ETEC ORLANDO QUAGLIATO

Desenvolvimento de sistemas

MURILO VIDOR DIAS

RAFAEL SALANDIN MORAES

WESLEY AMADEU STEFANO

RECIPE PER NOTE

UM SISTEMA WEB DE ORGANIZAÇÃO PESSOAL

SANTA CRUZ DO RIO PARDO-SP

2021

ETEC ORLANDO QUAGLIATO

MURILO VIDOR DIAS

RAFAEL SALANDIN MORAES

WESLEY AMADEU STEFANO

RECIPE PER NOTE

UM SISTEMA WEB DE ORGANIZAÇÃO PESSOAL

Projeto elaborado pelos alunos do curso de Desenvolvimento de Sistemas, apresentado à Disciplina PTCC, como requisito parcial de nota.

Orientadores: Raquel Santana Fonseca Rodrigues Gonçalves.

SANTA CRUZ DO RIO PARDO-SP

2021

Agradecimentos

Em nossos agradecimentos dedicamos nossa gratidão aos professores Carlos Eduardo da Silva, David Silva e a Mara Arcoleze. Devido ao auxílio em nosso TCC em conjunto com a nossa orientadora.

Outros que somos gratos são Alexandre de Souza, Miguel Gonzaga, Fernando Mendes, Luiz Gustavo Gazola. Com ajuda de novas ideias, correções e códigos

E a todos os deuses pois somos um grupo laico.

Epígrafe

“Até agora os filósofos se preocuparam em interpretar o mundo de várias formas. O que importa é transformá-lo.”

KARL MARX

RESUMO

Com o passar do tempo, o dia a dia vai ficando cada vez mais corrido, e com isso se torna mais complicado organizar a sua rotina. Isso fez com que aparecesse mais ferramentas para ajudar nessa tarefa. Na busca de fazer uma boa ferramenta para anotações, é apresentado o “Recipe per Note”, onde introduz uma plataforma web que ajuda a fazer anotações, tendo três modalidades de organização, o “To Do” para anotar tarefas, objetivos, tendo uma data limite para sua finalização; o “Notes” tem um espaço para anotações maiores, tipo post-it e o “WatchLater” para guardar links para poder assistir depois. E no Home do Site possuí um resumo com os ToDos do dia, os Notes e WatchLater em baixo.

Palavras-chaves: Anotações. Organização. Tarefa.

ABSTRACT

In the pass of the time, the day a day ar becoming more and more rushed, and with that, becomes more complicated organize your rotine. That makes apears more tools for help in that work. Following the idea of making a great tool for annotations is presented “Recipe per Note”, were is introduces a web platform that helps to make annotations, having three modalities of organization, “To Do” for annotate tasks, goals, that have a limit day to do; “Note” for write down bigger annotations, post-it like and “WatchLater” where can hold up links for watch later. And in the Home of the Web site has a brief with the ToDos for that day, the Notes and WatchLater below then.

Keywords: Annotations. Organization. Works

Sumário

Lista de Código Fonte

Lista de Figuras

Lista de Telas

1 Introdução

Tendo em vista que uma boa organização se faz necessário para que ocorra uma gestão e compreensão em diversos âmbitos, como social, empresarial e pessoal, percebe-se assim sua necessidade diária.

O significado de organização remete a capacidade de preservar a estabilidade e prosseguir facilmente os objetivos. Como tal pensamento e ação leva-se a uma melhoria em diversas áreas.

No âmbito social a organização reflete com a harmonia, ética e moral entre os indivíduos. Isto é, com a regência da moral sobre determinado grupo seu cumprimento é destacado, assim junto da ética. Como resultado criando-se a harmonia com a sociedade.

Para o quesito empresarial é composto por funções e cargos hierárquicos aos quais todos são submetidos. Com visões lógicas e racionais deve-se visar seus processos. Sua formação deve coordenar e dividir todos os seus afazeres e responsabilidades, pautando-se para um objetivo em comum em sua empresa.

Como fora dito acima, a organização auxilia de forma positiva e trazendo estabilidade, mas para que seus resultados sejam obtidos só serão garantidos quando realizada a área pessoal de organização.

Sua importância é correlacionada com as adversidades que a falta traz. Dentro delas estão a falta de motivação, produtividade e estresse, com esses defeitos atrelados torna-se incapaz uma coexistência estável em quaisquer dos âmbitos. Outrora com seu regimento ocorresse uma distribuição do tempo, planejamento e foco no prol dos objetivos, melhorando gradativamente a saúde e estabilidade.

Como isso, se viu necessário uma solução para organização pessoal, já que há muitos meios para isso, o que pode deixar as pessoas confusas. Assim veio a ideia do Recipe Per Note. Um site onde se tem categorias ToDos, WatchLater e Notes, para poder se organizar melhor.

Com o ToDos, é possível marcar as suas tarefas, e visualizar o tempo de finalização da tarefa por texto e visualmente. Já com o WatchLater você pode deixar links de sites, vídeos, textos para ver depois. E com o Notes permite um sistema de anotações similar a um Post-it, para lembrar de coisas mais simples.

2 Justificativa

É fácil reparar que cada vez mais a desorganização e esquecimento estão presentes no cotidiano, com isso, depois de alguns meses se viu que seria preciso uma ferramenta de anotações prática para uma melhora nesse defeito. Uma das causas da existência desse problema foi a alta instabilidade encontrada no cotidiano de alunos e adultos.

3 Objetivos

3.1 Geral:

- Melhorar a organização pessoal do usuário, através de anotações.
- Provendo categorias de separação das anotações.

3.2 Específicos:

- Disponibilizar meios de organização.
- Permitir alterações e atualizações nos dados.
- Lembretes para o usuário.

4 Idealização

Em sua idealização, compreende-se que seria mais pratico a utilização da linguagem Elixir, junto com o framework Phoenix, onde uma vez em que dois terços do curso foi realizado de forma remota, foi possível aprender outras linguagens junto com o progresso do curso, além disso, o Phoenix e Elixir, permitem uma prototipagem rápida, o que ajudaria ainda mais na continuação desse projeto.

Tendo isso em mente, o sistema foi criado utilizando o Framework Phoenix escrito com a linguagem Elixir, sendo utilizado no Full Stack. Junto com o Framework, temos a biblioteca Ecto, que facilita a conexão com o banco de dados PostgreSQL.

Com a dependência bamboo, foi feito um sistema para entrega de email, se registra uma conta pela primeira vez.

Utilizando o site Figma, foi feito uma moldagem de como ficará a interface do sistema.

O host do site está sendo pelo o Ngrok, onde passando a porta do seu site, torna o computador pessoal num servidor web, permitindo que pessoas de fora vejam o seu site.

5 Fundamentação teórica

5.1 CSS

Segundo Gonçalves (2020) CSS é a sigla para o termo inglês *Cascading Style Sheets*, que significa Folha de Estilo em Cascatas. Nele estará separado o conteúdo da representação visual do site escrito em HTML.

Hoje em dia quase todos os navegadores suportam o CSS pois segundo o site do WDN, ela é uma das principais linguagens da open web e é padronizada em navegadores web. É concluído que utilizando o CSS você customiza a sua página web. Para escrevê-lo, você não precisa de mais do que um editor de texto, entretanto existem diversas ferramentas disponíveis que tornam isso ainda mais fácil.

Figura 1. Logo CSS

5.2 Docker

Docker é um projeto da comunidade open source; suas ferramentas resultantes; a empresa Docker Inc. que é a principal apoiadora do projeto; e as ferramentas compatíveis formalmente com a empresa.

No artigo da Red Hat (2021) é explicado que é possível usar containers como se fossem máquinas virtuais modulares leves. Além de que os containers lhe dão uma maior flexibilidade para você criar, implantar, copiar ou migrar um container de um ambiente para o outro.

A tecnologia Docker usa o kernel do Linux e recursos do kernel como Cgroups e namespaces para segregar processos. Assim sendo possível fazer com que sejam executados de maneira independente. O objetivo dos containers é criar essa independência: a habilidade de executar diversos processos e aplicações separadamente para utilizar melhor a infraestrutura e, ao mesmo tempo, manter a segurança que você teria em sistemas separados.

No mesmo artigo citado anteriormente, se vê que as ferramentas de container, incluindo o Docker, fornecem um modelo de implantação com base em imagem. Isso facilita o compartilhamento de uma aplicação ou conjunto de serviços, incluindo todas as dependências deles em vários ambientes. O Docker também automatiza a implantação da aplicação (ou de conjuntos de processos que constituem uma aplicação) dentro desse ambiente de container.

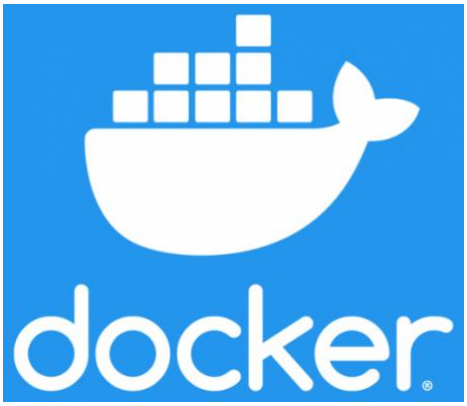


Figura 2. Logo Docker

5.3Elixir

É introduzido por Souza (2020), Elixir é uma linguagem funcional dinâmica, ela é executada na máquina virtual Erlang. Elixir compila em cima de Erlang para fornecer aplicações distribuídas, tolerante a falhas além disso, também se entende para suportar metaprogramação com macros e polimorfismo via protocolos. Alguns benefícios do Elixir é a execução distribuída e a tolerância a falhas.

Como Santos (2020) fala, o Elixir tem seu código executado em pequenas threads, chamadas de processo, e esses processos são totalmente isolados e trocam informações por mensagens, devido isso, não é incomum ter centenas de milhares de processos rodando ao mesmo tempo. Os processos também podem se comunicar

entre outros processos em máquinas diferentes na mesma rede. Isso fornece a base para a distribuição, permitindo que os desenvolvedores coordenem o trabalho em vários nodes.

Na comunidade do erlang/Elixir, possui o conceito chamado “let it crash” (“deixe falhar”). Pois a única certeza que sobre um código que é executado é que as coisas podem dar errado. Para isso é implementado no Elixir um feature de supervisão do código que está sendo executado. A combinação da tolerância a falhas e a programação orientada a eventos via a passagem de mensagens torna o Elixir uma ótima escolha para programação reativa e arquiteturas robustas.

Figura 3. Logo Elixir

5.4 Git

Em seu texto Reis (2020) explica que o Git é um sistema de controle de versões distribuído, sendo usado para o desenvolvimento de software, entretanto, pode ser usado para registrar o histórico de edições de qualquer tipo de arquivo. Pois cada diretório de trabalho do Git é um repositório com um histórico completo e habilidade total de acompanhamento das revisões, não dependente de acesso a uma rede ou a um servidor central.

Sendo assim, é usado para desenvolver projetos na qual várias pessoas podem contribuir ao mesmo tempo, criando arquivos, editando e permitindo que possam existir sem o risco de suas alterações serem sobrescritas.

Figura 4. Logo Git

5.5 HTML

Marques (2019) introduz ao HTML dizendo que a sua criação é atribuída no início dos anos 1990 pelo Tim Berners-Lee, o mesmo inventor da *World Wide Web* (www). O acrônimo HTML *significa HyperText Markup Language*, traduzindo ao português: Linguagem de Marcação de Hipertexto. O hipertexto é referente aos links que conectam as páginas da Web entre si, seja dentro de um mesmo site ou entre vários sites que organizam conhecimentos e guardam informações.

É complementado pelo site da MDN (2021) que ao trabalhar com HTML simplesmente se é codificado estruturas (tags e atributos) para marcar a página de um site. Por exemplo, a criação de um parágrafo colocando o texto entre as tags e . O nome de um elemento dentro de uma tag pode ser escrito em maiúscula, minúscula ou uma mistura, pois ela é insensível quanto a maiúsculas e minúsculas.

Para chegar nos dias de hoje, o HTML passou por diversas melhorias, e agora é componente básico de todos os tipos de páginas acessíveis de um navegador web, seja blogs, seja e-commerces ou redes sociais. Portanto, ele define o significado e a estrutura do conteúdo da web. Os sites em HTML normalmente são construídos em conjunto de outras linguagens, como o CSS e o JavaScript, que formam a base de todos os websites atuais.

Figura 5. Logo HTML

5.6 Markdown

Como Craveiro (2020) diz, o Markdown é uma linguagem de marcação, assim como o HTML, pensada para os escritores Web. Desenvolvida por John Gruber e Aaron Swatz para facilitar a visualização e a escrita do texto, pode ser facilmente convertida para o HTML e é totalmente open source e de ótima portabilidade.

Por ser de ótima portabilidade, a linguagem Markdown é utilizada em grandes empresas como o Github, Microsoft Word e o WordPress.

Segundo Castro (2004), pelo fato de o Markdown possuir sintaxes simples a codificação realmente fica mais “limpa”, fácil de escrever e ler o conteúdo. Por ter pouca sintaxe ele acaba tendo um tamanho muito pequeno e dificilmente acaba “quebrando”.

As sintaxes utilizadas pelo Markdown são basicamente símbolos ou caracteres não-alfabéticos padronizados, como por exemplo: “#”, “\”, “*”, “!“, que são usados para configurar títulos, listas, itálico e negrito.

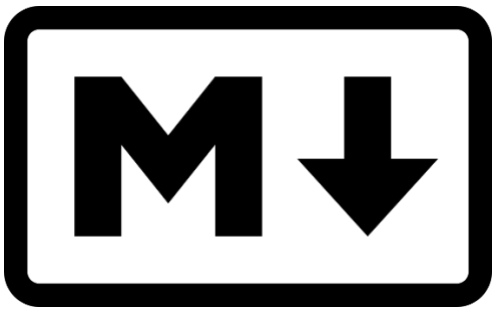


Figura 6. Logo Markdown

5.7 Ngrok

Como Paixão (2021) explica em seu artigo o Ngrok é uma ferramenta CLI (Comand Line Interface), que permite criar um túnel seguro, atrás de NATs e Firewalls, que te expõem serviços locais para a internet, de forma fácil e segura.

Figura 7. Logo Ngrok

5.8 Phoenix

Em sua documentação (2021), se possui a introdução ao Phoenix, que é um Framework de desenvolvimento web escrito em Elixir, no qual implementa o padrão Model View Controller (MVC) do lado do servidor. Muitos dos componentes e conceitos utilizados podem parecer familiar com a experiência de usar outros frameworks como Ruby on Rails ou Django do Python.

O Framework provê o melhor dos dois mundos, uma alta produtividade de desenvolvimento e uma aplicação de alta performance. O que pode ser interessante para a implementação de canais com a implementação de novidade em tempo real.

O Phoenix provê uma suave comunicação com os clientes externos usando o WebSockets.

Figura 8. Logo Phoenix

5.8.1 Phoenix LiveView

Como Noleto (2020) apresenta, o Phoenix LiveView é uma biblioteca que funciona em cima do Phoenix, trazendo o poder de criar aplicações ricas em tempo real. Tudo via server side e renderizando HTML.

O LiveView usa todo o potencial do Elixir com os websocklets e a comunicação em tempo real, fazendo que sua página pareça uma aplicação controlada com um Framework JavaScripit, mas é apenas HTML + CSS e Back-end.

Em seu artigo, Noleto (2020) explica melhor sobre o como o Phoenix Live View funciona.

5.9 PostgreSQL

Em sua documentação (2021), é introduzido ao PostgreSQL, que é um sistema de gerenciamento de banco de dados objeto-relacional, baseado no POSTGRES 4.2 e desenvolvido pelo Departamento de Ciência da Computação da Universidade da Califórnia em Berkeley no ano de 1986, ele é considerado um dos mais avançado na área de banco de dados de código aberto, ou seja, você está livre para usar, modificar e distribuir o PostgreSQL para qualquer finalidade, seja pessoal, para estudos ou comercial, livre de quaisquer encargos.

Um dos pioneiros em vários conceitos, que só mais tarde foram disponibilizados por alguns outros bancos de dados comerciais. Ele suporta grande parte do padrão SQL e funcionalidades modernas como: visões, integridade transacional, chaves estrangeiras, controle de simultaneidade multiversão, e muitas outras funções, dados, operadores entre outras coisas que podem ser adicionadas pelo usuário. Todos os anos uma nova versão majoritária é lançada com dezenas de novas funcionalidades. As versões têm ciclo de vida de 5 anos.

Figura 9. Logo PostgreSQL

5.10 Softwares

5.10.1 Figma

O Figma é uma ferramenta de design colaborativa online. Flinco(2021) explica mais sobre o Figma, onde além de nos introduzir ao figma mostra os pontos bons dele como o fato dele ser gratuito, e até mesmo o fato de não termos que fazer o download do arquivo para compartilhar um projeto, mas sim só mandar um link gerado para isso.

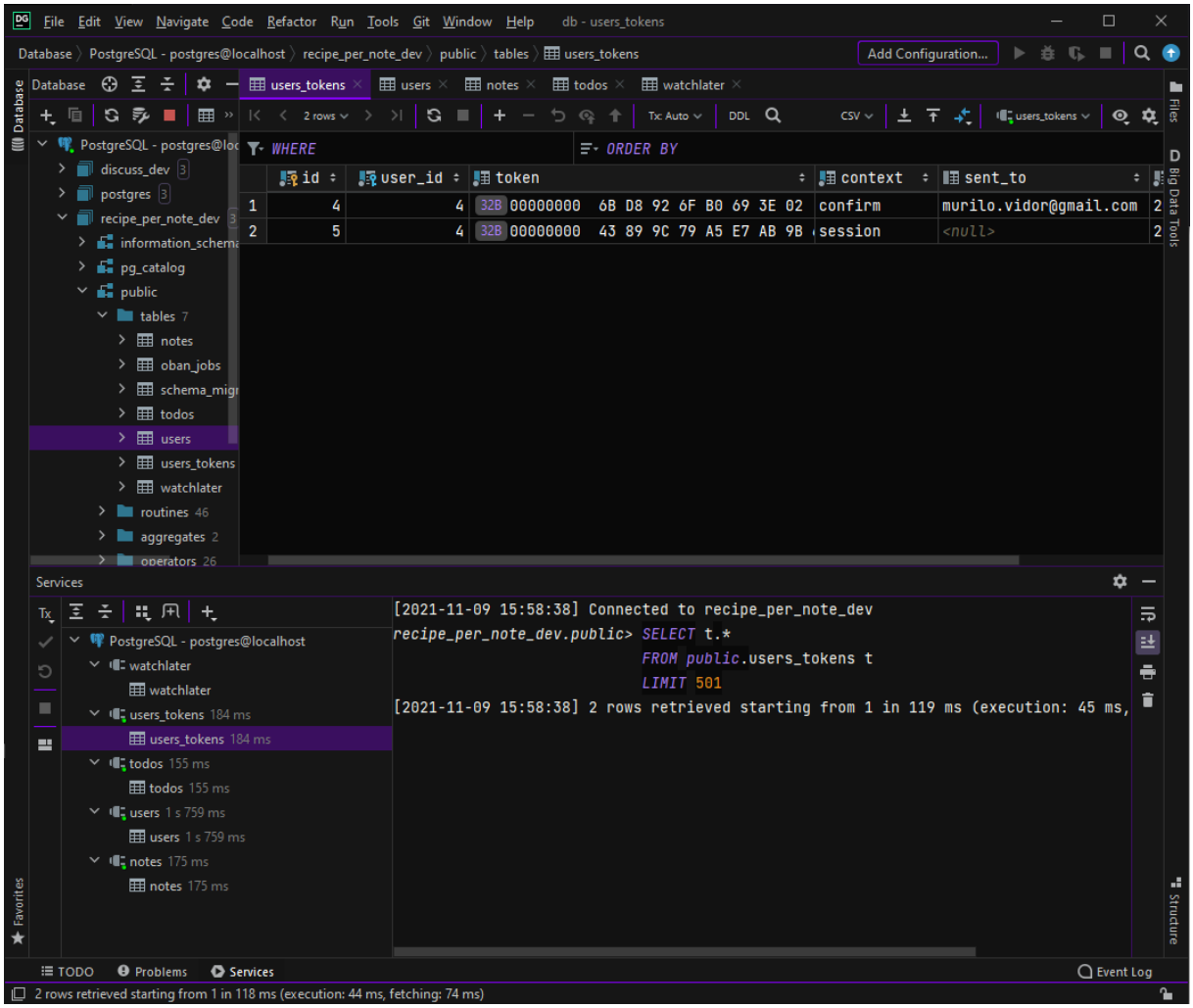
Figura 10. Logo Figma

5.10.2 DataGrip

Destacado em sua documentação (2021), o DataGrip é um Ambiente de gerenciamento de banco de dados para desenvolvedores. Criado pela empresa JetBrains.

O artigo da L3 software destaca que:

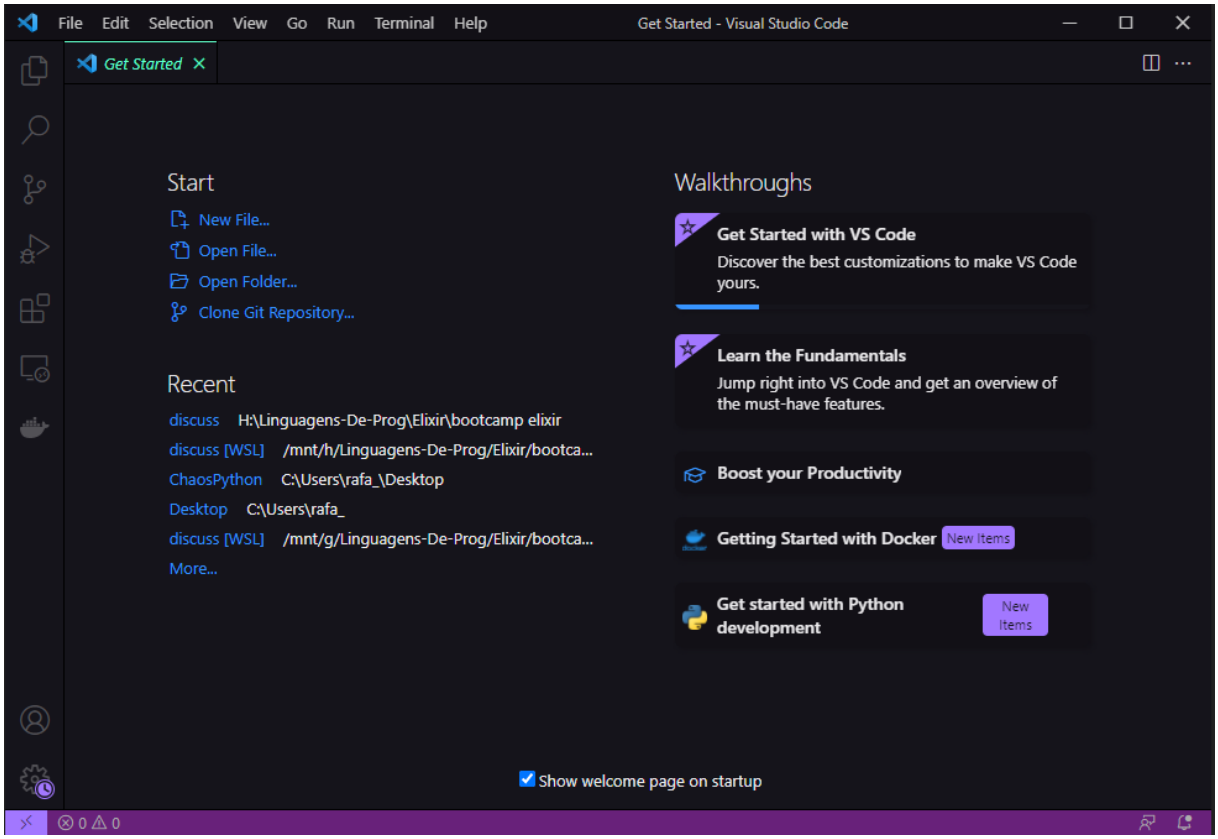
> A plataforma oferece uma navegação não linear, de forma que o desenvolvedor possa consultar qualquer tabela e executar consultas rápidas de diferentes modos. Destaque para o histórico das atividades realizadas, uma maneira bem eficiente de evitar que qualquer parte do trabalho seja perdida.
> A conclusão inteligente de código é outra funcionalidade bem atrativa para os desenvolvedores de banco de dados SQL. Isso porque, para preencher as linhas e colunas, é necessário repetir determinadas informações inúmeras vezes ao longo do projeto.



Tela 1. DataGrip com uma tabela SQL

5.10.3 Visual Studio Code

Como Edson(2016) define, Visual Studio Code é um editor de código destinado ao desenvolvimento de aplicações web. É uma ferramenta leve e multiplataforma, disponível para Windows, Linux e Mac OS, ele possui suporte a sintaxe de diversas linguagens como Python, Ruby, C++, Php, Silq, Elixir, entre outras.



Tela 2. VisualStudio Code inicial

6 Recipe per Note

6.1 Back end

Para o Backend foi usado o Elixir, que provê contextos para trabalhar com o código ao invés de Objetos – como se tem em linguagens como Python ou R, com isso, foi usado dois contextos o ´accounts´ para se trabalhar com o usuário. E o ´annotations´ para se trabalhar com os Notes, ToDos e WatchLater.

6.1.1 Contexto Accounts

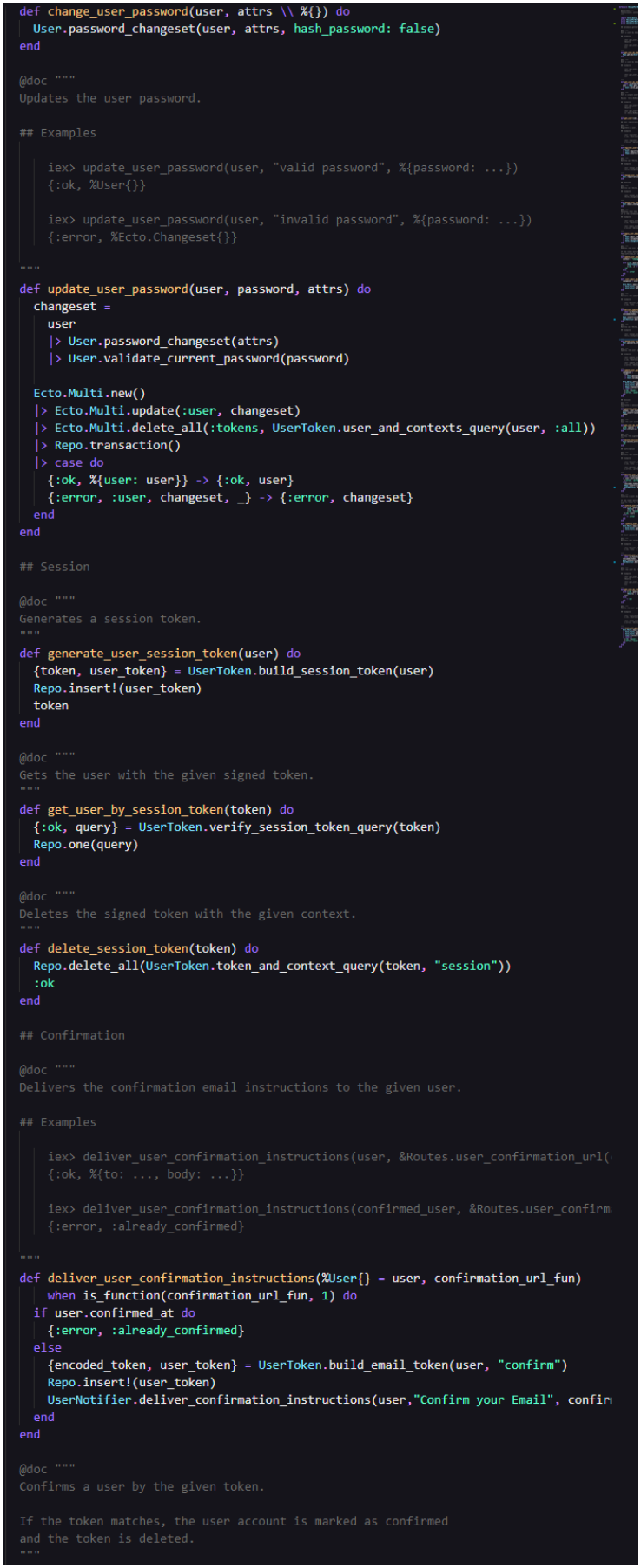
Com o contexto Accounts foi trabalhado os códigos de conexão com o banco de dados referente ao usuário.

```elixir
defmodule RecipePerNote.Accounts do

  @moduledoc """
  The Accounts context.
  """       SaLandin, 3 months ago • adding the firt part of the project :)

  import Ecto.Query, warn: false
  alias RecipePerNote.Repo
  alias RecipePerNote.Accounts.{User, UserToken, UserNotifier}
  alias RecipePerNote.Mailer

  ## Database getters

  @doc """
  Gets a user by email.

  ## Examples

      iex> get_user_by_email("foo@example.com")
      %User{}

      iex> get_user_by_email("unknown@example.com")
      nil

  """
  def get_user_by_email(email) when is_binary(email) do
    Repo.get_by(User, email: email)
  end

  @doc """
  Gets a user by email and password.

  ## Examples

      iex> get_user_by_email_and_password("foo@example.com", "correct_password")
      %User{}

      iex> get_user_by_email_and_password("foo@example.com", "invalid_password")
      nil

  """
  def get_user_by_email_and_password(email, password)
      when is_binary(email) and is_binary(password) do
    user = Repo.get_by(User, email: email)
    if User.valid_password?(user, password), do: user
  end

  @doc """
  Gets a single user.

  Raises `Ecto.NoResultsError` if the User does not exist.

  ## Examples

      iex> get_user!(123)
      %User{}

      iex> get_user!(456)
      ** (Ecto.NoResultsError)

  """
  def get_user!(id), do: Repo.get!(User, id)

  ## User registration

  @doc """
  Registers a user.

  ## Examples

      iex> register_user(%{field: value})
      {:ok, %User{}}

      iex> register_user(%{field: bad_value})
      {:error, %Ecto.Changeset{}}

  """
  def register_user(attrs) do
    %User{}
    |> User.registration_changeset(attrs)
    |> Repo.insert()
  end

  @doc """
  Returns an `%Ecto.Changeset{}` for tracking user changes.

  ## Examples

      iex> change_user_registration(user)
      %Ecto.Changeset{data: %User{}}

  """
```

Código Fonte 1. Accounts 1/4

```elixir
def change_user_registration(%User{} = user, attrs \\ %{}) do
  User.registration_changeset(user, attrs, hash_password: false)
end

## Settings

@doc """
Returns an `%Ecto.Changeset{}` for changing the user email.

## Examples

    iex> change_user_email(user)
    %Ecto.Changeset{data: %User{}}

"""
def change_user_email(user, attrs \\ %{}) do
  User.email_changeset(user, attrs)
end

@doc """
Emulates that the email will change without actually changing
it in the database.

## Examples

    iex> apply_user_email(user, "valid password", %{email: ...})
    {:ok, %User{}}

    iex> apply_user_email(user, "invalid password", %{email: ...})
    {:error, %Ecto.Changeset{}}

"""
def apply_user_email(user, password, attrs) do
  user
  |> User.email_changeset(attrs)
  |> User.validate_current_password(password)
  |> Ecto.Changeset.apply_action(:update)
end

@doc """
Updates the user email using the given token.

If the token matches, the user email is updated and the token is deleted.
The confirmed_at date is also updated to the current time.
"""
def update_user_email(user, token) do
  context = "change:#{user.email}"

  with {:ok, query} <- UserToken.verify_change_email_token_query(token, context),
       %UserToken{sent_to: email} <- Repo.one(query),
       {:ok, _} <- Repo.transaction(user_email_multi(user, email, context)) do
    :ok
  else
    _ -> :error
  end
end

defp user_email_multi(user, email, context) do
  changeset = user |> User.email_changeset(%{email: email}) |> User.confirm_changese

  Ecto.Multi.new()
  |> Ecto.Multi.update(:user, changeset)
  |> Ecto.Multi.delete_all(:tokens, UserToken.user_and_contexts_query(user, [context
end

@doc """
Delivers the update email instructions to the given user.

## Examples

    iex> deliver_update_email_instructions(user, current_email, &Routes.user_update_
    {:ok, %{to: ..., body: ...}}

"""
def deliver_update_email_instructions(%User{} = user, current_email, update_email_ur
    when is_function(update_email_url_fun, 1) do
  {encoded_token, user_token} = UserToken.build_email_token(user, "change:#{current_

  Repo.insert!(user_token)
  UserNotifier.deliver_update_email_instructions(user, "Update your Email", update_e
end

@doc """
Returns an `%Ecto.Changeset{}` for changing the user password.

## Examples

    iex> change_user_password(user)
    %Ecto.Changeset{data: %User{}}

"""
```

Código Fonte 2 .Accounts 2/4



```elixir
def change_user_password(user, attrs \\ %{}) do
  User.password_changeset(user, attrs, hash_password: false)
end

@doc """
Updates the user password.

## Examples

    iex> update_user_password(user, "valid password", %{password: ...})
    {:ok, %User{}}

    iex> update_user_password(user, "invalid password", %{password: ...})
    {:error, %Ecto.Changeset{}}

"""
def update_user_password(user, password, attrs) do
  changeset =
    user
    |> User.password_changeset(attrs)
    |> User.validate_current_password(password)

  Ecto.Multi.new()
  |> Ecto.Multi.update(:user, changeset)
  |> Ecto.Multi.delete_all(:tokens, UserToken.user_and_contexts_query(user, :all))
  |> Repo.transaction()
  |> case do
    {:ok, %{user: user}} -> {:ok, user}
    {:error, :user, changeset, _} -> {:error, changeset}
  end
end

## Session

@doc """
Generates a session token.
"""
def generate_user_session_token(user) do
  {token, user_token} = UserToken.build_session_token(user)
  Repo.insert!(user_token)
  token
end

@doc """
Gets the user with the given signed token.
"""
def get_user_by_session_token(token) do
  {:ok, query} = UserToken.verify_session_token_query(token)
  Repo.one(query)
end

@doc """
Deletes the signed token with the given context.
"""
def delete_session_token(token) do
  Repo.delete_all(UserToken.token_and_context_query(token, "session"))
  :ok
end

## Confirmation

@doc """
Delivers the confirmation email instructions to the given user.

## Examples

    iex> deliver_user_confirmation_instructions(user, &Routes.user_confirmation_url(
    {:ok, %{to: ..., body: ...}}

    iex> deliver_user_confirmation_instructions(confirmed_user, &Routes.user_confirm
    {:error, :already_confirmed}

"""
def deliver_user_confirmation_instructions(%User{} = user, confirmation_url_fun)
    when is_function(confirmation_url_fun, 1) do
  if user.confirmed_at do
    {:error, :already_confirmed}
  else
    {encoded_token, user_token} = UserToken.build_email_token(user, "confirm")
    Repo.insert!(user_token)
    UserNotifier.deliver_confirmation_instructions(user,"Confirm your Email", confir
  end
end

@doc """
Confirms a user by the given token.

If the token matches, the user account is marked as confirmed
and the token is deleted.
"""
```

Código Fonte 3. Accounts 3/4

```elixir
  def confirm_user(token) do
    with {:ok, query} <- UserToken.verify_email_token_query(token, "confirm"),
         %User{} = user <- Repo.one(query),
         {:ok, %{user: user}} <- Repo.transaction(confirm_user_multi(user)) do
      {:ok, user}
    else
      _ -> :error
    end
  end

  defp confirm_user_multi(user) do
    Ecto.Multi.new()
    |> Ecto.Multi.update(:user, User.confirm_changeset(user))
    |> Ecto.Multi.delete_all(:tokens, UserToken.user_and_contexts_query(user, ["confirm
  end

  ## Reset password

  @doc """
  Delivers the reset password email to the given user.

  ## Examples

      iex> deliver_user_reset_password_instructions(user, &Routes.user_reset_password_
      {:ok, %{to: ..., body: ...}}

  """
  def deliver_user_reset_password_instructions(%User{} = user, reset_password_url_fun)
      when is_function(reset_password_url_fun, 1) do
    {encoded_token, user_token} = UserToken.build_email_token(user, "reset_password")
    Repo.insert!(user_token)
    UserNotifier.deliver_reset_password_instructions(user, "Reset your password",reset_
  end

  @doc """
  Gets the user by reset password token.

  ## Examples

      iex> get_user_by_reset_password_token("validtoken")
      %User{}

      iex> get_user_by_reset_password_token("invalidtoken")
      nil

  """
  def get_user_by_reset_password_token(token) do
    with {:ok, query} <- UserToken.verify_email_token_query(token, "reset_password"),
         %User{} = user <- Repo.one(query) do
      user
    else
      _ -> nil
    end
  end

  @doc """
  Resets the user password.

  ## Examples

      iex> reset_user_password(user, %{password: "new long password", password_confirm
      {:ok, %User{}}

      iex> reset_user_password(user, %{password: "valid", password_confirmation: "not
      {:error, %Ecto.Changeset{}}

  """
  def reset_user_password(user, attrs) do
    Ecto.Multi.new()
    |> Ecto.Multi.update(:user, User.password_changeset(user, attrs))
    |> Ecto.Multi.delete_all(:tokens, UserToken.user_and_contexts_query(user, :all))
    |> Repo.transaction()
    |> case do
      {:ok, %{user: user}} -> {:ok, user}
      {:error, :user, changeset, _} -> {:error, changeset}
    end
  end
end
```

Código Fonte 4. Accounts 4/4

6.1.2 Código do Usuário

Os códigos do arquivo `user.ex` “fala” pro Phoenix como que o usuário se comporta, as características dele e suas funções, como os campos da tabela do banco de dados que se refere a ele, se as informações que o cliente passou são validas, entre outros.

```elixir
defmodule RecipePerNote.Accounts.User do

  use Ecto.Schema
  import Ecto.Changeset
  alias  RecipePerNote.Annotations
  @derive {Inspect, except: [:password]}
  schema "users" do
    field :name, :string
    field :email, :string
    field :password, :string, virtual: true
    field :hashed_password, :string
    field :confirmed_at, :naive_datetime

    has_many :todos, Annotations.ToDo
    has_many :watchlater, Annotations.WatchLater
    has_many :notes, Annotations.Notes

    timestamps()
  end

  @doc """
  A user changeset for registration.

  It is important to validate the length of both email and password.
  Otherwise databases may truncate the email without warnings, which
  could lead to unpredictable or insecure behaviour. Long passwords may
  also be very expensive to hash for certain algorithms.

  ## Options

    * `:hash_password` - Hashes the password so it can be stored securely
      in the database and ensures the password field is cleared to prevent
      leaks in the logs. If password hashing is not needed and clearing the
      password field is not desired (like when using this changeset for
      validations on a LiveView form), this option can be set to `false`.
      Defaults to `true`.
  """
  def registration_changeset(user, attrs, opts \\ []) do
    user
    |> cast(attrs, [:name, :email, :password])
    |> validate_email()
    |> validate_password(opts)
  end

  defp validate_email(changeset) do
    changeset
    |> validate_required([:email])
    |> validate_format(:email, ~r/^[^\s]+@[^\s]+$/, message: "must have the @ sign and
    |> validate_length(:email, max: 160)
    |> unsafe_validate_unique(:email, RecipePerNote.Repo)
    |> unique_constraint(:email)
  end

  defp validate_password(changeset, opts) do
    changeset
    |> validate_required([:password])
    |> validate_length(:password, min: 12, max: 80)
    |> validate_format(:password, ~r/[a-z]/, message: "at least one lower case charact
    |> validate_format(:password, ~r/[A-Z]/, message: "at least one upper case charact
    |> validate_format(:password, ~r/[!?@#$%^&*_0-9]/, message: "at least one digit or
    |> maybe_hash_password(opts)
  end

  defp maybe_hash_password(changeset, opts) do
    hash_password? = Keyword.get(opts, :hash_password, true)
    password = get_change(changeset, :password)

    if hash_password? && password && changeset.valid? do
      changeset
      |> put_change(:hashed_password, Bcrypt.hash_pwd_salt(password))
      |> delete_change(:password)
    else
      changeset
    end
  end

  @doc """
  A user changeset for changing the email.

  It requires the email to change otherwise an error is added.
  """
  def email_changeset(user, attrs) do
    user
    |> cast(attrs, [:email])
    |> validate_email()
    |> case do
      %{changes: %{email: _}} = changeset -> changeset
      %{} = changeset -> add_error(changeset, :email, "did not change")
    end
  end
```

Código Fonte 5. Accounts.User 1/2

```elixir
  @doc """
  A user changeset for changing the password.

  ## Options

    * `:hash_password` - Hashes the password so it can be stored securely
      in the database and ensures the password field is cleared to prevent
      leaks in the logs. If password hashing is not needed and clearing the
      password field is not desired (like when using this changeset for
      validations on a LiveView form), this option can be set to `false`.
      Defaults to `true`.
  """
  def password_changeset(user, attrs, opts \\ []) do
    user
    |> cast(attrs, [:password])
    |> validate_confirmation(:password, message: "does not match password")
    |> validate_password(opts)
  end

  @doc """
  Confirms the account by setting `confirmed_at`.
  """
  def confirm_changeset(user) do
    now = NaiveDateTime.utc_now() |> NaiveDateTime.truncate(:second)
    change(user, confirmed_at: now)
  end

  @doc """
  Verifies the password.

  If there is no user or the user doesn't have a password, we call
  `Bcrypt.no_user_verify/0` to avoid timing attacks.
  """
  def valid_password?(%RecipePerNote.Accounts.User{hashed_password: hashed_password}, |
      when is_binary(hashed_password) and byte_size(password) > 0 do
    Bcrypt.verify_pass(password, hashed_password)
  end

  def valid_password?(_, _) do
    Bcrypt.no_user_verify()
    false
  end

  @doc """
  Validates the current password otherwise adds an error to the changeset.
  """
  def validate_current_password(changeset, password) do
    if valid_password?(changeset.data, password) do
      changeset
    else
      add_error(changeset, :current_password, "is not valid")
    end
  end
end
```

Código Fonte 6. Accounts.User 2/2

6.1.3 Contexto Annotations

Com o contexto Annotations foi trabalhado os códigos de conexão com o banco de dados referente as anotações, no caso os Notes, os ToDos e os WatchLaters.

```elixir
defmodule RecipePerNote.Annotations do
  @moduledoc """
  The Annotations context.
  """

  import Ecto.Query, warn: false
  alias RecipePerNote.Repo
  alias RecipePerNote.Annotations.ToDo

  @doc """
  Returns the list of todos.

  ## Examples

      iex> list_todos()
      [%ToDo{}, ...]

  """
  def list_todos(user) do
    ToDo
    |> where([t], t.user_id == ^user.id)
    |> preload([:user])
    |> Repo.all()
  end

  @doc """
  Returns the list of todos from given date

  ## Examples

      iex> list_todos_from_date()
      [?]
  """

  def list_todos_from_date(date) do
    ToDo
    |> where([t], t.datelimit == ^date)
    |> preload([:user])
    |> Repo.all()
  end

  @doc """
  Gets a single to_do.

  Raises `Ecto.NoResultsError` if the To do does not exist.

  ## Examples

      iex> get_to_do!(123)
      %ToDo{}

      iex> get_to_do!(456)
      ** (Ecto.NoResultsError)

  """
  def get_to_do!(id), do: Repo.get!(ToDo, id)

  @doc """
  Creates a to_do.

  ## Examples

      iex> create_to_do(%{field: value})
      {:ok, %ToDo{}}

      iex> create_to_do(%{field: bad_value})
      {:error, %Ecto.Changeset{}}

  """
  def create_to_do(attrs \\ %{}) do
    %ToDo{}
    |> ToDo.changeset(attrs)
    |> Repo.insert()
  end

  @doc """
  Updates a to_do.

  ## Examples

      iex> update_to_do(to_do, %{field: new_value})
      {:ok, %ToDo{}}

      iex> update_to_do(to_do, %{field: bad_value})
      {:error, %Ecto.Changeset{}}

  """
  def update_to_do(%ToDo{} = to_do, attrs) do
    to_do
    |> ToDo.changeset(attrs)
    |> Repo.update()
  end
```

Código Fonte 7. Annotations 1/4

```elixir
@doc """
Deletes a to_do.

## Examples

    iex> delete_to_do(to_do)
    {:ok, %ToDo{}}

    iex> delete_to_do(to_do)
    {:error, %Ecto.Changeset{}}

"""
def delete_to_do(%ToDo{} = to_do) do
  Repo.delete(to_do)
end

@doc """
Returns an `%Ecto.Changeset{}` for tracking to_do changes.

## Examples

    iex> change_to_do(to_do)
    %Ecto.Changeset{data: %ToDo{}}

"""
def change_to_do(%ToDo{} = to_do, attrs \\ %{}) do
  ToDo.changeset(to_do, attrs)
end

alias RecipePerNote.Annotations.WatchLater

@doc """
Returns the list of watchlater.

## Examples

    iex> list_watchlater()
    [%WatchLater{}, ...]

"""
def list_watchlater(user) do
  WatchLater
  |> where([w], w.user_id == ^user.id)
  |> preload([:user])
  |> Repo.all()
end

@doc """
Gets a single watch_later.

Raises `Ecto.NoResultsError` if the Watch later does not exist.

## Examples

    iex> get_watch_later!(123)
    %WatchLater{}

    iex> get_watch_later!(456)
    ** (Ecto.NoResultsError)

"""
def get_watch_later!(id), do: Repo.get!(WatchLater, id)

@doc """
Creates a watch_later.

## Examples

    iex> create_watch_later(%{field: value})
    {:ok, %WatchLater{}}

    iex> create_watch_later(%{field: bad_value})
    {:error, %Ecto.Changeset{}}

"""
def create_watch_later(attrs \\ %{}) do
  %WatchLater{}
  |> WatchLater.changeset(attrs)
  |> Repo.insert()
end

@doc """
Updates a watch_later.

## Examples

    iex> update_watch_later(watch_later, %{field: new_value})
    {:ok, %WatchLater{}}

    iex> update_watch_later(watch_later, %{field: bad_value})
    {:error, %Ecto.Changeset{}}

"""
```

Código Fonte 8. Annotations 2/4

```elixir
@doc """
Updates a watch_later.

## Examples

    iex> update_watch_later(watch_later, %{field: new_value})
    {:ok, %WatchLater{}}

    iex> update_watch_later(watch_later, %{field: bad_value})
    {:error, %Ecto.Changeset{}}

"""
def update_watch_later(%WatchLater{} = watch_later, attrs) do
  watch_later
  |> WatchLater.changeset(attrs)
  |> Repo.update()
end

@doc """
Deletes a watch_later.

## Examples

    iex> delete_watch_later(watch_later)
    {:ok, %WatchLater{}}

    iex> delete_watch_later(watch_later)
    {:error, %Ecto.Changeset{}}

"""
def delete_watch_later(%WatchLater{} = watch_later) do
  Repo.delete(watch_later)
end

@doc """
Returns an `%Ecto.Changeset{}` for tracking watch_later changes.

## Examples

    iex> change_watch_later(watch_later)
    %Ecto.Changeset{data: %WatchLater{}}

"""
def change_watch_later(%WatchLater{} = watch_later, attrs \\ %{}) do
  WatchLater.changeset(watch_later, attrs)
end

alias RecipePerNote.Annotations.Notes

@doc """
Returns the list of notes.

## Examples

    iex> list_notes()
    [%Notes{}, ...]

"""
def list_notes(user) do
  Notes
  |> where([n], n.user_id == ^user.id)
  |> preload([:user])
  |> Repo.all()
end

@doc """
Gets a single notes.

Raises `Ecto.NoResultsError` if the Notes does not exist.

## Examples

    iex> get_notes!(123)
    %Notes{}

    iex> get_notes!(456)
    ** (Ecto.NoResultsError)

"""
def get_notes!(id), do: Repo.get!(Notes, id)

@doc """
Creates a notes.

## Examples

    iex> create_notes(%{field: value})
    {:ok, %Notes{}}

    iex> create_notes(%{field: bad_value})
    {:error, %Ecto.Changeset{}}

"""
```

Código Fonte 9. Annotations 3/4

```
  def create_notes(attrs \\ %{}) do
    %Notes{}
    |> Notes.changeset(attrs)
    |> Repo.insert()
  end

  @doc """
  Updates a notes.

  ## Examples

      iex> update_notes(notes, %{field: new_value})
      {:ok, %Notes{}}

      iex> update_notes(notes, %{field: bad_value})
      {:error, %Ecto.Changeset{}}

  """
  def update_notes(%Notes{} = notes, attrs) do
    notes
    |> Notes.changeset(attrs)
    |> Repo.update()
  end

  @doc """
  Deletes a notes.

  ## Examples

      iex> delete_notes(notes)
      {:ok, %Notes{}}

      iex> delete_notes(notes)
      {:error, %Ecto.Changeset{}}

  """
  def delete_notes(%Notes{} = notes) do
    Repo.delete(notes)
  end

  @doc """
  Returns an `%Ecto.Changeset{}` for tracking notes changes.

  ## Examples

      iex> change_notes(notes)
      %Ecto.Changeset{data: %Notes{}}

  """
  def change_notes(%Notes{} = notes, attrs \\ %{}) do
    Notes.changeset(notes, attrs)
  end

end
```

Código Fonte 10. Annotations 4/4

6.1.4 Código do Notes

Os códigos do arquivo `notes.ex` “fala” pro Phoenix como que o usuário se comporta, as características dele e suas funções, como os campos da tabela do banco de dados que se refere a ele, se as informações que o cliente passou são validas, entre outros.

```elixir
defmodule RecipePerNote.Annotations.Notes do
  use Ecto.Schema
  import Ecto.Changeset
  alias RecipePerNote.Accounts.User

  schema "notes" do
    field :descri_notes, :string
    field :title, :string
    belongs_to :user, User

    timestamps()
  end

  @doc false
  def changeset(notes, attrs) do
    notes
    |> cast(attrs, [:title, :descri_notes, :user_id])
    |> validate_required([:title, :descri_notes, :user_id])
  end
end
```

Código Fonte 11. Annotations.Notes

6.1.5 Código do ToDos

Os códigos do arquivo `to_do.ex` “fala” pro Phoenix como que o usuário se comporta, as características dele e suas funções, como os campos da tabela do banco de dados que se refere a ele, se as informações que o cliente passou são validas, entre outros.

```elixir
defmodule RecipePerNote.Annotations.ToDo do
  use Ecto.Schema
  import Ecto.Changeset
  alias RecipePerNote.Accounts.User

  schema "todos" do
    field :datelimit, :date
    field :descri, :string
    field :titles, :string
    belongs_to :user, User

    timestamps()
  end

  @doc false
  def changeset(to_do, attrs) do
    to_do
    |> cast(attrs, [:titles, :datelimit, :descri, :user_id])
    |> validate_required([:titles, :datelimit, :descri, :user_id])
  end
end
```

Código Fonte 12. Annotations.ToDo

6.1.6 Código do WatchLater

Os códigos do arquivo `watch_later.ex` “fala” pro Phoenix como que o usuário se comporta, as características dele e suas funções, como os campos da tabela do banco de dados que se refere a ele, se as informações que o cliente passou são validas, entre outros.

```elixir
defmodule RecipePerNote.Annotations.WatchLater do
  use Ecto.Schema
  import Ecto.Changeset
  alias RecipePerNote.Accounts.User

  schema "watchlater" do
    field :link, :string
    field :title, :string
    belongs_to :user, User

    timestamps()
  end

  @doc false
  def changeset(watch_later, attrs) do
    watch_later
    |> cast(attrs, [:title, :link, :user_id])
    |> validate_required([:title, :link, :user_id])
  end
end
```

Código Fonte 13. Annotations.WatchLater

6.1.7 Banco de Dados

No Elixir tem a biblioteca Ecto para se trabalhar com o banco de dados, onde o PostgreSQL é o adapter padrão. É importante ressaltar que o Ecto é dividido em quatro partes, com Oliveira (2019) fala em seu artigo.

O ´Ecto.Repo´ é responsável pelas operações no banco de dados. Via repositório é possível criar, atualizar, deletar e realizar queries.

O ´Ecto.Schema´ é utilizado para mapear as tabelas do banco de dados dentro da estrutura do Elixir.

O ´Ecto.Changeset´ é um meio onde os desenvolvedores poderão fazer os filtros, validações e os casts antes de atualizar ou adicionar uma informação ao banco de dados.

E o ´Ecto.Query´ é escrita na sintaxe Elixir, as queries são usadas para recuperar informações do banco de dados. É bom lembrar que as Queries no Ecto são seguras, evitando problemas como SQL Injection.

Conhecendo melhor o Ecto, ao rever os códigos apresentados acima, é possível compreender melhor onde que fora utilizado e consequentemente, entender melhor o código.

6.1.7.1 MER

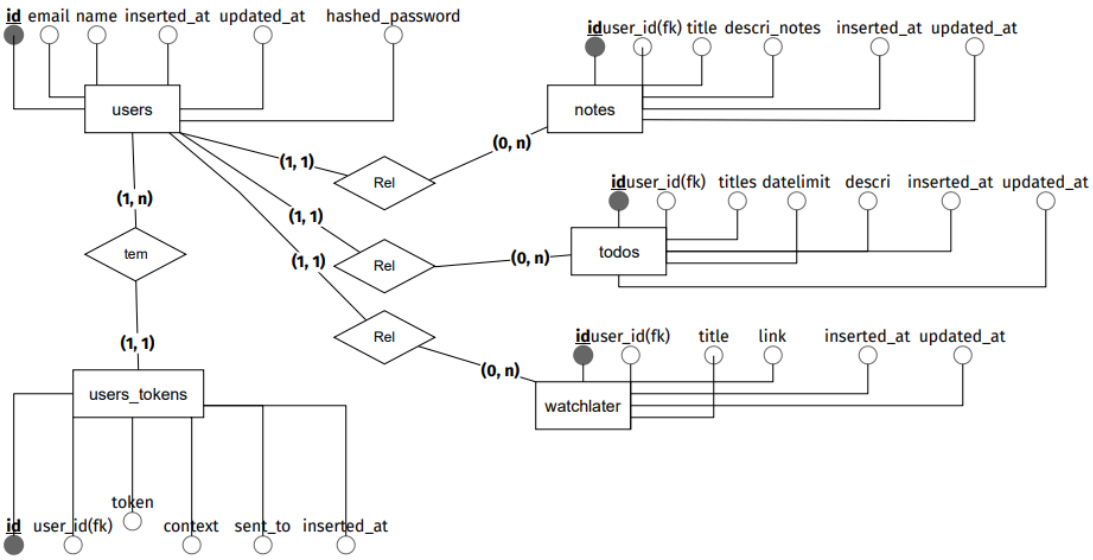
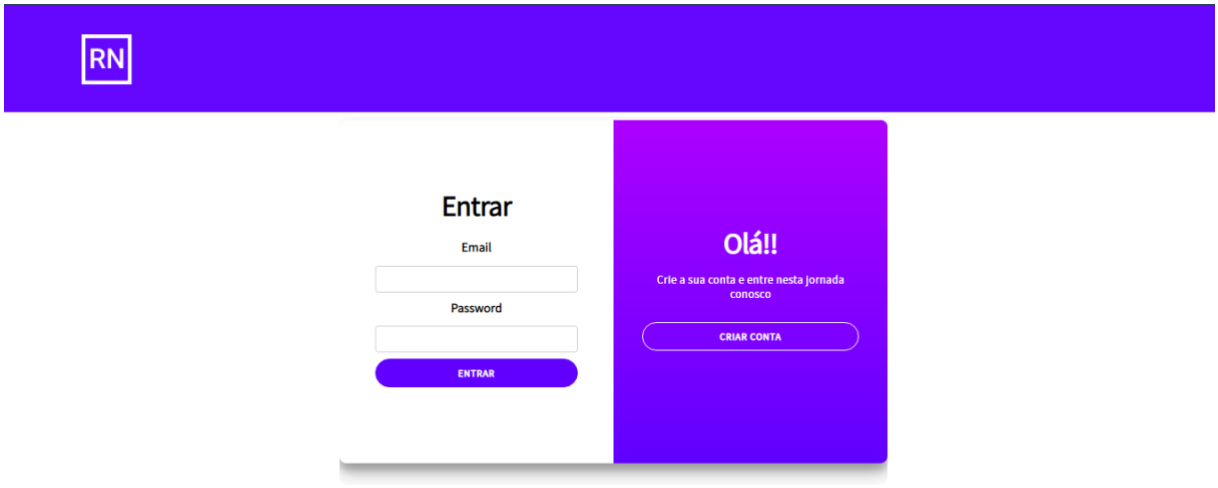


Figura 11. MER

6.2 Front end

6.2.1 Login

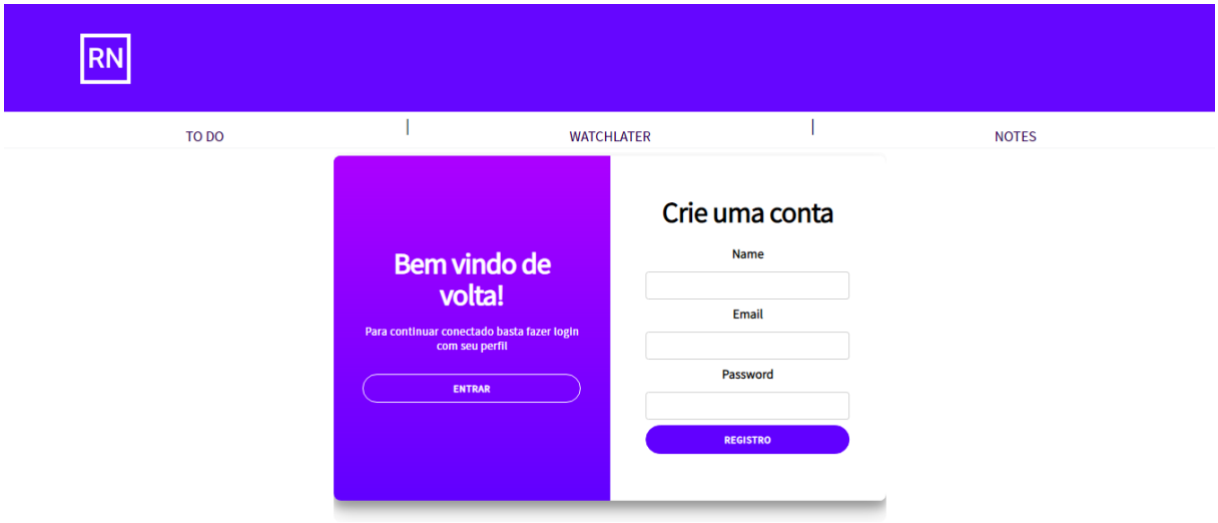
Ao entrar no site o usuário se encontra com a página de login.



Tela 3. Login

6.2.2 Registro

A página de registro também apresenta um formulário para efetuar a criação de uma conta para o usuário. Utilizando nome, e-mail e senha.



Tela 4. Registro

6.2.2.1 Código do User Session Controller

```elixir
defmodule RecipePerNoteWeb.UserSessionController do      SaLandin, 3 months ago • addi
  use RecipePerNoteWeb, :controller

  alias RecipePerNote.Accounts
  alias RecipePerNote.Accounts.User
  alias RecipePerNoteWeb.UserAuth

  def new(conn, _params) do
    changeset = Accounts.change_user_registration(%User{}, %{})

    render(conn, "new.html", error_message: nil, changeset: changeset)
  end

  def create(conn, %{"user" => user_params}) do
    %{"email" => email, "password" => password} = user_params
    changeset = Accounts.change_user_registration(%User{}, %{})

    if user = Accounts.get_user_by_email_and_password(email, password) do
      UserAuth.log_in_user(conn, user, user_params)
    else
      render(conn, "new.html", error_message: "Invalid email or password", changeset:
    end
  end

  def delete(conn, _params) do
    conn
    |> put_flash(:info, "Logged out successfully.")
    |> UserAuth.log_out_user()
  end
end
```

Código Fonte 14. UserSessionController

No código do controle de seção do usuário é apresentado funções para lidar com a criação de um usuário, com o login de um usuário e com o logout do usuário.

6.2.2.2 Codigo user_auth.ex

O user_auth serve pro servidor Phoenix autenticar a sessão de um usuário.

```elixir
defmodule RecipePerNoteWeb.UserAuth do
  import Plug.Conn
  import Phoenix.Controller

  alias RecipePerNote.Accounts
  alias RecipePerNoteWeb.Router.Helpers, as: Routes

  # Make the remember me cookie valid for 60 days.
  # If you want bump or reduce this value, also change
  # the token expiry itself in UserToken.
  @max_age 60 * 60 * 24 * 60
  @remember_me_cookie "_recipe_per_note_web_user_remember_me"
  @remember_me_options [sign: true, max_age: @max_age, same_site: "Lax"]

  @doc """
  Logs the user in.

  It renews the session ID and clears the whole session
  to avoid fixation attacks. See the renew_session
  function to customize this behaviour.

  It also sets a `:live_socket_id` key in the session,
  so LiveView sessions are identified and automatically
  disconnected on log out. The line can be safely removed
  if you are not using LiveView.
  """
  def log_in_user(conn, user, params \\ %{}) do
    token = Accounts.generate_user_session_token(user)
    user_return_to = get_session(conn, :user_return_to)

    conn
    |> renew_session()
    |> put_session(:user_token, token)
    |> put_session(:live_socket_id, "users_sessions:#{Base.url_encode64(token)}")
    |> maybe_write_remember_me_cookie(token, params)
    |> redirect(to: user_return_to || signed_in_path(conn))
  end

  defp maybe_write_remember_me_cookie(conn, token, %{"remember_me" => "true"}) do
    put_resp_cookie(conn, @remember_me_cookie, token, @remember_me_options)
  end

  defp maybe_write_remember_me_cookie(conn, _token, _params) do
    conn
  end

  # This function renews the session ID and erases the whole
  # session to avoid fixation attacks. If there is any data
  # in the session you may want to preserve after log in/log out,
  # you must explicitly fetch the session data before clearing
  # and then immediately set it after clearing, for example:
  #
  #     defp renew_session(conn) do
  #       preferred_locale = get_session(conn, :preferred_locale)
  #
  #       conn
  #       |> configure_session(renew: true)
  #       |> clear_session()
  #       |> put_session(:preferred_locale, preferred_locale)
  #     end
  #
  defp renew_session(conn) do
    conn
    |> configure_session(renew: true)
    |> clear_session()
  end

  @doc """
  Logs the user out.

  It clears all session data for safety. See renew_session.
  """
  def log_out_user(conn) do
    user_token = get_session(conn, :user_token)
    user_token && Accounts.delete_session_token(user_token)

    if live_socket_id = get_session(conn, :live_socket_id) do
      RecipePerNoteWeb.Endpoint.broadcast(live_socket_id, "disconnect", %{})
    end

    conn
    |> renew_session()
    |> delete_resp_cookie(@remember_me_cookie)
    |> redirect(to: "/")
  end
```

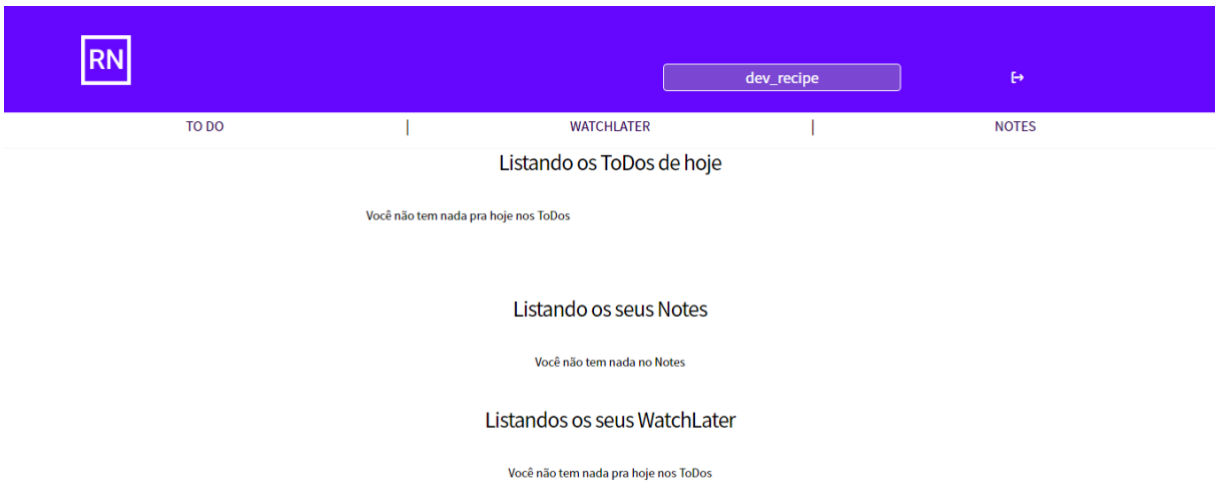
Código Fonte 15. UserAuth 1/2

```
  @doc """
  Authenticates the user by looking into the session
  and remember me token.
  """
  def fetch_current_user(conn, _opts) do
    {user_token, conn} = ensure_user_token(conn)
    user = user_token && Accounts.get_user_by_session_token(user_token)
    assign(conn, :current_user, user)
  end

  defp ensure_user_token(conn) do
    if user_token = get_session(conn, :user_token) do
      {user_token, conn}
    else
      conn = fetch_cookies(conn, signed: [@remember_me_cookie])

      if user_token = conn.cookies[@remember_me_cookie] do
        {user_token, put_session(conn, :user_token, user_token)}
      else
        {nil, conn}
      end
    end
  end

  @doc """
  Used for routes that require the user to not be authenticated.
  """
  def redirect_if_user_is_authenticated(conn, _opts) do
    if conn.assigns[:current_user] do
      conn
      |> redirect(to: signed_in_path(conn))
      |> halt()
    else
      conn
    end
  end

  @doc """
  Used for routes that require the user to be authenticated.

  If you want to enforce the user email is confirmed before
  they use the application at all, here would be a good place.
  """
  def require_authenticated_user(conn, _opts) do
    if conn.assigns[:current_user] do
      conn
    else
      conn
      |> put_flash(:error, "You must log in to access this page.")
      |> maybe_store_return_to()
      |> redirect(to: Routes.user_session_path(conn, :new))
      |> halt()
    end
  end

  defp maybe_store_return_to(%{method: "GET"} = conn) do
    put_session(conn, :user_return_to, current_path(conn))
  end

  defp maybe_store_return_to(conn), do: conn

  defp signed_in_path(_conn), do: "/"
end
```

Código Fonte 16. UserAuth 2/2

6.2.3 Home

Feito o Login ou o Registro de um Usuário, é apresentado a página “home”, que irá listar os ToDos do dia, os Notes e os WatchLater. Estando uma vez logado, será possível navegar pela aplicação.



Tela 5. Home

6.2.3.1 Código do Controller do home

```elixir
defmodule RecipePerNoteWeb.HomeController do
  use RecipePerNoteWeb, :controller

  alias RecipePerNote.Annotations
  alias RecipePerNote.Accounts.User
  alias RecipePerNote.ComparesTime.Time

  def index(conn, _params) do
    todos = todos_have_something?(list_todos(Date.utc_today))

    notes = notes_have_something?(list_notes(conn))

    watchlater = watchlater_have_something?(list_watchlater(conn))

    render(conn, "home.html", todos: todos, notes: notes, watchlater: watchlater)
  end

  defp list_todos(date) do
    Annotations.list_todos_from_date(date)
  end

  defp todos_have_something?(todos) do
    if todos == [] do
      :nada
    else
      todos
    end
  end

  defp list_notes(conn) do
    Annotations.list_notes(conn.assigns.current_user)
  end

  defp notes_have_something?(notes) do
    if notes == [] do
      :nada
    else
      notes
    end
  end

  defp list_watchlater(conn) do
    Annotations.list_watchlater(conn.assigns.current_user)
  end

  defp watchlater_have_something?(watchlater) do
    if watchlater == [] do
      :nada
    else
      watchlater
    end
  end

end
```

Código Fonte 17. HomeController

Com ele listamos o Notes, ToDos e o WatchLater, e ele vê se o usuário tem algo ou não.

6.2.4 Mapa do site

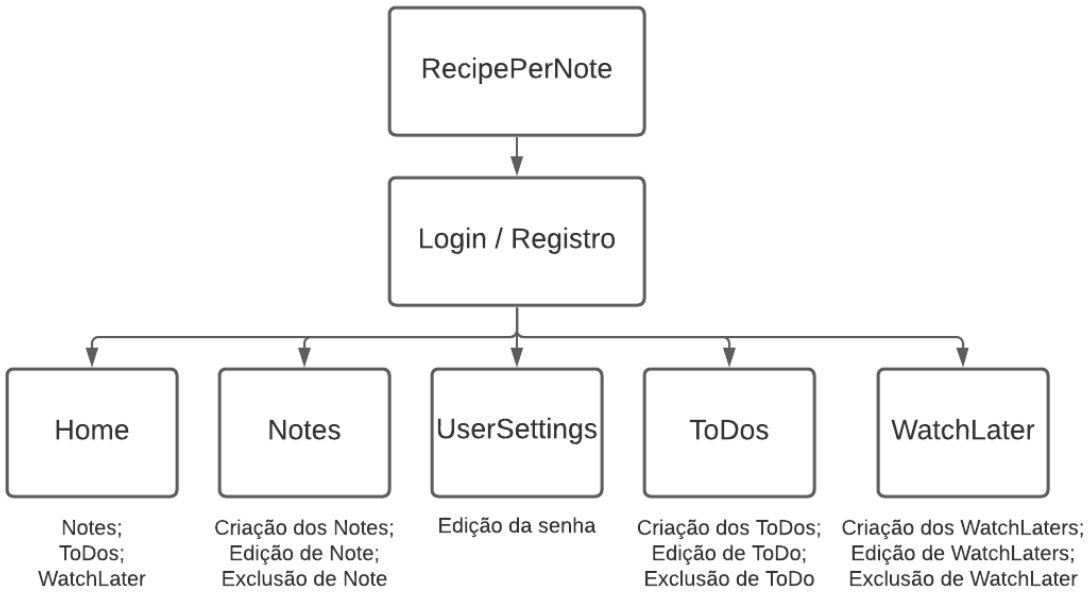
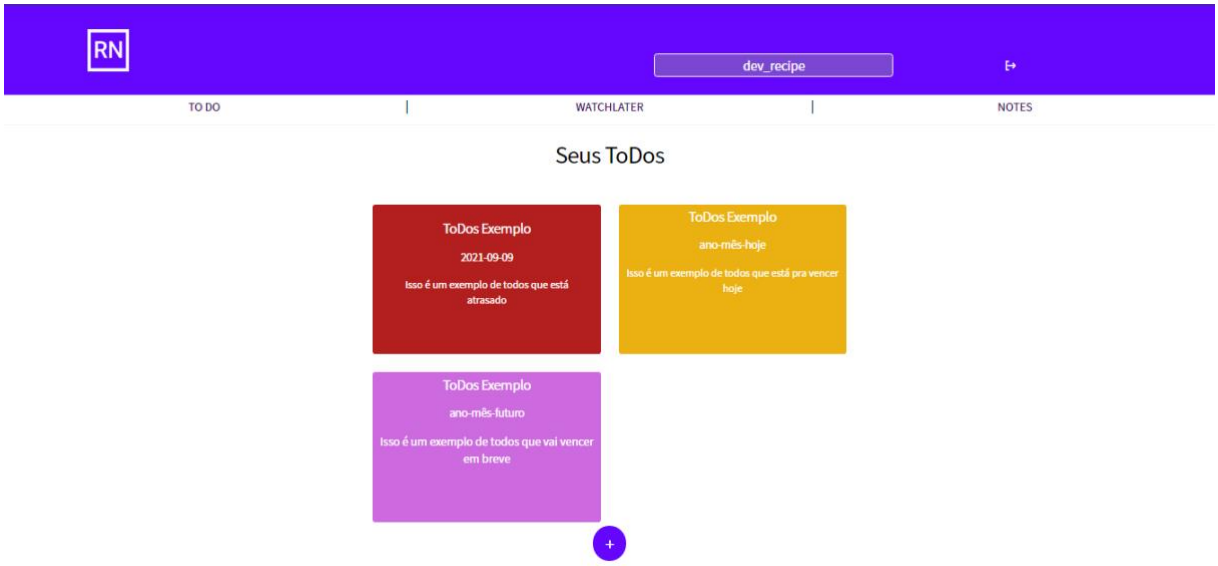


Figura 12. Mapa do site

6.2.5 ToDo

A função da página ToDo será para criação de ToDos, que servem de lembretes de tarefas anexadas com prazos limites. Variando sua coloração com a aproximação do prazo final.



Tela 6. ToDo

6.2.5.1 Código para verificar o prazo do ToDo

```elixir
defmodule RecipePerNote.ComparesTime.Time do
  require Math

  defp module_function(value) do
    value
    |> Math.pow(2)
    |> Math.sqrt
    |> round
  end

  def return_date do
    date_test = Date.utc_today()

    IO.puts(date_test)
  end

  def is_passed_the_limit_date_text?(limit, date) do
    diff = Date.diff(limit, date)

    case diff do
      diff when diff < 0 ->
        "Faz #{diff |> module_function} dias que passou"

      diff when diff == 0 ->
        "Acaba hoje o prazo"

      diff when diff > 0 ->
        "Falta #{diff} dias pra acabar o prazo"

      _ ->
        "Amém"
    end
  end

  def is_passed_the_limit_date_class?(limit, date) do
    diff = Date.diff(limit, date)

    case diff do
      diff when diff < 0 ->
        "box_late"

      diff when diff == 0 ->
        "box_now"

      diff when diff > 0 ->
        "box_havetime"

      _ ->
        "Amém"
    end
  end
end
```

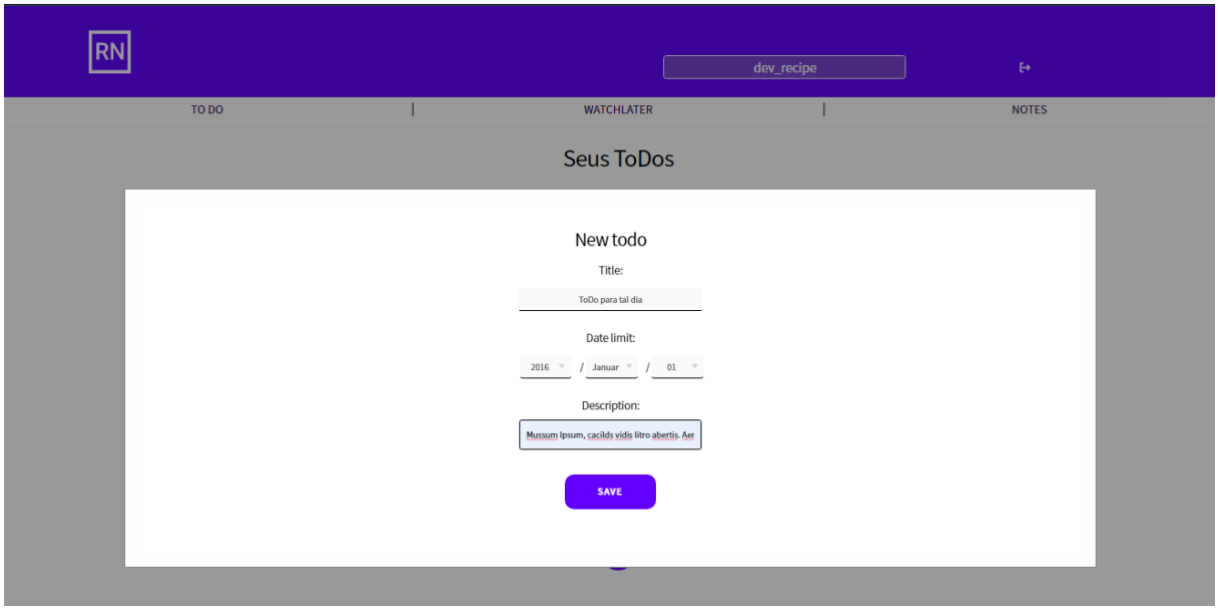
Código Fonte 18. ComparesTime

6.2.5.2 Codigo ToDo Live

```elixir
defmodule RecipePerNoteWeb.ToDoLive.Index do
  use RecipePerNoteWeb, :live_view

  alias RecipePerNote.ComparesTime.Time
  alias RecipePerNote.Annotations
  alias RecipePerNote.Annotations.ToDo

  @impl true
  def mount(_params, session, socket) do
    socket =
      assign_defaults(session, socket)

    todos =
      socket
      |> list_todos
      |> is_todos_nil?

    IO.inspect(todos)
    {:ok, assign(socket, :todos, todos)}
  end

  @impl true
  def handle_params(params, _url, socket) do
    {:noreply, apply_action(socket, socket.assigns.live_action, params)}
  end

  defp apply_action(socket, :edit, %{"id" => id}) do
    socket
    |> assign(:page_title, "Edit To do")
    |> assign(:to_do, Annotations.get_to_do!(id))
  end

  defp apply_action(socket, :new, _params) do
    socket
    |> assign(:page_title, "New To do")
    |> assign(:to_do, %ToDo{})
  end

  defp apply_action(socket, :index, _params) do
    socket
    |> assign(:page_title, "Listing Todos")
    |> assign(:to_do, nil)
  end

  @impl true
  def handle_event("delete", %{"id" => id}, socket) do
    to_do = Annotations.get_to_do!(id)
    {:ok, _} = Annotations.delete_to_do(to_do)

    {:noreply, assign(socket, :todos, list_todos(socket))}
  end

  defp is_todos_nil?(todos) do
    if todos == [] do
      :nada
    else
      todos
    end
  end

  defp list_todos(socket) do
    Annotations.list_todos(socket.assigns[:current_user])
  end
end
```

Código Fonte 19. ToDoLive.Index

6.2.6 ToDo Formulário



Tela 7. ToDo Form

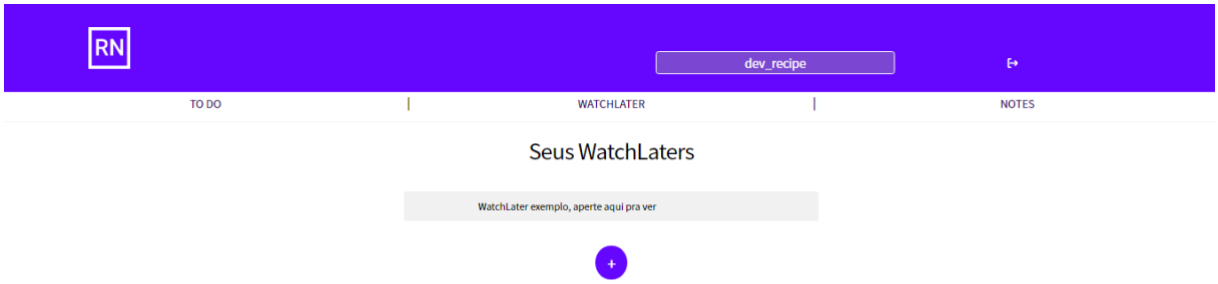
6.2.6.1 Codigo ToDo Live Form

```elixir
defmodule RecipePerNoteWeb.ToDoLive.FormComponent do
  use RecipePerNoteWeb, :live_component

  alias RecipePerNote.Annotations

  @impl true
  def update(%{to_do: to_do} = assigns, socket) do
    changeset = Annotations.change_to_do(to_do)

    {:ok,
     socket
     |> assign(assigns)
     |> assign(:changeset, changeset)}
  end

  @impl true
  def handle_event("validate", %{"to_do" => to_do_params}, socket) do
    changeset =
      socket.assigns.to_do
      |> Annotations.change_to_do(to_do_params)
      |> Map.put(:action, :validate)

    {:noreply, assign(socket, :changeset, changeset)}
  end

  def handle_event("save", %{"to_do" => to_do_params}, socket) do
    save_to_do(socket, socket.assigns.action, to_do_params)
  end

  defp save_to_do(socket, :edit, to_do_params) do
    case Annotations.update_to_do(socket.assigns.to_do, to_do_params) do
      {:ok, _to_do} ->
        {:noreply,
         socket
         |> put_flash(:info, "To do updated successfully")
         |> push_redirect(to: socket.assigns.return_to)}

      {:error, %Ecto.Changeset{} = changeset} ->
        {:noreply, assign(socket, :changeset, changeset)}
    end
  end

  defp save_to_do(socket, :new, to_do_params) do
    case Annotations.create_to_do(to_do_params) do
      {:ok, _to_do} ->
        {:noreply,
         socket
         |> put_flash(:info, "To do created successfully")
         |> push_redirect(to: socket.assigns.return_to)}

      {:error, %Ecto.Changeset{} = changeset} ->
        {:noreply, assign(socket, changeset: changeset)}
    end
  end
end
```

Código Fonte 20. ToDoLive.FormComponent

6.2.7 Watch Later

Tem como função a página Watch Later a gravação de links de vídeos e sites. Assim podendo o usuário visualizar o conteúdo da página posteriormente.



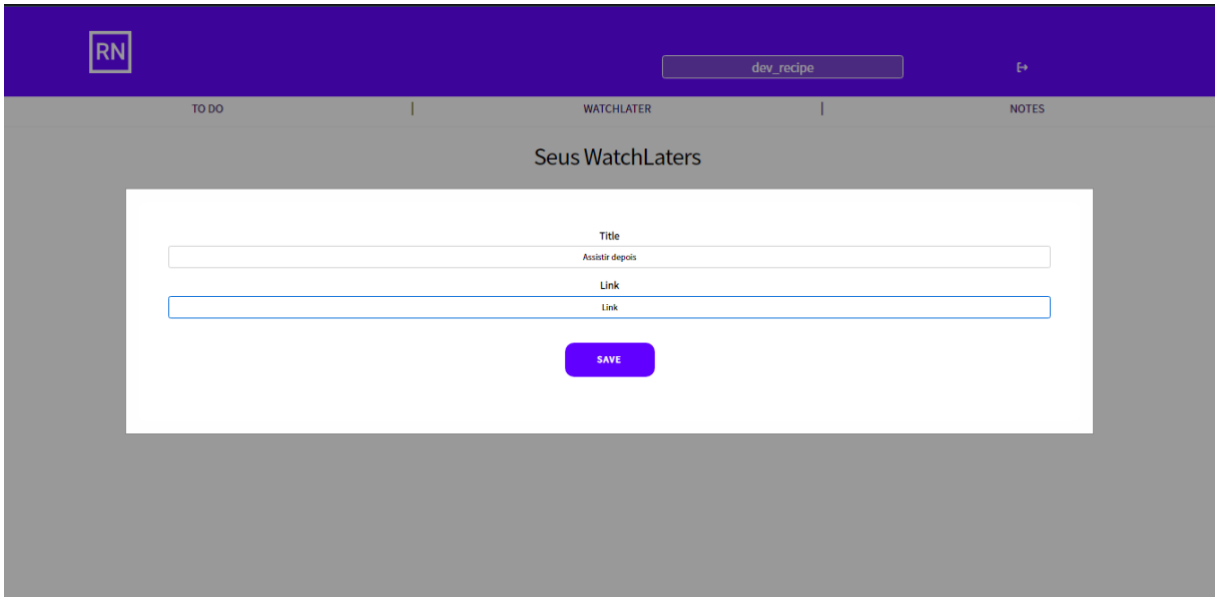
Tela 8. WatchLater

6.2.7.1 Codigo Watch Later Live

```elixir
defmodule RecipePerNoteWeb.WatchLaterLive.Index do
  use RecipePerNoteWeb, :live_view

  alias RecipePerNote.Annotations
  alias RecipePerNote.Annotations.WatchLater

  @impl true
  def mount(_params, session, socket) do
    socket =
      assign_defaults(session, socket)

   watchlater =
      socket
        |> list_watchlater
        |> is_watchlater_nil?

    {:ok, assign(socket, :watchlater, watchlater)}
  end

  @impl true
  def handle_params(params, _url, socket) do
    {:noreply, apply_action(socket, socket.assigns.live_action, params)}
  end

  defp apply_action(socket, :edit, %{"id" => id}) do
    socket
    |> assign(:page_title, "Edit Watch later")
    |> assign(:watch_later, Annotations.get_watch_later!(id))
  end

  defp apply_action(socket, :new, _params) do
    socket
    |> assign(:page_title, "New Watch later")
    |> assign(:watch_later, %WatchLater{})
  end

  defp apply_action(socket, :index, _params) do
    socket
    |> assign(:page_title, "Listing Watchlater")
    |> assign(:watch_later, nil)
  end

  @impl true
  def handle_event("delete", %{"id" => id}, socket) do
    watch_later = Annotations.get_watch_later!(id)
    {:ok, _} = Annotations.delete_watch_later(watch_later)

    {:noreply, assign(socket, :watchlater, list_watchlater(socket))}
  end

  defp is_watchlater_nil?(watchlater) do
    if watchlater == [] do
      :nada
    else
      watchlater
    end
  end

  defp list_watchlater(socket) do
    Annotations.list_watchlater(socket.assigns[:current_user])
  end
end
```

Código Fonte 21. WatchLaterLive.Index

6.2.8 Watch Later Formulário



Tela 9. WatchLater Form

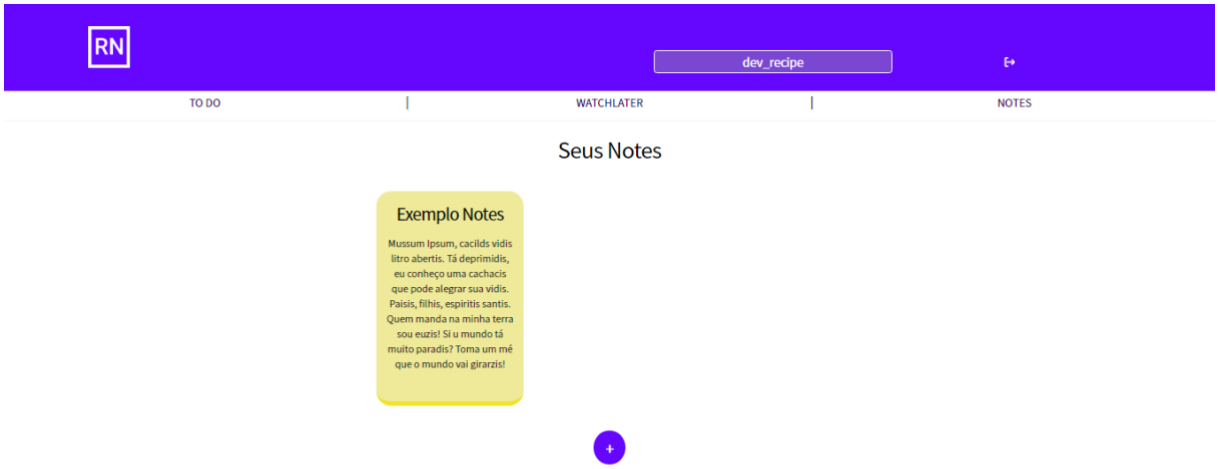
6.2.8.1 Codigo Watch Later Live Form

```elixir
defmodule RecipePerNoteWeb.WatchLaterLive.FormComponent do
  use RecipePerNoteWeb, :live_component

  alias RecipePerNote.Annotations

  @impl true
  def update(%{watch_later: watch_later} = assigns, socket) do
    changeset = Annotations.change_watch_later(watch_later)

    {:ok,
     socket
     |> assign(assigns)
     |> assign(:changeset, changeset)}
  end

  @impl true
  def handle_event("validate", %{"watch_later" => watch_later_params}, socket) do
    changeset =
      socket.assigns.watch_later
      |> Annotations.change_watch_later(watch_later_params)
      |> Map.put(:action, :validate)

    {:noreply, assign(socket, :changeset, changeset)}
  end

  def handle_event("save", %{"watch_later" => watch_later_params}, socket) do
    save_watch_later(socket, socket.assigns.action, watch_later_params)
  end

  defp save_watch_later(socket, :edit, watch_later_params) do
    case Annotations.update_watch_later(socket.assigns.watch_later, watch_later_params
      {:ok, _watch_later} ->
        {:noreply,
         socket
         |> put_flash(:info, "Watch later updated successfully")
         |> push_redirect(to: socket.assigns.return_to)}

      {:error, %Ecto.Changeset{} = changeset} ->
        {:noreply, assign(socket, :changeset, changeset)}
    end
  end

  defp save_watch_later(socket, :new, watch_later_params) do
    case Annotations.create_watch_later(watch_later_params) do
      {:ok, _watch_later} ->
        {:noreply,
         socket
         |> put_flash(:info, "Watch later created successfully")
         |> push_redirect(to: socket.assigns.return_to)}

      {:error, %Ecto.Changeset{} = changeset} ->
        {:noreply, assign(socket, changeset: changeset)}
    end
  end
end
```

Código Fonte 22. WatchLaterLive.FormComponent

6.2.9 Notes

A função da página Notes é permitir o usuário anotar lembretes para visualização posterior.



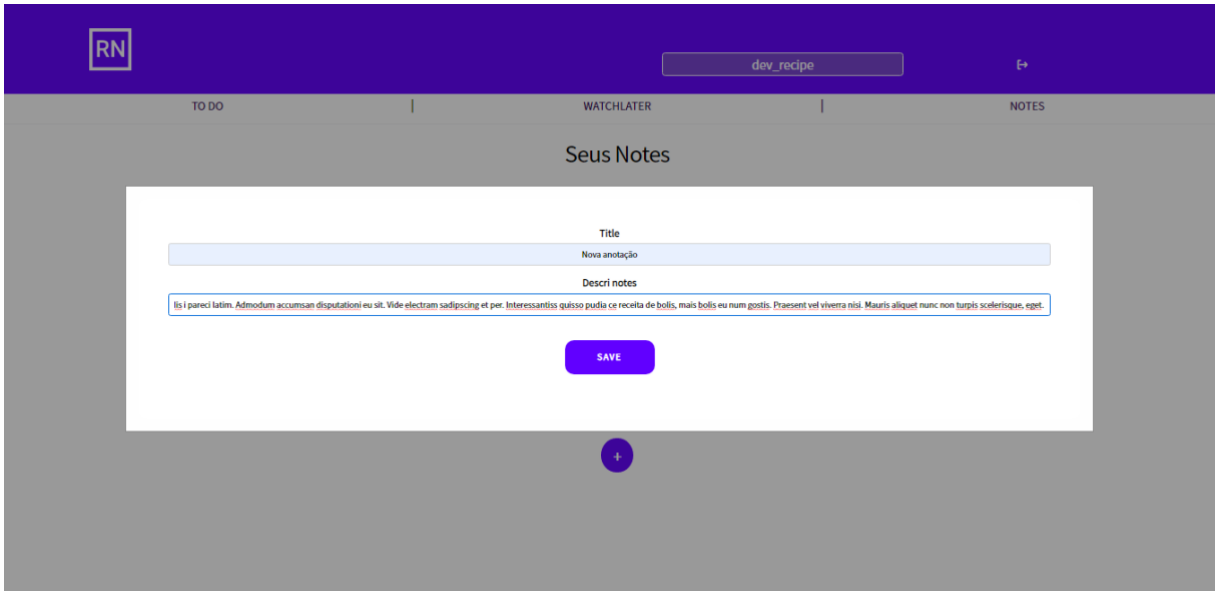
Tela 10. Notes

6.2.9.1 Codigo Notes Live

```elixir
defmodule RecipePerNoteWeb.NotesLive.Index do
  use RecipePerNoteWeb, :live_view

  alias RecipePerNote.Annotations
  alias RecipePerNote.Annotations.Notes

  @impl true
  def mount(_params, session, socket) do
    socket =
      assign_defaults(session, socket)

    notes =
      socket
      |> list_notes
      |> is_notes_nil?

    {:ok, assign(socket, :notes, notes)}
  end

  @impl true
  def handle_params(params, _url, socket) do
    {:noreply, apply_action(socket, socket.assigns.live_action, param
  end

  defp apply_action(socket, :edit, %{"id" => id}) do
    socket
    |> assign(:page_title, "Edit Notes")
    |> assign(:note, Annotations.get_notes!(id))
  end

  defp apply_action(socket, :new, _params) do
    socket
    |> assign(:page_title, "New Notes")
    |> assign(:note, %Notes{})
  end

  defp apply_action(socket, :index, _params) do
    socket
    |> assign(:page_title, "Listing notes")
    |> assign(:note, nil)
  end

  @impl true
  def handle_event("delete", %{"id" => id}, socket) do
    notes = Annotations.get_notes!(id)
    {:ok, _} = Annotations.delete_notes(notes)

    {:noreply, assign(socket, :notes, list_notes(socket))}
  end

  defp is_notes_nil?(notes) do
    if notes == [] do
      :nada
    else
      notes
    end
  end

  defp list_notes(socket) do
    Annotations.list_notes(socket.assigns[:current_user])
  end
end
```

Código Fonte 23. NotesLive.Index

6.2.10 Notes Formulário



Tela 11. Notes Form

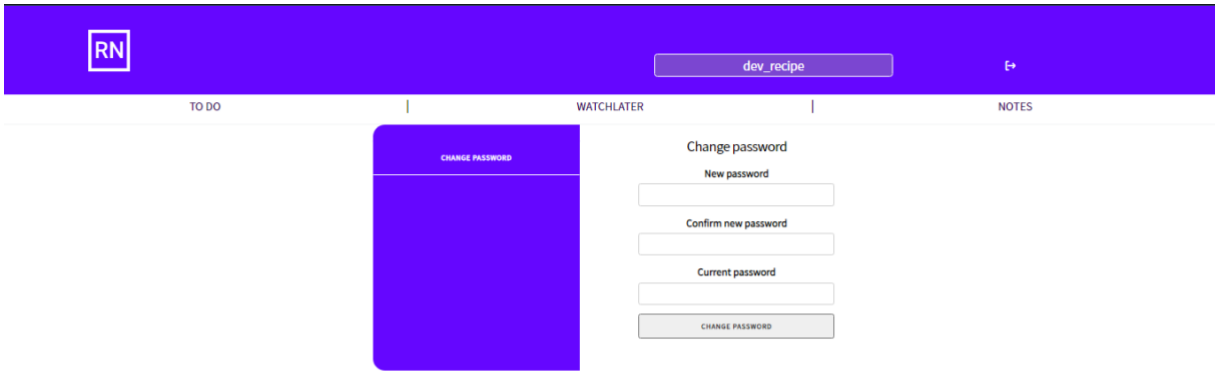
6.2.10.1 Codigo Notes Live Form

```elixir
defmodule RecipePerNoteWeb.NotesLive.FormComponent do
  use RecipePerNoteWeb, :live_component

  alias RecipePerNote.Annotations

  @impl true
  def update(%{notes: notes} = assigns, socket) do
    changeset = Annotations.change_notes(notes)

    {:ok,
     socket
     |> assign(assigns)
     |> assign(:changeset, changeset)}
  end

  @impl true
  def handle_event("validate", %{"notes" => notes_params}, socket) do
    changeset =
      socket.assigns.notes
      |> Annotations.change_notes(notes_params)
      |> Map.put(:action, :validate)

    {:noreply, assign(socket, :changeset, changeset)}
  end

  def handle_event("save", %{"notes" => notes_params}, socket) do
    save_notes(socket, socket.assigns.action, notes_params)
  end

  defp save_notes(socket, :edit, notes_params) do
    case Annotations.update_notes(socket.assigns.notes, notes_params) do
      {:ok, _notes} ->
        {:noreply,
         socket
         |> put_flash(:info, "Watch later updated successfully")
         |> push_redirect(to: socket.assigns.return_to)}

      {:error, %Ecto.Changeset{} = changeset} ->
        {:noreply, assign(socket, :changeset, changeset)}
    end
  end

  defp save_notes(socket, :new, notes_params) do
    case Annotations.create_notes(notes_params) do
      {:ok, _notes} ->
        {:noreply,
         socket
         |> put_flash(:info, "Watch later created successfully")
         |> push_redirect(to: socket.assigns.return_to)}

      {:error, %Ecto.Changeset{} = changeset} ->
        {:noreply, assign(socket, changeset: changeset)}
    end
  end
end
```

Código Fonte 24. NotesLive.FormComponent

6.2.11 User Settings

Esta página concede a troca de senha pelo usuário caso tenha perdido a anterior.



Tela 12. User Settings

6.2.11.1 Codigo User Settings Controller

```elixir
defmodule RecipePerNoteWeb.UserSettingsController do
  use RecipePerNoteWeb, :controller

  alias RecipePerNote.Accounts
  alias RecipePerNoteWeb.UserAuth

  plug :assign_email_and_password_changesets

  def edit(conn, _params) do
    render(conn, "edit.html")
  end

  def update(conn, %{"action" => "update_password"} = params) do
    %{"current_password" => password, "user" => user_params} = params
    user = conn.assigns.current_user

    case Accounts.update_user_password(user, password, user_params) do
      {:ok, user} ->
        conn
        |> put_flash(:info, "Password updated successfully.")
        |> put_session(:user_return_to, Routes.user_settings_path(conn, :edit))
        |> UserAuth.log_in_user(user)

      {:error, changeset} ->
        render(conn, "edit.html", password_changeset: changeset)
    end
  end

  def confirm_email(conn, %{"token" => token}) do
    case Accounts.update_user_email(conn.assigns.current_user, token) do
      :ok ->
        conn
        |> put_flash(:info, "Email changed successfully.")
        |> redirect(to: Routes.user_settings_path(conn, :edit))

      :error ->
        conn
        |> put_flash(:error, "Email change link is invalid or it has expired.")
        |> redirect(to: Routes.user_settings_path(conn, :edit))
    end
  end

  defp assign_email_and_password_changesets(conn, _opts) do
    user = conn.assigns.current_user

    conn
    |> assign(:email_changeset, Accounts.change_user_email(user))
    |> assign(:password_changeset, Accounts.change_user_password(user))
  end
end
```

Código Fonte 25. UserSettingsController

7 Conclusões finais

Este trabalho possibilitou a melhora da organização pessoal do indivíduo através de ferramentas de anotações vinculas a um site web. Graças a isso, pôde-se perceber a ajuda e necessidade sobre a organização diária de cada pessoa.

Para criação do site e seu foco, definiram-se três objetivos específicos. O primeiro objetivo buscava a criação dos meios de organização implementados ao site, os quais foram baseados em possíveis utilizações no dia a dia. O segundo objetivo buscava adotar alterações e atualizações em sequência aos meios de organizações para que o usuário pudesse possuir a liberdade de adulteração desses dados. E como o terceiro e último fora a criação de lembretes com a utilização de cores para identificação de prazos de tarefas no site.

Após a criação e fundamentação de tais funções é capaz de visualizar a possibilidade de melhora de vida do usuário com a utilização do site. Trazendo uma maior estabilidade com essas ferramentas e uma estabilidade pessoal, no trabalho e social.

Em evoluções futuras busca-se melhoria das ferramentas já existentes e criação de novos meios para melhores aperfeiçoamentos.

8 REFERÊNCIAS

Castro, Gabriel. Você sabe o que é Markdown. Disponível em: <https://canaltech.com.br/software/Voce-sabe-o-que-e-Markdown/>. Acesso em 06 maio 2021.

Craveiro, Raul. O que é Markdown. 31 mar 2020. Disponível em: <https://diolinux.com.br/tutoriais/o-que-e-markdown.html>. Acesso em 06 maio 2021.

Edson. Introdução ao Visual Studio Code, 2016. Disponível em: <https://www.devmedia.com.br/introducao-ao-visual-studio-code/34418>. Acesso em 09 nov 2021.

Flinco, Rubens. O que é Figma e por que usar ele? 28 fev 2021. Disponível em: <https://designe.com.br/o-que-e-o-figma-e-por-que-usar-ele/>. Acesso em 09 nov 2021.

Gonçalves, Ariane. O que é CSS? Guia Básico para Iniciantes, 30 nov 2020. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-css-guia-basico-de-css>. Acesso em 02 maio 2021.

JetBrains. Introduction, 13 set 2021. Diposnível em: <https://www.jetbrains.com/help/datagrip/meet-the-product.html#datagrip_overview>. Acesso em 09 nov 2021. Tradução livre.

L3 Software. DataGrip: uma plataforma indicada para quem trabalha com linguagem SQL. Disponível em: <https://l3software.com.br/softwares/datagrip-uma-plataforma-indicada-para-quem-trabalha-com-linguagem-sql/>. Acesso em: 09 nov 2021

Marques, Rafael. O que é HTML? Entenda de forma descomplicada. Disponível em: <https://www.homehost.com.br/blog/tutoriais/o-que-e-html/>. Acesso em: 06 maio 2021.

MDN contributors. CSS: Tutorial, 04 maio 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/CSS>. Acesso em 05 maio 2021.

MDN contributors. HTML: Linguagem de Marcação de Hipertexto, 02 maio 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>. Acesso em 02 maio 2021.

Noleto, Cairo. Ciclo de vida do Phoenix LiveView. Disponível em: <https://blog.betrybe.com/txn-avancado/ciclo-de-vida-phoenix-liveview/>. Acesso em 14 nov 2021

O QUE É DOCKER, Red Hat, Disponível em: <https://www.redhat.com/pt-br/topics/containers/what-is-docker>. Acesso em 02 maio 2021.

Oliveira, Felipe. Elixir & Phoenix: Primeira impressões, 20 mar 2019. Disponível em: < https://medium.com/codengage/elixir-phoenix-primeiras-impress%C3%B5es-8624bd491158>. Acesso em 14 nov 2021

Paixão, João Roberto. Ngrok: do Localhost para o Mundo, 3 jan 2021. Disponível em: <https://medium.com/desenvolvendo-com-paixao/ngrok-do-localhost-para-o-mundo-5445ad08419>. Acesso em 30 out 2021

Patryk Bak, Concurrency and parallelism with Elixir and BEAM. 25 ago. 2020, Disponivel em: <https://appunite.com/blog/concurrency-and-parallelism-with-elixir-and-beam>. Acesso em 30 out. 2021. Tradução Livre.

Phoenix, HexDocs. Disponível em: <https://hexdocs.pm/phoenix/overview.html>. Acesso em 02 maio 2021. Tradução Livre.

Reis, Daniel. O QUE É GIT: Primeiro Exemplo, Disponivel em: <https://github.com/DanielHe4rt/git4noobs/blob/master/2-o-que-e-git/primeiro-exemplo.md>. Acesso em 02 maio 2021.

Reis, Daniel. O QUE É GIT: Secundo Exemplo, Git4Noobs, Disponivel em: <https://github.com/DanielHe4rt/git4noobs/blob/master/2-o-que-e-git/segundo-exemplo.md>. Acesso em 02 maio 2021.

Santos, Peterson F. FAQ Elixir: Tudo o que você precisa saber para fazer a escolha certa, 20 jun 2020. Disponível em: <https://ateliware.com/blog/tudo-que-voce-precisa-para-escolher-elixir>. Acesso em 02 maio 2021.

Souza, Alexandre. Sobre a linguagem, 29 mar 2020. Disponível em: <https://github.com/aleDsz/elixir4noobs/blob/master/contents/3%20-%20Linguagem/1-Sobre%20a%20linguagem.md>. Acesso em 02 maio 2021.

Strzibny, Josef. Elixir and Phoenix after two year. 20 abr 2021. Disponível em: <https://nts.strzibny.name/elixir-phoenix-after-two-year/>. Acesso em 06 maio 2021. Tradução Livre.

Teixeira, Henrique F, Elixir: 10 motivos para aprender. 25 nov. 2018. Disponível em: <https://medium.com/true-henrique/elixir-10-motivos-para-aprender-6cd4d6876f05>. Acesso em 30 out. 2021.

The Elixir Team, Elixir lang. Disponível em: <https://elixir-lang.org>. Acesso em 02 maio 2021. Tradução Livre.

The PostgreSQL Global Development Group. About:What is PostgreSQL?, 2021. Disponível em: <https://www.postgresql.org/about/>. Acesso em 27 maio 2021. Tradução livre.