Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного автономного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИУК «Информатика и управление»**

КАФЕДРА **ИУК5 «Системы обработки информации»**

ЛАБОРАТОРНАЯ РАБОТА №4

«Практика применения Haskell»

ДИСЦИПЛИНА: «Перспективные технологии разработки программных средств»

Выполнил: студент гр. ИУК4-31М _____ (Сафронов Н.С,)
(подпись) (Ф.И.О.)

Проверил: _____ (Кириллов В.Ю.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2025

Цель:

Изучение практических аспектов написания программ на Haskell.

Задачи:

- Изучение организации кода, сборки и тестирования проектов.
- Навыки обработки ошибок.
- Реализация методов работы с http-запросами, разбором json и базовых операций с СУБД.
- Написание отчета о работе.

Результаты выполнения работы

Задача 39.1. Реализуйте функцию buildRequestNOSSL, которая работает аналогично buildRequest, но не поддерживает SSL.

Задача 39.2. Улучшите вывод программы при возникновении ошибок. Функция getResponseStatus возвращает значение, содержащее два компонента: statusCode и statusMessage. Исправьте main так, чтобы при получении кода, отличного от 200, выводилось подходящее сообщение об ошибке.

Листинг программы

```
module Main where

import qualified Data.ByteString as B
import qualified Data.ByteString.Char8 as BC
import qualified Data.ByteString.Lazy as L
import qualified Data.ByteString.Lazy.Char8 as LC
import Network.HTTP.Simple
import Network.HTTP.Types.Status (statusCode, statusMessage)

noaaHost :: BC.ByteString
noaaHost = "www.ncei.noaa.gov"

apiPath :: BC.ByteString
apiPath = "/access/services/search/v1/datasets"

buildRequest :: BC.ByteString -> BC.ByteString -> BC.ByteString ->
Request
buildRequest host method path =
    setRequestMethod method
    $ setRequestHost host
    $ setRequestPath path
    $ setRequestSecure True
    $ setRequestPort 443
    $ addRequestHeader "Accept-Encoding" (BC.pack "application/json")
    $ defaultRequest
```

```
      -- Задача 39.1
      buildRequestNOSSL :: BC.ByteString -> BC.ByteString -> BC.ByteString ->
Request
      buildRequestNOSSL host method path =
          setRequestMethod method
          $ setRequestHost host
          $ setRequestPath path
          $ setRequestSecure False
          $ setRequestPort 80
          $ addRequestHeader "Accept-Encoding" (BC.pack "application/json")
          $ defaultRequest

      -- Задача 39.2
      main :: IO ()
      main = do
          let request = buildRequestNOSSL noaaHost "GET" apiPath
          response <- httpLBS request
          let status = getResponseStatus response
              code = statusCode status
              message = statusMessage status
          if code == 200
              then do
                  putStrLn "Succesfully saved data to data.json"
                  L.writeFile "data.json" (getResponseBody response)
              else
                  putStrLn $ "HTTP Error " ++ show code ++ ": " ++ BC.unpack
message
```

Задача 40.1. Сделайте NOAAResponse экземпляром ToJSON. Это потребует написания экземпляров ToJSON для всех вложенных типов.

Листинг программы

```
module Main where

import Data.Aeson
import qualified Data.ByteString.Lazy.Char8 as L
import GHC.Generics
import Control.Monad (forM_)
import Data.List (intercalate)
import qualified Data.Text as T

data NOAASearchResponse = NOAASearchResponse
  { totalCount :: Int
  , results    :: [Dataset]
  } deriving (Show, Generic)

instance FromJSON NOAASearchResponse where
  parseJSON (Object v) =
    NOAASearchResponse
      <$> v .: "totalCount"
      <*> v .: "results"
  parseJSON _ = fail "Expected object"
```

```
data Dataset = Dataset
  { datasetId   :: T.Text
  , name        :: T.Text
  , description :: T.Text
  , startDate   :: T.Text
  , endDate     :: T.Text
  , available   :: Bool
  , keywords    :: Maybe [Keyword]
  , links       :: Maybe AccessLinks
  } deriving (Show, Generic)

instance FromJSON Dataset where
  parseJSON (Object v) =
    Dataset
      <$> v .: "id"
      <*> v .: "name"
      <*> v .: "description"
      <*> v .: "startDate"
      <*> v .: "endDate"
      <*> v .: "available"
      <*> v .:? "keywords"
      <*> v .:? "links"
  parseJSON _ = fail "Expected object"

data Keyword = Keyword
  { keywordName :: T.Text
  , keywordID   :: T.Text
  } deriving (Show, Generic)

instance FromJSON Keyword where
  parseJSON (Object v) =
    Keyword
      <$> v .: "name"
      <*> v .: "id"
  parseJSON _ = fail "Expected Keyword object"

data AccessLinks = AccessLinks
  { other :: Maybe [Link]
  , access :: Maybe [Link]
  , documentation :: Maybe [Link]
  } deriving (Show, Generic)

instance FromJSON AccessLinks where
  parseJSON (Object v) =
    AccessLinks
      <$> v .:? "other"
      <*> v .:? "access"
      <*> v .:? "documentation"
  parseJSON _ = fail "Expected object for links"

data Link = Link
  { linkName :: T.Text
  , linkType :: T.Text
  , linkURL  :: T.Text
  } deriving (Show, Generic)
```

```
instance FromJSON Link where
  parseJSON (Object v) =
    Link
      <$> v .: "name"
      <*> v .: "type"
      <*> v .: "url"
  parseJSON _ = fail "Expected object"

printResults :: Maybe [Dataset] -> IO ()
printResults Nothing = putStrLn "Error: failed to load datasets."
printResults (Just datasets) = do
  putStrLn $ "Found datasets: " ++ show (length datasets)
  forM_ datasets $ \ds -> do
    putStrLn $ "* " ++ T.unpack (name ds)
    putStrLn $ "  ID: " ++ T.unpack (datasetId ds)
    putStrLn $ "  Period: " ++ T.unpack (startDate ds) ++ " - " ++ T.unpack
(endDate ds)
    putStrLn $ "  Available: " ++ if available ds then "Yes" else "No"
    putStrLn $ "  Description: " ++ T.unpack (description ds)

    case keywords ds of
      Nothing -> return ()
      Just kws -> do
        putStr "  Keywords: "
        let kwStrs = map (\kw -> T.unpack (keywordName kw)) kws
        putStrLn (intercalate ", " kwStrs)

    case links ds of
      Nothing -> return ()
      Just al -> case access al of
        Nothing -> return ()
        Just linkList -> do
          putStrLn "  Access URLs:"
          forM_ linkList $ \lnk -> do
            putStrLn $ "    - " ++ T.unpack (linkURL lnk)

    putStrLn ""

-- Задача 40.1
instance ToJSON NOAASearchResponse where
  toJSON = genericToJSON defaultOptions
    { fieldLabelModifier = camelTo2 '_' }

instance ToJSON Dataset where
  toJSON = genericToJSON defaultOptions
    { fieldLabelModifier = camelTo2 '_' }

instance ToJSON Keyword where
  toJSON = genericToJSON defaultOptions
    { fieldLabelModifier = \s ->
        case s of
          "keywordName" -> "name"
          "keywordID"   -> "id"
          _             -> s
```

```
    }

instance ToJSON AccessLinks where
  toJSON = genericToJSON defaultOptions
    { fieldLabelModifier = camelTo2 '_' }


instance ToJSON Link where
  toJSON = genericToJSON defaultOptions
    { fieldLabelModifier = \s ->
        case s of
          "linkName" -> "name"
          "linkType" -> "type"
          "linkURL"  -> "url"
          _          -> s
    }

main :: IO ()
main = do
  result <- eitherDecodeFileStrict "data.json" :: IO (Either String
NOAASearchResponse)
  case result of
    Left err -> putStrLn $ "Ошибка парсинга JSON: " ++ err
    Right response -> do
        printResults (Just (results response))
        L.writeFile "reformatted.json" (encode response)
```

```
PS E:\Dev\bmstu-magistracy> E:\Dev\bmstu-magistracy\term_3\haskell\lab_4\src\40\json-lesson\.stack-work\install\eb475849\bin\json-lesson-exe.exe
Found datasets: 10
* Global Summary of the Month (GSOM), Version 1
  ID: global-summary-of-the-month
  Period: 1763-01-01 - 2025-12-06
  Available: Yes
  Description: This Global Summaries dataset, known as GSOM for Monthly, contains a monthly resolution of meteorological elements from 1763 to present with
updates applied weekly. The major parameters are: monthly mean maximum, mean minimum and mean temperatures; monthly total precipitation and snowfall; depart
ure from normal of the mean temperature and total precipitation; monthly heating and cooling degree days; number of days that temperatures and precipitation
 are above or below certain thresholds; extreme daily temperature and precipitation amounts; number of days with fog; and number of days with thunderstorms.
 The primary input data source is the Global Historical Climatology Network - Daily (GHCN-Daily) dataset. The Global Summaries datasets also include a yearl
y resolution of meteorological elements in the GSOY (for Yearly) dataset. See associated resources for more information. These datasets are not to be confus
ed with "GHCN-Monthly", "Annual Summaries" or "NCDC Summary of the Month". There are unique elements that are produced globally within the GSOM and GSOY dat
a files. There are also bias corrected temperature data in GHCN-Monthly, which are not available in GSOM and GSOY. The GSOM and GSOY datasets replace the le
gacy U.S. COOP Summaries (DSI-3220), and have been expanded to include non-U.S. (global) stations. U.S. COOP Summaries (DSI-3220) only includes National Wea
ther Service (NWS) COOP Published, or "Published in CD", sites.
  Keywords: PRECIPITATION AMOUNT, RAIN, SNOW, SURFACE PRESSURE, MAXIMUM/MINIMUM TEMPERATURE, HEATING DEGREE DAYS, COOLING DEGREE DAYS, SURFACE WINDS (ATMOSP
HERIC WINDS), FOG (TROPOSPHERIC/LOW LEVEL CLOUDS (OBSERVED/ANALYZED)), RAIN STORMS
  Access URLs:
    - https://www.ncei.noaa.gov/access/search/data-search/global-summary-of-the-month
    - https://www.ncei.noaa.gov/maps/monthly/
    - https://www.ncei.noaa.gov/data/global-summary-of-the-month/
```

Рисунок 1 – Результат вывода программы 40

Рисунок 2 – Пример данных в сохранённом файле (camelCase тегов источника переведён в snake_case)

Задача 41.1. Определите IO-действие addTool, которое добавляет в базу данных новые инструменты аналогично addUser.

Задача 41.2. Добавьте команду addtool, которая запрашивает у пользователя необходимую информацию, а затем добавляет в базу данных новый инструмент, используя функцию addTool из предыдущего упражнения.

Листинг программы

```
module Main where

import Control.Applicative
import Data.Time
import Data.Time.Calendar (dayOfWeek)
import Database.PostgreSQL.Simple
import Database.PostgreSQL.Simple.FromRow
import System.IO

connInfo :: ConnectInfo
connInfo = ConnectInfo
  { connectHost     = "localhost"
  , connectPort     = 5432
  , connectUser     = "myuser"
  , connectPassword = "mypassword"
  , connectDatabase = "mydatabase"
  }

data Tool = Tool
  { toolId        :: Int
  , name          :: String
  , description   :: String
  , lastReturned  :: Maybe Day
  , timesBorrowed :: Int
```

```
  }

instance Show Tool where
  show tool = mconcat
    [ show (toolId tool)
    , ".) "
    , name tool
    , "\n description: "
    , description tool
    , "\n last returned: "
    , show (lastReturned tool)
    , "\n times borrowed: "
    , show (timesBorrowed tool)
    ]

data User = User
  { userId   :: Int
  , userName :: String
  }

instance Show User where
  show user = mconcat [show (userId user), ".) ", userName user]

withConn :: (Connection -> IO ()) -> IO ()
withConn action = do
  conn <- connect connInfo
  action conn
  close conn

addUser :: String -> IO ()
addUser userName = withConn $ \conn -> do
  execute conn "INSERT INTO users (username) VALUES (?)" (Only userName)
  putStrLn "User added."

-- Задание 41.1
addTool :: String -> String -> IO ()
addTool name description = withConn $ \conn -> do
  execute conn
    "INSERT INTO tools (name, description, timesBorrowed) VALUES (?, ?, ?)"
    (name, description, 0 :: Int)
  putStrLn "Tool added."

checkout :: Int -> Int -> IO ()
checkout userId toolId = withConn $ \conn -> do
  execute conn
    "INSERT INTO checkedout (user_id, tool_id) VALUES (?, ?)"
    (userId, toolId)
  putStrLn "Checked out."

instance FromRow User where
  fromRow = User <$> field <*> field

instance FromRow Tool where
  fromRow = Tool <$> field <*> field <*> field <*> field <*> field
```

```
printUsers :: IO ()
printUsers = withConn $ \conn -> do
  response <- query_ conn "SELECT * FROM users" :: IO [User]
  mapM_ print response

printToolQuery :: Query -> IO ()
printToolQuery q = withConn $ \conn -> do
  response <- query_ conn q :: IO [Tool]
  mapM_ print response

printTools :: IO ()
printTools = printToolQuery "SELECT * FROM tools"

printAvailableTools :: IO ()
printAvailableTools =
  printToolQuery "SELECT * FROM tools WHERE id NOT IN (SELECT tool_id FROM
checkedout)"

printCheckedoutTools :: IO ()
printCheckedoutTools =
  printToolQuery "SELECT * FROM tools WHERE id IN (SELECT tool_id FROM
checkedout)"

selectTool :: Connection -> Int -> IO (Maybe Tool)
selectTool conn toolId = do
  response <- query conn "SELECT * FROM tools WHERE id = ?" (Only toolId) ::
IO [Tool]
  return $ case response of
    []      -> Nothing
    (x:_)   -> Just x

updateTool :: Tool -> Day -> Tool
updateTool tool date =
  tool { lastReturned = Just date, timesBorrowed = 1 + timesBorrowed tool }

updateOrWarn :: Maybe Tool -> IO ()
updateOrWarn Nothing = putStrLn "ID not found"
updateOrWarn (Just tool) = withConn $ \conn -> do
  execute conn
    "UPDATE tools SET lastReturned = ?, timesBorrowed = ? WHERE id = ?"
    (lastReturned tool, timesBorrowed tool, toolId tool)
  putStrLn "Tool updated"

updateToolTable :: Int -> IO ()
updateToolTable toolId = withConn $ \conn -> do
  tool <- selectTool conn toolId
  currentDay <- utctDay <$> getCurrentTime
  case tool of
    Nothing -> putStrLn "ID not found"
    Just t  -> do
      let updated = updateTool t currentDay
      execute conn
        "UPDATE tools SET lastReturned = ?, timesBorrowed = ? WHERE id = ?"
        (lastReturned updated, timesBorrowed updated, toolId)
      putStrLn "Tool updated"
```

```
checkin :: Int -> IO ()
checkin toolId = withConn $ \conn -> do
  execute conn "DELETE FROM checkedout WHERE tool_id = ?" (Only toolId)
  putStrLn "Checked out."

checkinAndUpdate :: Int -> IO ()
checkinAndUpdate toolId = do
  checkin toolId
  updateToolTable toolId

promptAndAddUser :: IO ()
promptAndAddUser = do
  hSetBuffering stdout NoBuffering
  putStr "User name? "
  hFlush stdout
  userName <- getLine
  addUser userName

-- Задание 41.2
promptAndAddTool :: IO ()
promptAndAddTool = do
  hSetBuffering stdout NoBuffering
  putStr "Tool name? "
  hFlush stdout
  name <- getLine
  putStr "Tool description? "
  hFlush stdout
  description <- getLine
  addTool name description

promptAndCheckout :: IO ()
promptAndCheckout = do
  hSetBuffering stdout NoBuffering
  putStr "User ID? "
  hFlush stdout
  userId <- read <$> getLine
  putStr "Tool ID? "
  hFlush stdout
  toolId <- read <$> getLine
  checkout userId toolId

promptAndCheckin :: IO ()
promptAndCheckin = do
  hSetBuffering stdout NoBuffering
  putStr "Tool ID? "
  hFlush stdout
  toolId <- read <$> getLine
  checkinAndUpdate toolId

performCommand :: String -> IO ()
performCommand "users"    = printUsers >> main
performCommand "tools"    = printTools >> main
performCommand "adduser"  = promptAndAddUser >> main
performCommand "addtool"  = promptAndAddTool >> main
```

```
performCommand "checkout" = promptAndCheckout >> main
performCommand "checkin"  = promptAndCheckin >> main
performCommand "in"       = printAvailableTools >> main
performCommand "out"      = printCheckedoutTools >> main
performCommand "quit"     = putStrLn "Bye!"
performCommand _          = putStrLn "Error: command not found" >> main

main :: IO ()
main = do
  hSetBuffering stdout NoBuffering
  putStr "Command? "
  hFlush stdout
  command <- getLine
  performCommand command
```

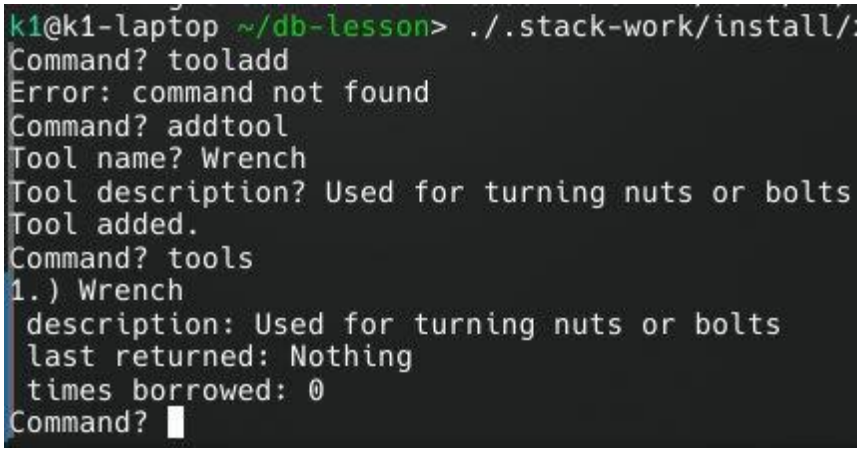

Рисунок 3 – Пример добавления инструмента в БД Postgres

Оценка времени, затраченного на выполнение

На выполнение лабораторной работы ушло в общей сложности около 8 часов.

Оценка понятности материала

Сами концепции в материале понятны, но много неактуальных моментов (связанных с package.yml у stack и совершенно другим форматом работы приведённого API).

Выводы о наиболее сложных на текущей стадии аспектах языка

- Точное согласование типов при работе с библиотеками типа aeson и http-client. Необходимость явного указания типов при декодировании JSON

(:: Maybe MyType) или использовании FromJSON/ToJSON требует внимательности. Ошибки часто проявляются только во время выполнения (в виде Nothing при парсинге), что усложняет отладку.

- Работа с внешними зависимостями и нативными библиотеками на Windows. Интеграция Haskell-кода с системными библиотеками (например, libpq для PostgreSQL) оказалась нетривиальной из-за различий между MSVC и MinGW. Это не столько сложность самого языка, сколько следствие экосистемы на Windows (пришлось для работы с Postgres в Haskell перейти на Linux).

Аналогии с другими знакомыми средствами разработки программ

- Работа с HTTP-запросами в Haskell через http-conduit напоминает использование библиотек вроде requests в Python или net/http в Go. Однако в Haskell настройка запроса строится через композицию функций (setRequestMethod, addRequestHeader и т.д.), тогда как в Python или Go чаще используется изменение объекта запроса или передача параметров в конструктор.

- Парсинг JSON с помощью aeson аналогичен использованию json-модуля в Python или encoding/json в Go, но с ключевым отличием: в Haskell требуется явное описание типа данных и реализация (или автоматическая генерация) экземпляров FromJSON/ToJSON. Это сопоставимо с использованием dataclass + dataclasses-json в Python или структур с тегами в Go, но в Haskell проверка типов происходит на этапе компиляции, что исключает целый класс ошибок времени выполнения.

Вывод: в процессе выполнения лабораторной работы изучены практические аспекты написания программ на Haskell.