

Day 7 — Async in Practice: `async/await`, Fetch, Abort

This document explains all Day 7 concepts in detail, including:

- `async/await` with Fetch
 - try/catch/finally patterns
 - `AbortController` (cancellation & timeouts)
 - Retry and Exponential Backoff strategies
 - Practical examples
-

1. `async/await` + Fetch

`fetch()` returns a **Promise**, which resolves to a `Response` object. Using `async/await` makes asynchronous code look synchronous.

Basic Example

```
async function loadUsers() {
  try {
    const res = await fetch("https://api.example.com/users");
    const data = await res.json();
    console.log(data);
  } catch (err) {
    console.error("Failed to load users:", err);
  }
}
```

2. try / catch / finally

- **try**: code that might throw an error
- **catch**: handles the error
- **finally**: always runs (cleanup, stop loading UI, etc.)

Example

```
async function getUser() {
  try {
    console.log("Fetching user...");
  }
```

```

    const res = await fetch("/api/user");
    const data = await res.json();
    console.log("User:", data);
} catch (err) {
    console.error("Error:", err);
} finally {
    console.log("Done (success or fail)");
}
}

```

3. AbortController (Cancel + Timeout)

`AbortController` allows you to cancel a fetch request. Useful when:

- A request takes too long
- User navigates away
- You want to manually stop a request

Basic Cancellation

```

const controller = new AbortController();

fetch("https://api.example.com/data", { signal: controller.signal })
    .then(res => res.json())
    .then(data => console.log(data))
    .catch(err => {
        if (err.name === "AbortError") {
            console.log("Request cancelled");
        }
    });
}

// Cancel later
controller.abort();

```

Timeout Using AbortController

```

async function fetchWithTimeout(url, { timeout = 5000 } = {}) {
    const controller = new AbortController();
    const id = setTimeout(() => controller.abort(), timeout);

    try {
        const res = await fetch(url, { signal: controller.signal });
    }
}

```

```

        return res;
    } catch (err) {
        if (err.name === "AbortError") {
            throw new Error("Request timed out");
        }
        throw err;
    } finally {
        clearTimeout(id);
    }
}

```

4. Retry Logic + Backoff

Why Retry?

- Network instability
- Temporary API errors (500, 503, 502)
- Improve robustness of your app

Simple Retry

```

async function fetchWithRetry(url, { retries = 3, delay = 500 } = {}) {
    let attempt = 0;

    while (attempt <= retries) {
        try {
            const res = await fetch(url);
            if (!res.ok) throw new Error(`HTTP ${res.status}`);
            return res;
        } catch (err) {
            if (attempt === retries) throw err;

            console.warn(`Attempt ${attempt + 1} failed, retrying...`);
            await new Promise(resolve => setTimeout(resolve, delay));
            attempt++;
        }
    }
}

```

Exponential Backoff

Backoff increases delay after each failure:

- 500ms
- 1000ms
- 2000ms

```
async function fetchWithBackoff(url, { retries = 3, baseDelay = 500 } = {}) {
  let attempt = 0;

  while (attempt <= retries) {
    try {
      const res = await fetch(url);
      if (!res.ok) throw new Error(`HTTP ${res.status}`);
      return res;
    } catch (err) {
      if (attempt === retries) throw err;

      const delay = baseDelay * 2 ** attempt;
      console.warn(`Attempt ${attempt + 1} failed, retrying in ${delay}ms...`);

      await new Promise(resolve => setTimeout(resolve, delay));
      attempt++;
    }
  }
}
```

5. Full Practical Flow

Combining:

- Timeout
- Retry
- async/await
- Error handling

```
async function loadData() {
  try {
    const res = await fetchWithBackoff("/api/data", {
      retries: 3,
      baseDelay: 500,
    });
  }
}
```

```
const data = await res.json();
console.log("Data received:", data);
} catch (err) {
  console.error("Final failure:", err);
} finally {
  console.log("Request finished");
}
```

6. Summary

- `async/await` makes async code readable and simple.
- `try/catch/finally` helps with robust error handling.
- `AbortController` provides cancellation + manual timeouts.
- Retry patterns improve reliability.
- Backoff prevents spamming the server.