Let's use `await` at the top level in a module `db-connection.js`. This makes sense since the connection to the DB needs to be established before the module is usable.

db-connection.js

```
_{{db-connection.js}}_
```

But `db-connection.js` is no longer a normal module now. It's an **async module** now. Async modules have a different evaluation semantics. While normal modules evaluate synchronously, async modules evaluate asynchronously.

Async modules can still be imported with a normal `import`. But importing an async module makes the importing module also an async module.

The `import`s still hoist and are evaluated in parallel.

Tree shaking still works as usual. Here the `close` function is never used and will be removed from the output bundle in production mode.

UserApi.js

```
_{{UserApi.js}}_
```

Now it looks like that this pattern will continue and will infect all using modules as async modules.

Yes, this is kind of true and makes sense. All these modules have their evaluation semantics changed to be async.

But you as a developer don't want this. You want to break the chain at a point in your module graph where it makes sense. Luckily there is a nice way to break the chain.

You can use `import("./UserApi.js")` to import the module instead of `import`. As this returns a Promise it can be awaited to wait for module evaluation (including top-level-awaits) and handle failures.

Handling failures is an important point here. When using top-level-await there are more ways that a module evaluation can fail now. In this example connecting to the DB may fail.

Actions.js

```
_{{Actions.js}}_
```

As `Actions.js` doesn't use any top-level-await nor `import`s an async module directly so it's not an async module.

example.js

```
_{{example.js}}_
```

As a guideline, you should prevent your application entry point to become an async module when compiling for web targets. Doing async actions at application bootstrap will delay your application startup and may be negative for UX. Use `import()` to do async action on-demand or in the background and use spinners or other indicators to inform the user about background actions.

When compiling for other targets like node.js, electron or WebWorkers, it may be fine that your entry point becomes an async module.

dist/output.js

```
_{{dist/output.js}}_
```

dist/497.output.js

```
_{{dist/497.output.js}}_
```

in production mode:

```
_{{production:dist/497.output.js}}_
```

Info

Unoptimized

```
_{{stdout}}_
```

Production mode

```
_{{production:stdout}}_
```