Exercise 2: Error Handling

30 Points Possible

2/22/2026

Attempt 1

In Progress
NEXT UP: Submit Assignment

Add Comment

Unlimited Attempts Allowed
2/6/2026 to 2/25/2026

Details

In this assignment, you will implement an API layer for a (simplified) URL shortener application to practice error handling utilizing multiple error models.

Setup/Submission

- Setup: **UrlApi.kt (https://ufl.instructure.com/courses/555493/files/104689719?wrap=1)** **(https://ufl.instructure.com/courses/555493/files/104689719/download?download_frd=1)** contains the provided code. You can use an online editor such as the **Kotlin playground (https://play.kotlinlang.org/)**, or create a minimal Kotlin project in IntelliJ yourself (recommended for better IDE experience).
- Submission: You will submit your updated `UrlApi.kt` file. You should only implement the methods in `object UrlApi`; the surrounding code structure and logic should be left unchanged (though you may add additional tests as desired).

HTTP Overview

HTTP is Hypertext Transfer Protocol, a standard protocol used for server APIs. REST (Representation State Transfer) and CRUD (Create-Read-Update-Delete) are related terms for designing HTTP APIs. Each request contains a `METHOD`, a `/route`, and optionally a `<body>` with additional data if needed (typically JSON in practice). Our UrlApi will have the following endpoints:

- `POST /urls <body>`: Create a `urls` resource.
- `GET /urls/:id`: Reads a `urls` resource with id `:id`.
- `PATCH /urls/:id <body>`: **partially (https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods/PATCH)** Updates a `urls` resource with id `:id`.
- `DELETE /urls/:id`: Deletes a `urls` resource with id `:id` (or at least attempts to).

HTTP responses include a status code, which are standardized to represent different success/failure conditions. The "correct" status code to use is often a subject of debate and can impact compatibility with other systems (especially with caching), but the ones we will use in this assignment are "reasonable" for our use cases and listed below.

- `2xx`: Success responses (`200` is generic, `201` is creation).
- `4xx`: Client issues (`400` is bad request, `404` is not found, `409` is conflict).
- `5xx`: Server issues (`500` is generic).

UrlApi Endpoints

This class defines methods implementing the API using a variety of response/error architectures, the main pieces of which are below:

- `HttpResponse`/`HttpError` represents a response/error with an associated HTTP status code. `HttpError`, being an `Exception`, can be thrown.
- `Result<...>` represents a success/failure response as a value support compile-time checks instead of throwing an unchecked exception. See the **Kotlin docs for Result (https://kotlinlang.org/api/core/kotlin-stdlib/kotlin/-result/)** for relevant utility methods that may be helpful here.
- Unexpected failures (e.g broken preconditions via `require`) represent an unrecoverable programming error (bug).

- **For the purposes of our assignment, we will intentionally not validate `require` preconditions to allow simulating these failures.** For example, `POST /urls {url:''}` will intentionally not validate that `url` is non-empty, thus resulting in an "unexpected" exception from `UrlModel.insert` that is rethrown by `create`.
- In a "real" application, we would validate these types of preconditions along with the rest of the body and return a `400`. Additionally, our API would have middleware to the catch these unexpected exceptions and log/alert before translating it to a HTTP-compatible `500` rather than crashing the server itself.

The `UrlModel` class contains methods for interacting with our "database", each using their own approach to error handling as well. Since these are internal methods, they won't use `HttpResponse`/`HttpError`. Therefore, it is your responsibility to determine how different error conditions are handled and translate the logic appropriately using the provided implementation (which you may assume is properly defined). When in doubt, refer to the provided test cases for the expected behavior - these aren't guaranteed to be fully exhaustive but do cover the majority of behavior.

Endpoint	Description	Examples
`POST` `/urls` `<body>`	Creates a URL. This function uses exceptions for error handling, as does `UrlModel.insert`. • The `body` must contain only a required `url` and optional `alias`. The alias, if present, cannot *conflict* (hint, hint) with an existing alias.	• `POST /urls {url:'url'}`: `201` • `POST /urls {url:'url',alias:'alias'}`: `201` • `POST /urls {invalid:'invalid'}`: `400` • `POST /urls {url:'error'}`: `500` • `POST /urls {url:''}`: `Exception`
`GET` `/urls/:key`	Reads a URL by id or alias. This function also uses exceptions for error handling, as does `UrlModel.select(Int)`, but `UrlModel.select(String)` returns a `Nullable` type. • The `key` is either a URL `id` (if a valid integer) or else a URL `alias`. Hint: Review Kotlin's docs on **Null Safety (https://kotlinlang.org/docs/null-safety.html#safe-call-operator)**, most notably the Elvis operator `?:`. Additionally, note the existence of `String.toIntOrNull(): Int?` instead of just `String.toInt(): Int` for extracting the `id`.	• `GET /urls/1`: `200` or `404` • `GET /urls/alias`: `200` or `404` • `GET /urls/error`: `500` • `GET /urls/-2`: `Exception`
`PATCH` `/urls/:id` `<body>`	Updates a URL alias by id. This function now uses Kotlin's `Result` type for error handling, as does `UrlModel.update`. • The `id` must be a valid integer. • The `body` must contain only a required `alias`. Note: Kotlin's `Result` type leaves a fair bit to be desired compared to other languages (especially for handling specific failures), so just keep that in mind. Review the **Result docs (https://kotlinlang.org/api/core/kotlin-stdlib/kotlin/-result/)** for relevant utility methods that may be helpful here.	• `PATCH /urls/1 {alias:'new-alias'}`: `200` or `404` • `PATCH /urls/1 {invalid:'invalid'}`: `400` • `PATCH /urls/1 {alias:'error'}`: `500` • `PATCH /urls/1 {alias:''}`: `Exception`
`DELETE` `/urls/:id`	Deletes a URL by id. This function now uses just a `Boolean` value to indicate client success/failure conditions, exceptions for server failures, and having `UrlModel.delete` use Kotlin's `Result` type. • The `id` must be a valid integer. Note: Returning only a `Boolean success` is bad practice since it's not using proper HTTP status code, but this demonstrates how errors from different sources can be combined/transformed and allows for chaining operations.	• `DELETE /urls/1`: `true` or `false` • `DELETE /urls/invalid`: `false` • `DELETE /urls/-1`: `IOException` • `DELETE /urls/-2`: `Exception`

Changelog

2/20	• **Fixed Bug With parseBody Testing Function** ◦ Added checking for an empty body in requests, which caused breaking errors.

Choose a submission type

Upload Google Drive More

Choose a file to upload

File permitted: KT

or

Canvas Files

(https://ufl.instructure.com/courses/555493/modules/items/12536184)

Assignment

(https://ufl.instructure.com/courses/555493/modules/items/1253