Raskel Programming Language Sample - TaskFlow: A Task Scheduler API

Overview

This is a sample project using Raskel that shows some advanced programming ideas, API building, and clean coding practices. It simulates a task scheduler API with a structure similar to REST and includes concepts like immutability, concurrency, and error handling. The project illustrates how to create scalable and efficient systems while keeping the code neat and straightforward.

Code Sample

The following code demonstrates the TaskFlow project in Raskel. It includes task management features like creating, updating, and deleting tasks, along with simulating a REST API.

```
-- TASKFLOW: A Raskel-based Task Scheduler and API Framework
-- Description: Demonstrates concurrency, functional paradigms, RESTful simulation, error handling, and IO in Raskel

module TaskFlow exposing (main)

import Std.IO exposing (println, readFile, writeFile)
import Std.JSON exposing (encode, decode, Json)
import Std.List exposing (map, filter, foldl, head, tail)
import Std.Time exposing (now, sleep)
import Std.Result exposing (Result(..), mapError, withDefault)
import Std.Maybe exposing (Maybe(..))
import Std.String exposing (split, join, contains, toUpper)
import Std.Http exposing (Request, Response, Server, get, post)
import Std.Log exposing (logInfo, logError)
import Std.UUID exposing (uuid)

-- Task Type Definition
type alias Task =
    { id : String
    , title : String
    , completed : Bool
    , createdAt : String
    }

-- App State
type alias State =
    { tasks : List Task }

-- Initial state
initState : State
initState =
    { tasks = [] }

-- Generate new task with UUID and timestamp
createTask : String -> IO Task
createTask title =
    do
        id <- uuid
        timestamp <- now
        return
            { id = id
            , title = title
            , completed = False
            , createdAt = timestamp
            }

-- Toggle completion of a task
toggleTask : String -> State -> State
toggleTask taskId state =
    let
        updatedTasks =
            map
                (        -> if t.id == taskId then { t | completed = not t.completed } else t)
                state.tasks
    in
    { state | tasks = updatedTasks }

-- Delete a task
deleteTask : String -> State -> State
deleteTask taskId state =
    { state | tasks = filter (          -> t.id /= taskId) state.tasks }
```

```
-- Get all completed tasks
getCompletedTasks : State -> List Task
getCompletedTasks state =
    filter (          -> t.completed) state.tasks

-- Save state to file
saveState : State -> IO ()
saveState state =
    writeFile "taskflow_data.json" (encode state)

-- Load state from file
loadState : IO (Result String State)
loadState =
    readFile "taskflow_data.json"
        |> mapError (\_ -> "Failed to load state")
        |> map decode

-- REST API Simulated Routes

-- GET /tasks
getAllTasks : Request -> State -> Response
getAllTasks _ state =
    { status = 200, body = encode state.tasks }

-- POST /task
addNewTask : Request -> State -> IO Response
addNewTask req state =
    case decode req.body of
        Ok data ->
            case data["title"] of
                Just titleStr ->
                    do
                        newTask <- createTask titleStr
                        let updated = { state | tasks = newTask :: state.tasks }
                        saveState updated
                        return { status = 201, body = encode newTask }

                Nothing ->
                    return { status = 400, body = "Missing 'title' in request body" }
        Err _ ->
            return { status = 400, body = "Invalid JSON" }

-- POST /task/toggle/:id
toggleTaskHandler : Request -> State -> IO Response
toggleTaskHandler req state =
    let
        segments = split "/" req.path
        taskId = case tail segments of
            Just (_ :: _ :: id :: _) -> id
            _ -> ""
        updated = toggleTask taskId state
    in
    do
        saveState updated
        return { status = 200, body = encode (getCompletedTasks updated) }

-- DELETE /task/:id
deleteTaskHandler : Request -> State -> IO Response
deleteTaskHandler req state =
    let
        segments = split "/" req.path
        taskId = case tail segments of
            Just (_ :: id :: _) -> id
            _ -> ""
        updated = deleteTask taskId state
    in
    do
        saveState updated
        return { status = 204, body = "" }

-- Simulated Server Setup
routes : List (Request -> State -> IO Response)
routes =
    [ get "/tasks" getAllTasks
    , post "/task" addNewTask
    , post "/task/toggle/:id" toggleTaskHandler
    , post "/task/delete/:id" deleteTaskHandler
    ]
```

```
-- Server loop simulation
main : IO ()
main =
    do
        logInfo "Loading saved state..."
        loaded <- loadState
        let state = withDefault initState loaded

        logInfo "Starting simulated server..."
        server <- Server.init 3000 routes state
        server.run()
```