dart application.

```
import 'package:foobar/foo_crud.dart';  
...  
void main() { ... }
```

# webdev

---

*webdev is used for building and web server*

1. **Development Server:** Hot reload, live reloading
2. **Build Optimization:** Automatic dependency management
3. **Module System:** Better handling of package imports
4. **Development Tools:** Debugging support, source maps
5. **Asset Management:** Handles static files automatically



## *dart compile js Limitations:*

- **Manual process:** No built-in development server
- **No hot reload:** Must manually refresh browser
- **Complex dependencies:** Manual handling of package imports
- **Limited tooling:** Less integrated development experience

## *pubspec.yaml*

```
dev_dependencies:  
  build_runner: ^2.4.0          # Build system foundation  
  build_web_compilers: ^4.0.4 # Dart-to-JS compilation
```

- **build\_runner**: Coordinates build process
- **build\_web\_compilers**: Compiles Dart to optimized JavaScript

## *build.yaml*

- The `build.yaml` file is used to customize the Dart build system for web applications, especially when working with the webdev tool.

```
targets:  
  $default:  
    builders:  
      build_web_compilers:entrypoint:  
        generate_for:  
          - web/**/*.dart  
        options:  
          compiler: dart2js
```

## *Changing Options*

- Change `compiler: dart2js` to `compiler: dartdevc` for faster, incremental local builds during development.
- Use "webdev build --output web:build\_output" to change the output directory to `build_output`.

# Development Workflow

---

*# 1. Install dependencies*

```
dart pub get
```

*# 2. Start development server*

```
dart pub global run webdev serve
```

*# 3. Open browser to localhost:8080*

*# 4. Edit code – see changes immediately*

*# 5. Build for production*

```
dart pub global run webdev build
```

Or, just run 'webdev build' or  
'webdev serve'.

# Databases

Feature	PocketBase	Firebase	SQLite	IndexedDB
Type	Server + SQLite	Cloud NoSQL	File-based SQL	Browser NoSQL
Location	Self-hosted	Google Cloud	Local file	Browser storage
Real-time	✓ Built-in	✓ Built-in	✗ None	✗ None
Authentication	✓ Built-in	✓ Complete	✗ Manual	✗ Manual
Scalability	⚠ Manual	✓ Automatic	✗ Single user	✗ Single user
Queries	✓ REST API	✓ Rich NoSQL	✓ Full SQL	✗ Key-value
Offline	✗ Network only	✓ Smart sync	✓ Always	✓ Always
Cost	🆓 Free hosting	💰 Pay-per-use	🆓 Free	🆓 Free

# Decision Framework

---

- **Choose PocketBase** for: Self-hosted real-time apps, educational projects, MVPs, data control`

**Choose IndexedDB** for: Browser-only applications, offline-first web apps, client-side caching

- **Choose SQLite** for: Single-user apps, offline-first, embedded applications
- **Choose Firebase** for: Global scale, automatic scaling, rapid development without hosting



# IndexedDB Limitations

---

## Query Limitations:

- No complex joins or relationships
- Limited search capabilities (no full-text search)
- No built-in aggregation functions
- Index-based queries only

## Browser Limitations:

- Same-origin policy restrictions
- Storage quotas vary by browser
- Data can be cleared by user/browser
- No server-side processing

## Development Complexity:

- Asynchronous API can be complex
- Transaction management required
- Browser compatibility considerations