github.com/Faywynnn/nvim.lua

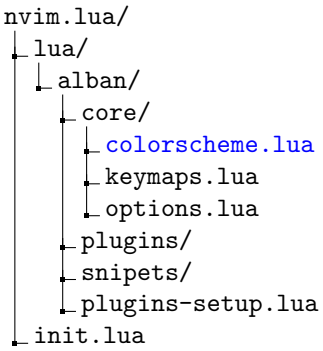
February 7, 2024

1 init.lua

nvim.lua/
- lua/
- init.lua

```lua
require("alban.plugins-setup")

require("alban.core.options")
require("alban.core.keymaps")
require("alban.core.colorscheme")

-- PLUGINS
require("alban.plugins.comment")
require("alban.plugins.nvim-tree")
require("alban.plugins.lualine")
require("alban.plugins.telescope")
require("alban.plugins.nvim-cmp")
require("alban.plugins.autopairs")
require("alban.plugins.treesitter")
require("alban.plugins.gitsigns")

require("alban.plugins.lsp.mason")
require("alban.plugins.lsp.lspsaga")
require("alban.plugins.lsp.lspconfig")
```

2 colorscheme.lua




```lua
local status, _ = pcall(vim.cmd, "colorscheme nightfly")
if not status then
  print("Colorscheme not found !")
  return
end
```

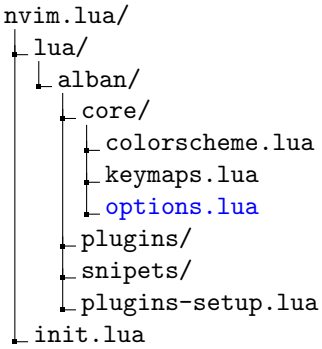
3 keymaps.lua

nvim.lua/
- lua/
  - alban/
    - core/
      - colorscheme.lua
      - keymaps.lua
      - options.lua
    - plugins/
    - snipets/
    - plugins-setup.lua
- init.lua

```lua
vim.g.mapleader = " "

local keymap = vim.keymap

---------------------
-- General Keymaps
---------------------

-- use jk to exit insert mode
keymap.set("i", "jk", "<ESC>")

-- clear search highlights
keymap.set("n", "<leader>nh", ":nohl<CR>")

-- delete single character without copying into register
keymap.set("n", "x", '"_x')

-- increment/decrement numbers
keymap.set("n", "<leader>+", "<C-a>") -- increment
keymap.set("n", "<leader>-", "<C-x>") -- decrement

-- window management
keymap.set("n", "<leader>sv", "<C-w>v")        -- split window vertically
keymap.set("n", "<leader>sh", "<C-w>s")        -- split window horizontally
keymap.set("n", "<leader>se", "<C-w>=")        -- make split windows equal width & height
keymap.set("n", "<leader>sx", ":close<CR>")    -- close current split window

keymap.set("n", "<leader>to", ":tabnew<CR>")   -- open new tab
keymap.set("n", "<leader>tx", ":tabclose<CR>") -- close current tab
keymap.set("n", "<leader>tn", ":tabn<CR>")     --  go to next tab
keymap.set("n", "<leader>tp", ":tabp<CR>")     --  go to previous tab

----------------------
-- Plugin Keybinds
----------------------

-- vim-maximizer
keymap.set("n", "<leader>sm", ":MaximizerToggle<CR>") -- toggle split window maximization

-- nvim-tree
keymap.set("n", "<leader>e", ":NvimTreeToggle<CR>") -- toggle file explorer

-- telescope
keymap.set("n", "<leader>ff", "<cmd>Telescope find_files<cr>")  -- find files within current working directory, respects .gitignore
keymap.set("n", "<leader>fs", "<cmd>Telescope live_grep<cr>")   -- find string in current working directory as you type
keymap.set("n", "<leader>fc", "<cmd>Telescope grep_string<cr>") -- find string under cursor in current working directory
keymap.set("n", "<leader>fb", "<cmd>Telescope buffers<cr>")     -- list open buffers in current neovim instance
keymap.set("n", "<leader>fh", "<cmd>Telescope help_tags<cr>")   -- list available help tags

-- telescope git commands
keymap.set("n", "<leader>gc", "<cmd>Telescope git_commits<cr>")   -- list all git commits (use <cr> to checkout) ["gc" for git commits]
keymap.set("n", "<leader>gfc", "<cmd>Telescope git_bcommits<cr>") -- list git commits for current file/buffer (use <cr> to checkout) ["gfc" for git file commits]
keymap.set("n", "<leader>gb", "<cmd>Telescope git_branches<cr>")  -- list git branches (use <cr> to checkout) ["gb" for git branch]
keymap.set("n", "<leader>gs", "<cmd>Telescope git_status<cr>")    -- list current changes per file with diff preview ["gs" for git status]

-- restart lsp server
keymap.set("n", "<leader>rs", ":LspRestart<CR>") -- mapping to restart lsp if necessary

-- luasnip
local ls = require("luasnip")
```

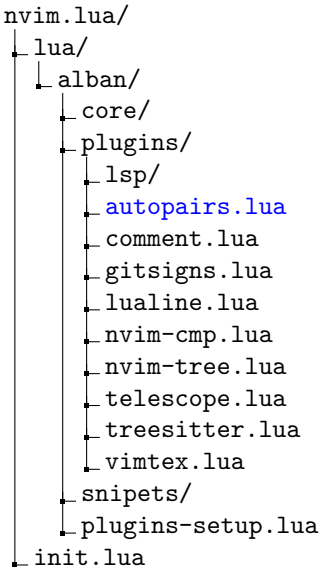
```lua
keymap.set({ "i", "s" }, "<C-j>", function() ls.jump(1) end, { silent = true })
keymap.set({ "i", "s" }, "<C-k>", function() ls.jump(-1) end, { silent = true })
```

4 options.lua



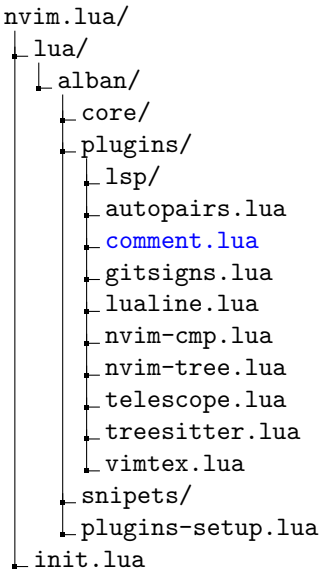

```lua
local opt = vim.opt

-- line numbers
opt.relativenumber = true
opt.number = true

-- tabs & indentation
opt.tabstop = 2
opt.shiftwidth = 2
opt.expandtab = true
opt.autoindent = true

-- line wrapping
opt.wrap = false

-- search settings
opt.ignorecase = true
opt.smartcase = true

-- cursor line
opt.cursorline = true
opt.scrolloff = 5

-- appearance
opt.termguicolors = true
opt.background = "dark"
opt.signcolumn = "yes"

-- backspace
opt.backspace = "indent,eol,start"

-- clipboard
opt.clipboard:append("unnamedplus")

-- split windows
opt.splitright = true
opt.splitbelow = true

opt.iskeyword:append("-")
```

5 autopairs.lua




```lua
-- import nvim-autopairs safely
local autopairs_setup, autopairs = pcall(require, "nvim-autopairs")
if not autopairs_setup then
  return
end

-- configure autopairs
autopairs.setup({
  check_ts = true, -- enable treesitter
  ts_config = {
    lua = { "string" }, -- don't add pairs in lua string treesitter nodes
    javascript = { "template_string" }, -- don't add pairs in javscript template_string treesitter nodes
    java = false, -- don't check treesitter on java
  },
})

-- import nvim-autopairs completion functionality safely
local cmp_autopairs_setup, cmp_autopairs = pcall(require, "nvim-autopairs.completion.cmp")
if not cmp_autopairs_setup then
  return
end

-- import nvim-cmp plugin safely (completions plugin)
local cmp_setup, cmp = pcall(require, "cmp")
if not cmp_setup then
  return
end

-- make autopairs and completion work together
cmp.event:on("confirm_done", cmp_autopairs.on_confirm_done())
```

6 comment.lua



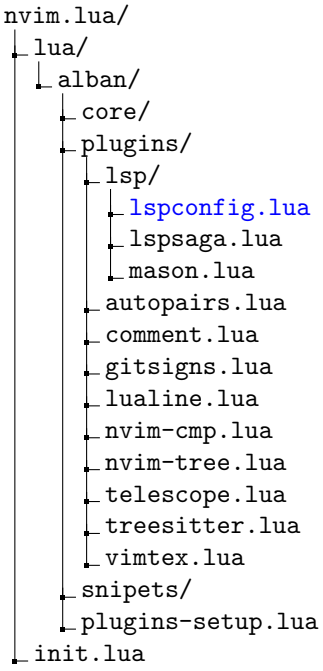

```lua
local setup, comment = pcall(require, "Comment")
if not setup then
  return
end

comment.setup()
```

7 gitsigns.lua

```
nvim.lua/
├─ lua/
│  └─ alban/
│     ├─ core/
│     ├─ plugins/
│     │  ├─ lsp/
│     │  ├─ autopairs.lua
│     │  ├─ comment.lua
│     │  ├─ gitsigns.lua
│     │  ├─ lualine.lua
│     │  ├─ nvim-cmp.lua
│     │  ├─ nvim-tree.lua
│     │  ├─ telescope.lua
│     │  ├─ treesitter.lua
│     │  └─ vimtex.lua
│     ├─ snipets/
│     └─ plugins-setup.lua
└─ init.lua
```

```lua
-- import gitsigns plugin safely
local setup, gitsigns = pcall(require, "gitsigns")
if not setup then
  return
end

-- configure/enable gitsigns
gitsigns.setup()
```

8 lspconfig.lua

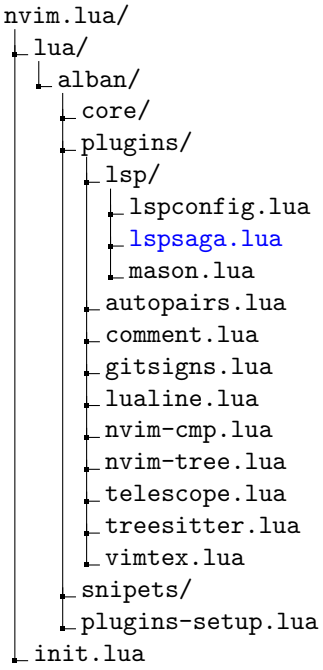



```lua
-- import lspconfig plugin safely
local lspconfig_status, lspconfig = pcall(require, "lspconfig")
if not lspconfig_status then
  return
end

-- import cmp-nvim-lsp plugin safely
local cmp_nvim_lsp_status, cmp_nvim_lsp = pcall(require, "cmp_nvim_lsp")
if not cmp_nvim_lsp_status then
  return
end

-- import typescript plugin safely
local typescript_setup, typescript = pcall(require, "typescript")
if not typescript_setup then
  return
end

local f_setup, lsp_format = pcall(require, "lsp-format")
if not f_setup then
  return
end


lsp_format.setup({})

local keymap = vim.keymap

-- enable keybinds only for when lsp server available
local on_attach = function(client, bufnr)
  lsp_format.on_attach(client)

  -- keybind options
  local opts = { noremap = true, silent = true, buffer = bufnr }

  -- set keybinds
  keymap.set("n", "gf", "<cmd>Lspsaga lsp_finder<CR>", opts)
          -- show definition, references
  keymap.set("n", "gD", "<Cmd>lua vim.lsp.buf.declaration()<CR>", opts)
          -- got to declaration
  keymap.set("n", "gd", "<cmd>Lspsaga peek_definition<CR>", opts)
          -- see definition and make edits in window
  keymap.set("n", "gi", "<cmd>lua vim.lsp.buf.implementation()<CR>", opts)
          -- go to implementation
  keymap.set("n", "<leader>ca", "<cmd>Lspsaga code_action<CR>", opts)
          -- see available code actions
  keymap.set("n", "<leader>rn", "<cmd>Lspsaga rename<CR>", opts)
          -- smart rename
  keymap.set("n", "<leader>D", "<cmd>Lspsaga show_line_diagnostics<CR>",
    opts)   -- show  diagnostics for line
  keymap.set("n", "<leader>d", "<cmd>Lspsaga show_cursor_diagnostics<CR>",
    opts) -- show diagnostics for cursor
  keymap.set("n", "[d", "<cmd>Lspsaga diagnostic_jump_prev<CR>", opts)
          -- jump to previous diagnostic in buffer
  keymap.set("n", "]d", "<cmd>Lspsaga diagnostic_jump_next<CR>", opts)
          -- jump to next diagnostic in buffer
  keymap.set("n", "K", "<cmd>Lspsaga hover_doc<CR>", opts)
          -- show documentation for what is under cursor
  keymap.set("n", "<leader>o", "<cmd>LSoutlineToggle<CR>", opts)
          -- see outline on right hand side

  -- typescript specific keymaps (e.g. rename file and update imports)
  if client.name == "tsserver" then
    keymap.set("n", "<leader>rf", ":TypescriptRenameFile<CR>")      --
    rename file and update imports
    keymap.set("n", "<leader>oi", ":TypescriptOrganizeImports<CR>") --
    organize imports (not in youtube nvim video)
    keymap.set("n", "<leader>ru", ":TypescriptRemoveUnused<CR>")    --
    remove unused variables (not in youtube nvim video)
  end
end

-- used to enable autocompletion (assign to every lsp server config)
local capabilities = cmp_nvim_lsp.default_capabilities()
```

```lua

-- Change the Diagnostic symbols in the sign column (gutter)
-- (not in youtube nvim video)
local signs = { Error = "    ", Warn = "    ", Hint = "    ", Info = "    " }
for type, icon in pairs(signs) do
  local hl = "DiagnosticSign" .. type
  vim.fn.sign_define(hl, { text = icon, texthl = hl, numhl = "" })
end


-- configure html server
lspconfig["html"].setup({
  capabilities = capabilities,
  on_attach = on_attach,
})

-- configure typescript server with plugin
typescript.setup({
  server = {
    capabilities = capabilities,
    on_attach = on_attach,
  },
})

-- configure css server
lspconfig["cssls"].setup({
  capabilities = capabilities,
  on_attach = on_attach,
})

-- configure tailwindcss server
lspconfig["tailwindcss"].setup({
  capabilities = capabilities,
  on_attach = on_attach,
})

-- configure emmet language server
lspconfig["emmet_ls"].setup({
  capabilities = capabilities,
  on_attach = on_attach,
  filetypes = { "html", "typescriptreact", "javascriptreact", "css", "sass", "scss", "less", "svelte" },
})

-- configure lua server (with special settings)
lspconfig["lua_ls"].setup({
  capabilities = capabilities,
  on_attach = on_attach,
  settings = { -- custom settings for lua
    Lua = {
      -- make the language server recognize "vim" global
      diagnostics = {
        globals = { "vim" },
      },
      workspace = {
        -- make language server aware of runtime files
        library = {
          [vim.fn.expand("$VIMRUNTIME/lua")] = true,
          [vim.fn.stdpath("config") .. "/lua"] = true,
        },
      },
    },
  },
})

-- configure cpp
lspconfig["clangd"].setup({
  capabilities = capabilities,
  on_attach = on_attach,
})

-- configure python
lspconfig["pyright"].setup({
  capabilities = capabilities,
  on_attach = on_attach,
```

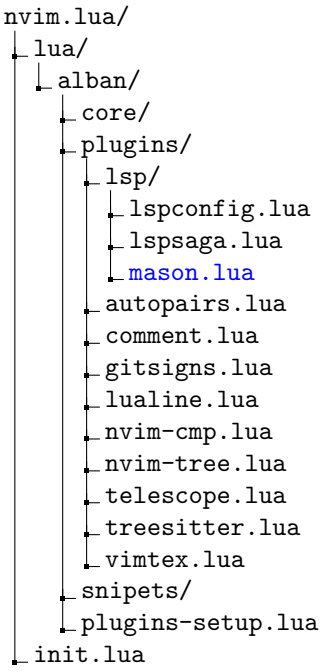
```lua
})

-- configure latex
lspconfig["texlab"].setup({
  capabilities = capabilities,
  on_attach = on_attach,
})

-- configure ocaml
lspconfig["ocamllsp"].setup({
  capabilities = capabilities,
  on_attach = on_attach,
})
```

9 lspsaga.lua



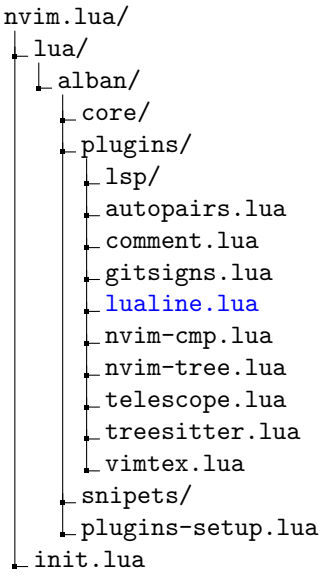

```lua
-- import lspsaga safely
local saga_status, saga = pcall(require, "lspsaga")
if not saga_status then
  return
end

saga.setup({
  -- keybinds for navigation in lspsaga window
  scroll_preview = { scroll_down = "<C-f>", scroll_up = "<C-b>" },
  -- use enter to open file with definition preview
  definition = {
    edit = "<CR>",
  },
  ui = {
    colors = {
      normal_bg = "#022746",
    },
  },
})
```

10 mason.lua



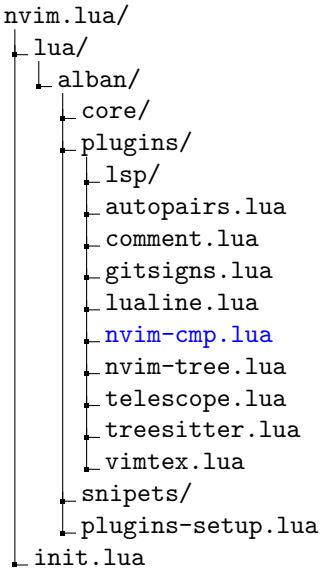

```lua
-- import mason plugin safely
local mason_status, mason = pcall(require, "mason")
if not mason_status then
  return
end

-- import mason-lspconfig plugin safely
local mason_lspconfig_status, mason_lspconfig = pcall(require, "mason-lspconfig")
if not mason_lspconfig_status then
  return
end

-- import mason-null-ls plugin safely
-- local mason_null_ls_status, mason_null_ls = pcall(require, "mason-null-ls")
-- if not mason_null_ls_status then
--   return
-- end

-- enable mason
mason.setup()

mason_lspconfig.setup({
  -- list of servers for mason to install
  ensure_installed = {
    "tsserver",
    "html",
    "cssls",
    "tailwindcss",
    "lua_ls",
    "emmet_ls",
    "clangd",
    "pyright",
    "ocamllsp"
  },
  -- auto-install configured servers (with lspconfig)
  automatic_installation = true, -- not the same as ensure_installed
})
```

11 lualine.lua



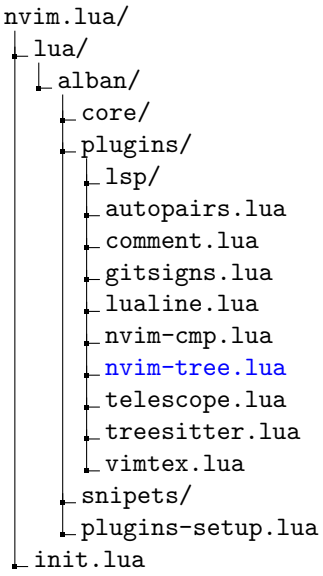

```lua
local status, lualine = pcall(require, "lualine")
if not status then
  return
end

-- get lualine nightfly theme
local lualine_nightfly = require("lualine.themes.nightfly")

-- new colors for theme
local new_colors = {
  blue = "#65D1FF",
  green = "#3EFFDC",
  violet = "#FF61EF",
  yellow = "#FFDA7B",
  black = "#000000",
}

-- change nightlfy theme colors
lualine_nightfly.normal.a.bg = new_colors.blue
lualine_nightfly.insert.a.bg = new_colors.green
lualine_nightfly.visual.a.bg = new_colors.violet
lualine_nightfly.command = {
  a = {
    gui = "bold",
    bg = new_colors.yellow,
    fg = new_colors.black, -- black
  },
}

-- configure lualine with modified theme
lualine.setup({
  options = {
    theme = lualine_nightfly,
  },
})
```

12 nvim-cmp.lua



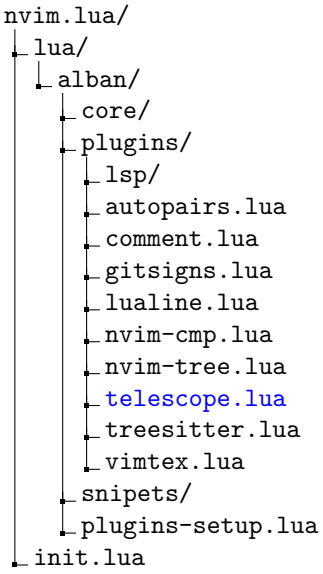

```lua
-- import nvim-cmp plugin safely
local cmp_status, cmp = pcall(require, "cmp")
if not cmp_status then
  return
end

-- import luasnip plugin safely
local luasnip_status, luasnip = pcall(require, "luasnip")
if not luasnip_status then
  return
end

-- import lspkind plugin safely
local lspkind_status, lspkind = pcall(require, "lspkind")
if not lspkind_status then
  return
end

require("luasnip.loaders.from_lua").load({paths = "~/.config/nvim/lua/alban/snipets/"})

luasnip.config.set_config({
  enable_autosnippets = true
})

vim.opt.completeopt = "menu,menuone,noselect"

cmp.setup({
  snippet = {
    expand = function(args)
      luasnip.lsp_expand(args.body)
    end,
  },
  mapping = cmp.mapping.preset.insert({
    ["<C-k>"] = cmp.mapping.select_prev_item(), -- previous suggestion
    ["<C-j>"] = cmp.mapping.select_next_item(), -- next suggestion
    ["<C-b>"] = cmp.mapping.scroll_docs(-4),
    ["<C-f>"] = cmp.mapping.scroll_docs(4),
    ["<C-Space>"] = cmp.mapping.complete(), -- show completion suggestions
    ["<C-e>"] = cmp.mapping.abort(), -- close completion window
    ["<CR>"] = cmp.mapping.confirm({ select = false }),
  }),
  -- sources for autocompletion
  sources = cmp.config.sources({
    { name = "nvim_lsp" }, -- lsp
    { name = "luasnip" }, -- snippets
    { name = "buffer" }, -- text within current buffer
    { name = "path" }, -- file system paths
  }),
  -- configure lspkind for vs-code like icons
  formatting = {
    format = lspkind.cmp_format({
      maxwidth = 50,
      ellipsis_char = "...",
    }),
  },
})
```

13 nvim-tree.lua



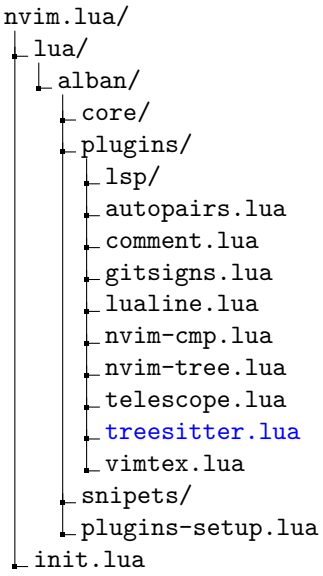

```lua
local setup, nvimtree = pcall(require, "nvim-tree")
if not setup then
  return
end

-- recommended settings from nvim-tree documentation
vim.g.loaded = 1
vim.g.loaded_netrwPlugin = 1

-- change color for arrows in tree to light blue
vim.cmd([[ highlight NvimTreeIndentMarker guifg=#3FC5FF ]])

nvimtree.setup({
  renderer = {
    icons = {
      glyphs = {
        folder = {
          arrow_closed = "    ", -- arrow when folder is closed
          arrow_open = "    ", -- arrow when folder is open
        },
      },
    },
  },
  -- disable window_picker for
  -- explorer to work well with
  -- window splits
  actions = {
    open_file = {
      window_picker = {
        enable = false,
      },
    },
  },
})
```

14 telescope.lua




```lua
-- import telescope plugin safely
local telescope_setup, telescope = pcall(require, "telescope")
if not telescope_setup then
  return
end

-- import telescope actions safely
local actions_setup, actions = pcall(require, "telescope.actions")
if not actions_setup then
  return
end

-- configure telescope
telescope.setup({
  -- configure custom mappings
  defaults = {
    mappings = {
      i = {
        ["<C-k>"] = actions.move_selection_previous, -- move to prev result
        ["<C-j>"] = actions.move_selection_next, -- move to next result
        ["<C-q>"] = actions.send_selected_to_qflist + actions.open_qflist,
    -- send selected to quickfixlist
      },
    },
  },
})

telescope.load_extension("fzf")
```

15 treesitter.lua




```lua
-- import nvim-treesitter plugin safely
local status, treesitter = pcall(require, "nvim-treesitter.configs")
if not status then
  return
end

-- configure treesitter
treesitter.setup({
  -- enable syntax highlighting
  highlight = {
    enable = true,
  },
  -- enable indentation
  indent = { enable = true },
  -- enable autotagging (w/ nvim-ts-autotag plugin)
  autotag = { enable = true },
  -- ensure these language parsers are installed
  ensure_installed = {
    "json",
    "javascript",
    "typescript",
    "tsx",
    "yaml",
    "html",
    "css",
    "markdown",
    "markdown_inline",
    "svelte",
    "graphql",
    "bash",
    "lua",
    "vim",
    "dockerfile",
    "gitignore",
    "c",
    "cpp",
    "python"
  },
  -- auto install above language parsers
  auto_install = true,
})
```

16 vimtex.lua

```
nvim.lua/
└─ lua/
   └─ alban/
      ├─ core/
      ├─ plugins/
      │  ├─ lsp/
      │  ├─ autopairs.lua
      │  ├─ comment.lua
      │  ├─ gitsigns.lua
      │  ├─ lualine.lua
      │  ├─ nvim-cmp.lua
      │  ├─ nvim-tree.lua
      │  ├─ telescope.lua
      │  ├─ treesitter.lua
      │  └─ vimtex.lua
      ├─ snipets/
      └─ plugins-setup.lua
└─ init.lua
```

```lua
local status, vimtex = pcall(require, "vimtex")
if not status then
  return
end

vimtex.setup()
```

17 plugins-setup.lua

```
nvim.lua/
├─ lua/
│  └─ alban/
│     ├─ core/
│     ├─ plugins/
│     ├─ snipets/
│     └─ plugins-setup.lua
└─ init.lua
```

```lua
-- auto install packer if not installed
local ensure_packer = function()
  local fn = vim.fn
  local install_path = fn.stdpath("data") .. "/site/pack/packer/start/packer.nvim"
  if fn.empty(fn.glob(install_path)) > 0 then
    fn.system({ "git", "clone", "--depth", "1", "https://github.com/wbthomason/packer.nvim", install_path })
    vim.cmd[[packadd packer.nvim]]
    return true
  end
  return false
end
local packer_bootstrap = ensure_packer() -- true if packer was just installed

-- autocommand that reloads neovim and installs/updates/removes plugins
-- when file is saved
vim.cmd([[
  augroup packer_user_config
    autocmd!
    autocmd BufWritePost plugins-setup.lua source <afile> | PackerSync
  augroup end
]])


-- import packer safely
local status, packer = pcall(require, "packer")
if not status then
  return
end

return packer.startup(function(use)
  use("wbthomason/packer.nvim")

  use("nvim-lua/plenary.nvim")          -- lua functions that plugins use

  use("bluz71/vim-nightfly-colors")     -- color theme

  use("christoomey/vim-tmux-navigator") -- tmux & split window navigation

  -- essential plugins
  use("tpope/vim-surround")               -- add, delete, change surroundings (it's awesome)
  use("inkarkat/vim-ReplaceWithRegister") -- replace with register contents using motion (gr + motion)

  -- commenting with gc
  use("numToStr/Comment.nvim")

  -- file explorer
  use("nvim-tree/nvim-tree.lua")

  -- icons
  use("nvim-tree/nvim-web-devicons")

  -- statusline
  use("nvim-lualine/lualine.nvim")

  -- fuzzy finding w/ telescope
  use({ "nvim-telescope/telescope-fzf-native.nvim", run = "make" }) -- dependency for better sorting performance
  use({ "nvim-telescope/telescope.nvim", branch = "0.1.x" })        -- fuzzy finder

  -- autocompletion
  use("hrsh7th/nvim-cmp")    -- completion plugin
  use("hrsh7th/cmp-buffer") -- source for text in buffer
  use("hrsh7th/cmp-path")    -- source for file system paths

  -- snippets
  use("L3MON4D3/LuaSnip")             -- snippet engine
  use("saadparwaiz1/cmp_luasnip")     -- for autocompletion
  use("rafamadriz/friendly-snippets") -- useful snippets
```

```lua

  -- managing & installing lsp servers, linters & formatters
  use("williamboman/mason.nvim")           -- in charge of managing lsp servers, linters & formatters
  use("williamboman/mason-lspconfig.nvim") -- bridges gap b/w mason & lspconfig

  -- configuring lsp servers
  use("neovim/nvim-lspconfig") -- easily configure language servers
  use("hrsh7th/cmp-nvim-lsp")  -- for autocompletion
  use({
    "glepnir/lspsaga.nvim",
    branch = "main",
    requires = {
      { "nvim-tree/nvim-web-devicons" },
      { "nvim-treesitter/nvim-treesitter" },
    },
  })
  use("lukas-reineke/lsp-format.nvim")
  -- enhanced lsp uis
  use("jose-elias-alvarez/typescript.nvim") -- additional functionality for typescript server (e.g. rename file & update imports)
  use("onsails/lspkind.nvim")               -- vs-code like icons for autocompletion


  -- treesitter configuration
  use({
    "nvim-treesitter/nvim-treesitter",
    run = function()
      local ts_update = require("nvim-treesitter.install").update({ with_sync = true })
      ts_update()
    end,
  })

  -- auto closing
  use("windwp/nvim-autopairs")                                  -- autoclose parens, brackets, quotes, etc...
  use({ "windwp/nvim-ts-autotag", after = "nvim-treesitter" }) -- autoclose tags

  -- git integration
  use("lewis6991/gitsigns.nvim") -- show line modifications on left hand side

  -- VimTex
  use("lervag/vimtex")

  -- Lua
  use {
    "folke/which-key.nvim",
    config = function()
      vim.o.timeout = true
      vim.o.timeoutlen = 300
      require("which-key").setup {
        -- your configuration comes here
        -- or leave it empty to use the default settings
        -- refer to the configuration section below
      }
    end
  }


  if packer_bootstrap then
    require("packer").sync()
  end
end)
```

18 tex.lua

```
nvim.lua/
  lua/
    alban/
      core/
      plugins/
      snipets/
        tex.lua
      plugins-setup.lua
  init.lua
```

```lua
local ls = require("luasnip")
-- some shorthands...
local s = ls.snippet
local sn = ls.snippet_node
local t = ls.text_node
local i = ls.insert_node
local f = ls.function_node
local c = ls.choice_node
local d = ls.dynamic_node
local r = ls.restore_node

local fmt = require("luasnip.extras.fmt").fmt

local snippets, autosnippets = {}, {}

table.insert(autosnippets, s("begin", fmt([[
\begin{{{}}}
  {}
\end{{{}}}
]],
  { i(1, ""), i(3, ""), i(2, "") }
)))

table.insert(autosnippets, s("bcode", fmt([[
\begin{{lstlisting}}[language={}]
{}
\end{{lstlisting}}
]],{
  i(1, ""), i(2, "")
})))

local replace_insert_0 = {
  {":al", "\\alpha"},
  {":Al", "\\Alpha"},
  {":be", "\\beta"},
  {":Be", "\\Beta"},
  {":ga", "\\gama"},
  {":Ga", "\\Gama"},
  {":mu", "\\mu"},
  {":Mu", "\\Mu"},
  {":mk", "$ $"},
  {":  ", "\\\\"}
}

local replace_insert_1 = {
  {":c", "\\section{{{}}}"},
  {":sc", "\\subsection{{{}}}"},
  {":ssc", "\\subsubsection{{{}}}"}
}

local replace_insert_2 = {
  {"frac", "\\frac{{{}}}{{{}}}"}
}

for _, item in ipairs(replace_insert_0) do
  table.insert(autosnippets, s(item[1], t(item[2])))
end

for _, item in ipairs(replace_insert_1) do
  table.insert(autosnippets, s(item[1], fmt(item[2], {i(1, "")} )))
end

for _, item in ipairs(replace_insert_2) do
  table.insert(autosnippets, s(item[1], fmt(item[2], {i(1, ""), i(2, "")} )
    ))
end

return snippets, autosnippets
```