Гайд по настройке VS Code под RoR

Настройка рабочего окружения в WSL

Многие из расширений VS Code хуже работают на Windows, да и в целом использование этой операционной системы при разработке на Ruby часто приводит к неожиданным проблемам. Один из вариантов обойти эти трудности - работать через WSL.
Предположим, вы решили следовать этому пути (если нет, можно пропускать действия в этом блоке). Установить WSL можно, например, по этому гайду, но в интернете можно найти и много других. Теперь установим всё необходимое для работы над проектом.

1. **Открываем консоль WSL.** В моём случае это Ubuntu 20.04.6LTS, но действия ниже должны быть аналогичны для любой +- актуальной Ubuntu.
2. **Устанавливаем PostgreSQL или вашу СУБД.** (гайд для PostgreSQL) https://www.digitalocean.com/community/tutorials/how-to-install-postgresql-on-ubuntu-20-04-quickstart
   Из отличий:
   - После установки PostgreSQL вместо создания новой роли можно поменять пароль для пользователя postgres (https://stackoverflow.com/questions/12720967/how-can-i-change-a-postgresql-user-password)
   - Пришлось отредактировать файл `pg_conf`, как указано тут: https://stackoverflow.com/questions/18664074/getting-error-peer-authentication-failed-for-user-postgres-when-trying-to-ge
   - **systemctl** у меня не сработал, вместо этого выполнял команду `sudo service postgresql start` (https://learn.microsoft.com/en-us/windows/wsl/tutorials/wsl-database)
3. **Устанавливаем Node.js** (нам нужна новая версия, поэтому через nvm) https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-ubuntu-20-04
4. **Устанавливаем Ruby**, лучше всего через rbenv https://www.digitalocean.com/community/tutorials/how-to-install-ruby-on-rails-with-rbenv-on-ubuntu-22-04
5. **Устанавливаем libpq-dev** (клиентская библиотека для PostgreSQL, без неё bundle install будет выдавать ошибку)

```
sudo apt install libpq-dev
```

6. **Устанавливаем Ruby on Rails** (`gem install rails`), создаем проект и делаем `bundle install`
7. **Настраиваем VS Code для работы с WSL** по этому гайду: https://code.visualstudio.com/docs/remote/wsl
8. `code .` в директории проекта на WSL, чтобы открыть его в VS Code

Расширения

1. Для работы с БД
    1. **SQLTools** https://marketplace.visualstudio.com/items?itemName=mtxr.sqltools
    2. **SQLTools PostgreSQL/Cockroach Driver** https://marketplace.visualstudio.com/items?itemName=mtxr.sqltools-driver-pg
2. Для поддержки Ruby
    1. Подсветка синтаксиса и автодополнение кода: **RubyLSP** https://marketplace.visualstudio.com/items?itemName=Shopify.ruby-lsp
    2. Отладчик: **VSCode rdbg Ruby Debugger** https://marketplace.visualstudio.com/items?itemName=KoichiSasada.vscode-rdbg
3. Для просмотра и запуска тестов: **Ruby Test Explorer** https://marketplace.visualstudio.com/items?itemName=connorshea.vscode-ruby-test-adapter
4. *Необязательно*: рекомендованные общие расширения, полезные и для других языков
    1. **Error Lens** - подсветка ошибок и ворнингов https://marketplace.visualstudio.com/items?itemName=usernamehw.errorlens
    2. **GitHub Theme** - цветовая тема от GitHub https://marketplace.visualstudio.com/items?itemName=GitHub.github-vscode-theme
    3. **Live Share** - для одновременного редактирования файлов проекта https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare
    4. **Remote SSH** - для удобного подключения к удаленным серверам https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-ssh
    5. **Auto Rename Tag** - для удобного переименования тэгов HTML https://marketplace.visualstudio.com/items?itemName=formulahendry.auto-

rename-tag

6. **Tailwind CSS IntelliSense** - если вы используете TailwindCSS для стилей https://marketplace.visualstudio.com/items?itemName=bradlc.vscode-tailwindcss

Настройка расширений

1. Добавляем подключение в **SQLTools** после того, как создали базу данных. Нужно при создании подключения выбрать Postgres, заполнить имя бд, имя пользователя, и пароль (рекомендую выбирать *Save as plain text in settings* как самый простой вариант. Ваш пароль от локальной БД все равно никому не нужен)
2. На вкладке тестов изначально два окна, одно из них пишет что нет расширений для тестов, а другое отображает тесты, найденные расширением **Ruby Test Explorer**. Верхнее окно сворачиваем или убираем, чтобы не мешало. Отладка тестов в Ruby Test Explorer работать не будет, а вот запуск и переход к файлам тестов - пожалуйста.
3. **RubyLSP** - должен работать без дополнительных настроек. Можно установить какой-нибудь linter вроде RuboCop (просто как gem в Gemfile) - RubyLSP сам его подхватит.

Настройка запуска проекта

Настраиваем дебагер

Ctrl+Shift+P открывает панель команд VS Code, в ней начинаем набирать `Debug: Add Configuration...`
Выбираем rdbg, создается файл `launch.json` в папке `.vscode` проекта. Его можно не менять, главное оставить в нём эту запись:

```
"configurations": [
        {
                "type": "rdbg",
                "name": "Attach with rdbg",
                "request": "attach"
        },
        // тут могут быть другие записи
]
```

В окне **Run and Debug** панели слева необходимо выбрать эту конфигурацию (Attach with rdbg)

Настраиваем foreman

Открываем `Procfile.dev`, в нем меняем то, что написано после **web:** на

```
rdbg -n --open=vscode -- bin/rails s
```

> Хорошей практикой будет вынести vscode из первой команды в переменную окружения, т.к. ссылаться на конкретную IDE в репозитории не очень хорошо. Тогда команда будет выглядеть следующим образом:

```
rdbg -n --open=${RGBG_IDE} -- bin/rails s
```

Теперь запускаем проект через foreman, а когда хотим дебажить, нажимаем **F5** и подключаемся отладчиком VS Code.

```
foreman start -s Procfile.dev
```

Добавляем task для запуска foreman по горячим клавишам

Tasks в VS Code - это пользовательские скрипты, которые можно вызывать комбинацией клавиш **Ctr+Shift+B**: https://code.visualstudio.com/docs/editor/tasks

Проходим по менюшкам и выбираем:
Terminal -> Configure Tasks -> Create tasks.json file from template -> Others.

После этого в папке `.vscode` проекта будет автоматически создан файлик `tasks.json`, в который нужно добавить следующее задание:

```
"tasks": [
        {
                "label": "Foreman start",
                "type": "shell",
                "command": "foreman start -f Procfile.dev",
                "group": "build",
                "presentation": {
                        "reveal": "always",
                        "panel": "shared",
                }
        }
]
```

Теперь **Ctrl + Shift + B** запустит foreman! В этот же файл можно через запятую добавить ещё заданий(например, для запуска тестов или для деплоя), и тогда по **Ctrl + Shift + B** будет вызываться диалоговое окно с выбором задания для запуска

> Для подключения дебагером к запущенному приложению, достаточно нажать **F5**, и точки останова вместе с консолью отладчика будут работать.