Create a new Haskell Project

Application Tutorial

Yann Esposito

[2020-02-10 Mon]

Writing a Haskell application can be quite challenging. You must know about:

- setup your coding environment
  - get the right compiler
  - use libraries
  - handle your Haskell tooling, editor/IDE
- project directory structure and best practices
  - write tests
  - benchmarks
  - profiling
- Code architecture
  - encode the data structure
  - manage state and effects

This is both a manual and a tutorial. If you follow it, you should be familiar enough with Haskell to be able to write your own applications. I will focus on command line interfaces and REST APIs.

Haskell Environment Setup

My no brainer solution for it:

1. Write this `shell.nix` file and launch `nix-shell`:

```
{ nixpkgs ? import (fetchGit {
  name = "nixos-release-19.09";
  url = "https://github.com/NixOS/nixpkgs";
  # obtained via
  # git ls-remote https://github.com/nixos/nixpkgs master
  ref = "refs/heads/nixpkgs-19.09-darwin";
  rev = "d5291756487d70bc336e33512a9baf9fa1788faf";
}) { config = { allowBroken = true; }; } }:
let
  inherit (nixpkgs) pkgs;
```

```
  inherit (pkgs) haskellPackages;

  haskellDeps = ps: with ps; [
    base
    protolude
    containers
  ];

  hspkgs = haskellPackages;

  ghc = hspkgs.ghcWithPackages haskellDeps;

  nixPackages = [
    ghc
    pkgs.gdb
    hspkgs.summoner
    hspkgs.summoner-tui
    haskellPackages.cabal-install
    haskellPackages.ghcid
  ];
in
pkgs.stdenv.mkDerivation {
  name = "env";
  buildInputs = nixPackages;
  shellHook = ''
        export PS1="\n\[[hs:\033[1;32m\]\W\[\033[0m\]]> "
    '';
}
```

2. now launch `summon-tui`

3. add the following nix files: The first file to create is the one that will pin the versions of all your packages and libraries:

   ./my-app/nixpkgs.nix

   ```
   import (fetchTarball https://github.com/NixOS/nixpkgs/archive/19.09.tar.gz) {}
   ```

   The second file is the `default.nix` file:

   ./my-app/default.nix

   ```
   { nixpkgs ? import ./nixpkgs.nix
   , compiler ? "default"
   , doBenchmark ? false }:
   let
     inherit (nixpkgs) pkgs;
     name = "my-app";
     haskellPackages = pkgs.haskellPackages;
   ```

```
  variant = if doBenchmark
            then pkgs.haskell.lib.doBenchmark
            else pkgs.lib.id;
  drv = haskellPackages.callCabal2nix name ./. {};
in
{
  my_project = drv;
  shell = haskellPackages.shellFor {
    # generate hoogle doc
    withHoogle = true;
    packages = p: [drv];
    # packages dependencies (by default haskellPackages)
    buildInputs = with haskellPackages;
      [ hlint
        ghcid
        cabal-install
        cabal2nix
        hindent
        # # if you want to add some system lib like ncurses
        # # you could by writing it like:
        # pkgs.ncurses
      ];
    # nice prompt for the nix-shell
    shellHook = ''
     export PS1="\n\[[${name}:\033[1;32m\]\W\[\033[0m\]]> "
  '';
  };
}
```

Retrieve Compiler

Dependency Management

Tooling

Haskell Project directoy structure

Tests

Benchmarks

Profiling

Haskell Code Architecture

Basic: IO

Easy: The Handle Pattern

Advanced: MTL

Expert: Free Monad