

ネットワークプログラミングI

レポート課題2 自由製作

提出期限 2020 年 12 月 25 日 17:00

提出日 2020 年 12 月 24 日

組番号 408

学籍番号 17406

氏名 金澤雄大

1 目的

ネットワークプログラミング I で学んだ内容の理解度を高めるために、自由製作の Web アプリケーションを作成することを目的とする。

2 アプリの説明

Web アプリケーションとして、ソースコードを共有するアプリを作成した。アプリケーションの主な機能を次に示す。本章では、アカウント管理および記事の例について述べる。記事の作成、閲覧、編集、削除、検索およびランキングの表示は 5 章で詳しく説明する。

- アカウントの作成
- ログイン/ログアウト
- パスワードの変更
- 記事の作成
- 記事の閲覧
- 記事の編集
- 記事の削除
- 記事の検索
- 「いいね」数に応じたランキング

アカウント管理としてアカウントの作成、ログイン/ログアウト、パスワードの変更の 3 つを実装している。ログイン/ログアウトといったセッションの継続には cookie を用いる。cookie について説明する。HTTP の接続は基本的に 1 回ごとに切断される。そこで、あるページを表示させるリクエストがサーバ側に来たときに、保存しておくべき情報をサーバから web ブラウザ、つまりユーザー側に保存する。この保存しておく情報を cookie という。別のページに移動するたびに、web ブラウザは新たにアクセスしたサイトの cookie を所持しているか確認する。もし cookie を持っていれば、それをサーバに送信する。サーバでは cookie の解読を行って、作業の続きの処理を行う。これによって、HTTP の接続は独立しているが、ユーザーにはあたかもセッションが継続しているように見える。

cookie 使用のイメージは、例えば Amazon[1] でカートに商品を入れたとする。このときに、ユーザーとカートの内容を識別できるような情報を cookie に持たせておく。そして、会計のページにアクセスしたときに web ブラウザはサーバに cookie を送信する。サーバは cookie を解読してカートの内容を理解し、会計の処理を行う。これによってユーザーはカートに入れた商品が次のページで消去されるということなく、スムーズな購入を行うことができる。

アプリのコンテンツとして、記事の投稿を行うことができる。本アプリでは、ソースコードの共有のための説明を記事の内容として想定している。そのため、記事は次の 4 つから構成されるような仕様にした。

1. タイトル
2. 説明
3. ソースコード
4. 実行結果

図 1 に記事の例を示す。図 1 は、「はじめての C 言語」というタイトルの記事である。投稿者はタイトルの横に「@投稿者名」という形式で表示される。図 1 では投稿者名は「test」である。レポートでは読み取れないが、投稿者名は緑色で表示されている。更新日、いいね数の情報も記事の先頭に表示される。図 1 では記事の更新日が「2020/12/16 07:25:57」、いいね数が 0 になっている。記事の説明は「C 言語の Hello World のためのコードです。」という部分である。図 1 の投稿では説明は 1 文しかないが、実際には 1000 文字まで入力できる。ソースコード、実行結果は枠で囲って表示される。ソースコード、実行結果ともに 1000 文字まで入力できる。



図 1: 記事の例

3 実行環境

本章では、通信の流れ、実行環境、ディレクトリ構造の 3 つについて述べる。

3.1 通信の流れ

実行環境は Windows 上の仮想環境で CentOS を動作させる。つまり、1 台のコンピュータの中で CentOS がサーバー側、Windows がクライアント側となって動作する。

図 2 にアプリケーションの通信の流れを示す。ユーザー (クライアント) からのアクセスがあると、サーバー側では Webrick という Ruby 用のサーバーフレームワークが待ち構えており、Webrick の命令によって Ruby プログラムが実行される。Ruby のプログラムはデータベースやリダイレクトの処理を行う。このため、Ruby からデータベース操作を行う必要がある。Ruby からデータベースの読み書きを行うために ActiveRecord を用いる。ActiveRecord は Ruby のオブジェクトを操作すると、内部で操作に対応したクエリ (SQL の命令文のこと) を発行する機能がある。この機能を用いることで、Ruby のオブジェクトを操作するだけで、データベースの操作を行うことができる。

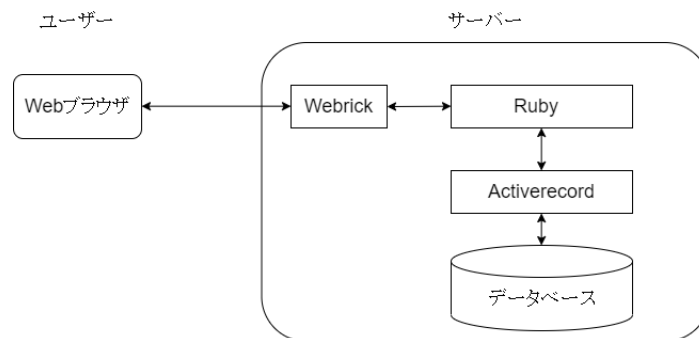


図 2: 通信の流れ

3.2 実行環境

本アプリでは、セッション管理を用いているため、ある端末でログインして、ページを見ているときに他のユーザーが同じページにアクセスした場合の処理を確認する必要がある。このため、クライアント用の端末が 2 台必要である。これにサーバ用の仮想環境を加えると合計 3 台の環境が必要となる。表 1 にサーバの仮想環境を示す。また、表 2 にホスト端末、表 3 に 2 台目の端末の実行環境を示す。以後、ホスト端末をユーザー 1, 2 台目の端末をユーザー 2 と呼ぶことにする。動作検証は「Google Chrome」で行う。「Microsoft Edge」や「Internet Explorer」, 「Mozilla Firefox」での動作は保証しない。

表 1: サーバー (仮想) の実行環境

メモリ	2048MB
OS	CentOS Linux release 8.1.1911(Core)
Ruby	2.7.1p83 (2020-03-31 revision a0c7c23c9c)
Sinatra	version 2.1.0
SQLite3	version 1.3.2
ActiveRecord	version 6.1.0

表 2: ユーザー 1 の実行環境

CPU	Intel(R) Core(TM) i7-6500U CPU 2.50Ghz
メモリ	16.0GB DDR3
OS	Microsoft Windows 10 Home
Google Chrome	version : 87.0.4280.88 (official Build) (64bit)

表 3: ユーザー 2 の実行環境

CPU	AMD Ryzen 5 3600 6-Core Processor
メモリ	16.0GB DDR4
OS	Microsoft Windows 10 Home
Google Chrome	version : 87.0.4280.88 (official Build) (64bit)

3.3 ディレクトリ構造

リスト 1 にディレクトリ構造を示す. リスト 1 は「tree」コマンドで Web アプリを構成するディレクトリを表示したものである.

リスト 1: ディレクトリ構造

```

1 4J_Chamacode
2   Gemfile
3   chamacode.db
4   chamacode.sq3
5   database.yml
6   main.rb
7   public
8       deleteCheck.js
9       image1.png
10      search.js
11  views
12      badrequest.erb
13      beginner.erb
14      contents.erb
15      detail.erb
16      edit.erb
17      failure.erb
18      failure1Resister.erb
19      failure2Resister.erb
20      failure3Resister.erb
21      forgetpass.erb
22      layout.erb
23      loginscr.erb
24      logout.erb
25      makeAccount.erb
26      myarticle.erb
27      newarticle.erb
28      ranking.erb
29      successRenewpass.erb
30      successResister.erb
31      successarticle.erb
32      unknownUser.erb

```

アプリの作成に必要なファイルは次のように 4 つに分けられる.

- Ruby とそのライブラリの管理, 設定を行うファイル
- データベースのためのファイル
- 公開するもの (JavaScript や画像) を格納するファイル
- ページの見た目を管理する HTML を格納するファイル

Ruby とそのライブラリ, 設定を行うファイルが, 「Gemfile」, 「main.rb」, 「database.yml」の 3 つである. 「Gemfile」は Ruby のライブラリ管理を行うためのファイルである. 「main.rb」はサイトにアクセスし

たときに、Webrick によって呼び出される Ruby のプログラムである。「database.yml」はデータベースの設定ファイルである。データベースと Ruby のプログラムのアダプターとなるデータベース言語の設定と、接続するデータベースを記述する。

データベースのためのファイルとして「chamacode.sqlite3」と「chamacode.db」の2つがある。「chamacode.sqlite3」はテーブルの定義を記述した独自拡張子のファイルである。「chamacode.db」はデータベースのファイルである。「chamacode.db」が破壊または削除されたときに、「chamacode.sqlite3」を用いることで、データベースの再定義を容易に行うことができる。

public ディレクトリには公開するもの (JavaScript や画像) を設置している。「public」というフォルダに設置することで、詳細なパスを指定しなくても JavaScript や画像を取得することができる。views ディレクトリにはページの見た目を管理する「*.erb」を設置している。ページ共通の見た目は「layout.erb」に記述し、ページ固有の見た目はそれぞれの erb ファイルで記述する。

4 データベース定義

本章では2つのデータベースの仕様、および関係について述べる。

4.1 users テーブル

users テーブルはユーザー名、パスワードを管理するためのテーブルである。ユーザー名は固有、つまり被りのないよう名前をつける仕様とする。パスワードは、パスワードとユーザーにランダムに設定した文字列とのハッシュ値をデータベースで保存する仕様にする。生のパスワードを保持すると、データベースが流出したときに大変なことになる。そこで、パスワードを暗号化することで、流出した場合も解読に時間がかかるようにする対策を行う。最も単純な暗号化の方法はパスワードのハッシュ値を保存することである。ハッシュ値の方向性を利用することで、解読に時間をかける対策を行うことができる。しかし、単純にハッシュ値を計算して保存することには問題点がある。この方法では、2人のパスワードが一致しているときに、ハッシュ値が一致するため、間接的にパスワードが解読される可能性がある。例えば Aさんと Bさんが共に「passdesu」というパスワードを使っていたとする。このとき、2人のパスワードから計算されたハッシュ値も一致する。Aさんがサーバーに侵入してデータベースを見ると、Aさんと Bさんが同じハッシュ値であることがわかるため、Bさんのパスワードがわかってしまう。

これを防ぐために、パスワードにランダムな文字列をつけて計算したハッシュ値をデータベースに保存する。このランダムな文字列を salt という。salt をつけることで、パスワードが同じであっても「パスワード+salt」から計算されたハッシュ値は異なるため、先のような問題が解決できる。

表 4 に users テーブルの定義を示す。テーブルの主キーは username カラムである。主キーとは、レコードを一意に認識するためのユニークな値のことを指す。ユーザー名、salt、パスワードは長さ 40 の文字列としている。ハッシュアルゴリズムはアルゴリズムの変更に対応するためのカラムである。しかし、今回のコードではハッシュアルゴリズムは MD5 で固定とし、このカラムには常に 1 の値を入れ、ハッシュアルゴリズムの変更は行われたいものとする。

表 4: users テーブルの定義

物理名	論理名	型	主キー
ユーザー名	username	char(40)	
salt	salt	varchar(40)	×
パスワードのハッシュ値	hashed	varchar(40)	×
ハッシュアルゴリズム	algo	char(5)	×

リスト 2 に users テーブルを定義するためのコードを示す。リスト 2 は chamacode.sql の抜粋である。コードの説明をする。リスト 2 の 1 行目では「users」というテーブルを作成することを宣言している。そして 2 行目から 5 行目で表 4 の仕様に従うように、テーブルの内容を定義している。カラムの宣言は「論理名 型 オプション」という形式で行う。2 行目の場合、論理名が「username」、型が「char(40)」、オプションが「primary key」である。表 4 において、username カラムの型は「char(40)」、オプションは主キーであるから、2 行目の宣言は仕様に一致している。

リスト 2: users テーブル (chamacode.sql の抜粋)

```

1 create table users (
2   username char(40) primary key,
3   salt varchar(40),
4   hashed varchar(40),
5   algo char(5)
6 );

```

4.2 contents テーブル

2 章で示したように、記事にはタイトル、説明、ソースコード、実行結果の 4 つの項目があった。contents テーブルはこの 4 つの項目に更新日やいいね数を加えた情報を保持するテーブルである。表 5 に contents テーブルの定義を示す。テーブルの主キーは id である。

表 5: contents テーブルの定義

物理名	論理名	型	主キー
id	id	char(40)	
ユーザー名	username	char(40)	×
更新日	date	char(19)	×
タイトル	title	char(60)	×
説明	description	char(1000)	×
ソースコード	code	char(1000)	×
実行結果	result	char(1000)	×
公開状況	open	int	×
いいね数	good	int	×

カラムの仕様について説明する。id は記事の登録時にランダムに割り振る。このため id の衝突が起きる可能性がある。id を構成する数字は 0~9 の 10 文字、アルファベットは a~z の 26 文字で大文字小文字を区別

すると 52 文字ある. id の文字数は 40 文字だから, id に割り振れる文字列の組み合わせは $(10 + 52)^{40} \simeq 10^{72}$ である. これは十分に大きい数であるから衝突する可能性は低いと考えられる. このため, id はランダムに割り振る仕様にする.

更新日は「2020/12/25 16:59:59」という形式で保存する. この形式で保存するためには 19 文字の文字列を確保すればよい. このため, 表 5 の更新日カラムの型は char(19) になっている..

公開状況は 0 のとき「非公開」, 1 のとき「公開」とする. いいね数は初期値を 0 とする. この仕様では誰がどの記事に「いいね」を押したかを保持するテーブルがないため, 同じ記事に何度でも「いいね」をつけられる. 記事に「いいね」を押したユーザーを記録するテーブルを作成すると, 記事とユーザーが増えたときに, 「いいね」を押したか判定する時間が増えるためサーバの負荷が高くなると考え, このような仕様になっている.

リスト 3 に contents テーブルを定義するコードを示す. リストのコードは 3 は表 5 の仕様を満たすテーブルの内容を定義している.

リスト 3: contents テーブル (chamacode.sql3 の抜粋)

```
1 create table contents (  
2 id char(40) primary key,  
3 username char(40),  
4 date char(19),  
5 title char(60),  
6 description char(1000),  
7 code char(1000),  
8 result char(1000),  
9 open int,  
10 good int  
11 );
```

4.3 ER 図

図 3 に ER 図を示す. ER 図はデータベース設計において, テーブルとその関係を示すために用いる図である. 2 つのテーブルにおいて username は共通したカラムである. 共通したカラム間では, 整合性を保ちたいため, 外部キー (Foreign Key) を用いることがある. 作成したデータベース定義では外部キーになっていないが, Ruby のプログラムではあたかも外部キーであるかのように扱う.

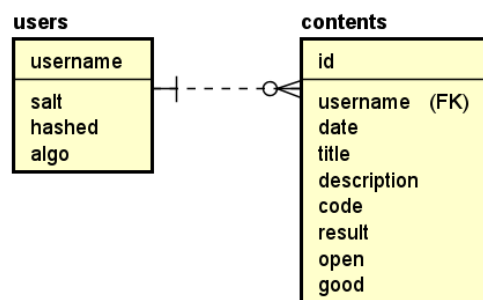


図 3: ER 図

5 機能の説明と実行結果

本章では次に示す機能の説明とその実行結果について述べる.

1. データベースの初期化とサーバーの起動
2. 全てのページに共通するレイアウトの設定
3. アカウントの作成
4. ログイン/ログアウト
5. パスワードの変更
6. 記事の作成
7. 記事の閲覧
8. 記事の編集
9. 記事の削除
10. 記事の検索
11. 「いいね」機能
12. ランキングの表示

5.1 データベースの初期化とサーバの起動

データベースの初期化およびサーバ起動の方法について説明する。データベースの初期化はリスト4のコマンドで行う。リスト4の1行目は, `chamacode.db` を削除するコマンドであるため, すでにファイルが存在するときに実行してほしい。2行目はSQLiteに「`chamacode.sq3`」(リスト26)のクエリを流して, 「`chamacode.db`」というデータベースを作成するコマンドである。

リスト 4: データベースの初期化

```
1 $ rm chamacode.db
2 $ sqlite3 chamacode.db < chamacode.sq3
```

データベースの作成が行えたから, サーバを起動する。リスト5にサーバを起動するコマンドを示す。リスト6にリスト5のコマンドの実行結果を示す。リスト6のような実行結果ができれば, サーバの起動に成功している。

リスト 5: サーバを起動するコマンド

```
1 $ bundle exec ruby main.rb
```

リスト 6: サーバ起動の成功例

```
1 [2020-12-16 11:36:27] INFO WEBrick 1.6.0
2 [2020-12-16 11:36:27] INFO ruby 2.7.1 (2020-03-31) [x86_64-linux]
3 == Sinatra (v2.1.0) has taken the stage on 4567 for production with backup from WEBrick
4 [2020-12-16 11:36:27] INFO WEBrick::HTTPServer#start: pid=5239 port=4567
```

起動に成功したら, Web ブラウザからサイトにアクセスする。ユーザー1の場合はホストであるから, 自分自身を指すIPアドレス(ループバックアドレス)である「127.0.0.1」に, ポート番号9998をつけた「127.0.0.1:9998」にアクセスすればよい。ユーザー2の場合はユーザー1(ホスト)のIPアドレスにポート番号をつけたページにアクセスすればよい。図4はユーザー1がGoogle ChromeでサイトにアクセスするためにURLを入力

した状態である。図 4 の状態で「Enter」キーを押すと、図 5 に示すように「127.0.0.1:9998/login」に自動でリダイレクトする。図 5 のページがログインページである。

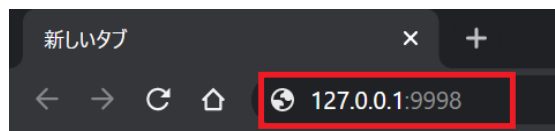


図 4: ユーザー 1 が URL を入力した状態

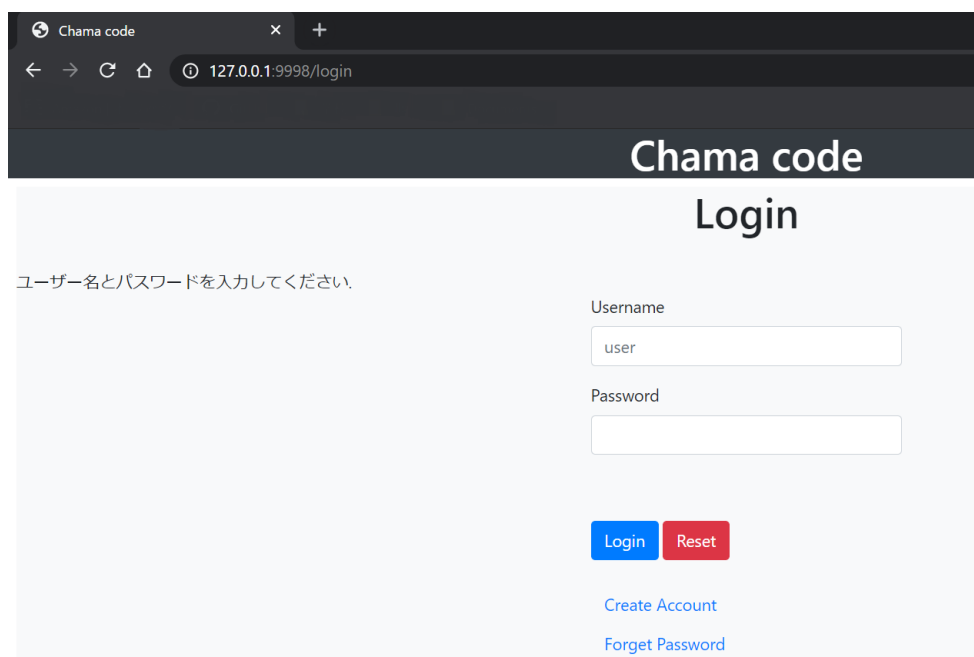


図 5: ログイン画面へのリダイレクト (ユーザー 1)

ユーザー 2 の場合、図 6 に示すように、ユーザー 1(ホスト)の IP アドレスにポート番号 9998 をつけた URL にアクセスする。図 6 では、ホストの IPv4 アドレスは「192.168.0.102」になっている。図 7 にユーザー 2 がサイトにアクセスしたときの様子を示す。図 7 からユーザー 2 の場合も「/login」に自動でリダイレクトして、ログインページが表示されていることが読み取れる。

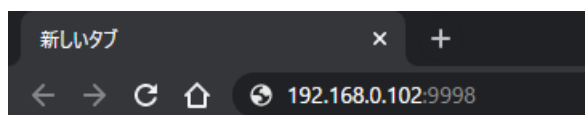


図 6: ユーザー 2 が URL を入力した状態

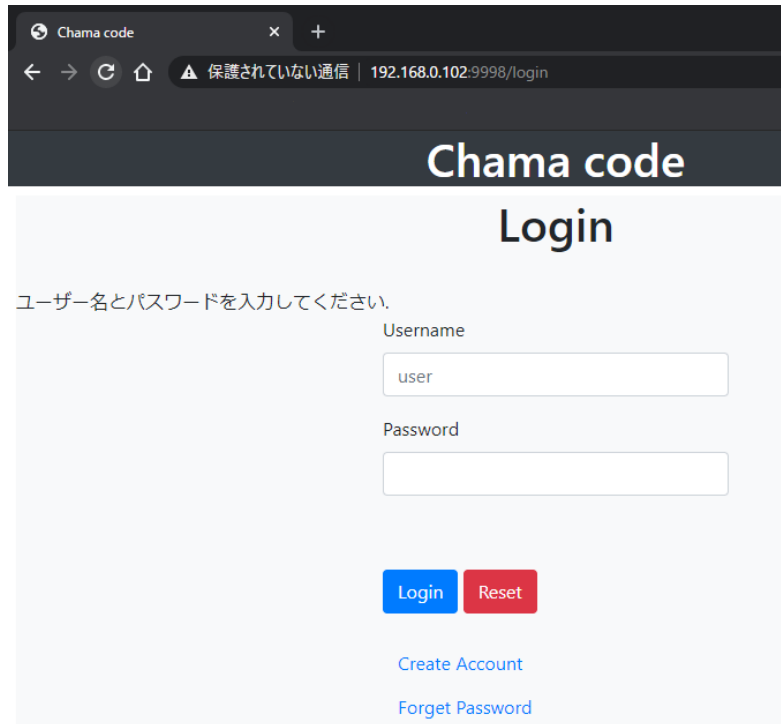


図 7: ログイン画面へのリダイレクト (ユーザー 2)

ユーザーが「/」にアクセスしてきたときに、自動で「/login」にリダイレクトが行われる仕組みについて説明する。サイトにアクセスしたときの処理は、計算やリダイレクトといったバックエンドの処理と、HTMLやJavaScriptのフロントの処理の2つに分けられる。リダイレクトの処理を行っているのは、前者である。リスト7に「/」にアクセスしたユーザーを「/login」にリダイレクトする部分のコードを示す。

リスト 7: 「/login」にリダイレクトするコード (main.rb)

```
1 # loginにリダイレクト
2 get '/' do
3   redirect '/login'
4 end
5
6 # ログインフォームを表示
7 get '/login' do
8   erb :loginscr
9 end
```

リスト7のコードについて説明する。リスト7の2行目は、「/」にアクセスがきたときの処理であることを宣言する命令である。「get '/hoge' do」と記述すると、「/hoge」にアクセスがきたときの処理を宣言するという意味になる。処理の内容は3行目である。3行目では、「/login」にリダイレクトする処理を行っている。これによって「/」にアクセスしたユーザーは自動で「/login」にリダイレクトされる。「/login」にアクセスしたときの処理は、リスト7の7行目から9行目である。「/login」での処理は「loginsrc.erb」の内容を表示するという処理である。「erb : hoge」という記述をするとhoge.erbの内容が表示される。loginsrc.erbの内容は付録のリスト47を参照してほしい。

5.2 全てのページに共通するレイアウトの設定

全てのページに共通するレイアウトを設定する方法を説明する. 図 8 および図 9 にページのサンプルを示す. 図 8 はログインページ, 図 9 はユーザーの記事の一覧を表示するページである. 2 つのページに共通しているデザインは 2 つある. 1 つ目は, 画面上部に黒背景の白文字で表示される「Chama code」という文字である. 2 つ目は, 図 8 の「ユーザー名とパスワードを入力してください。」という文字や, 図 9 の「test さんの記事一覧です」という文字は, 画面端から 8px 開けて表示するようになっている. これは細かい設定であるため, 画像から読み取ることが難しい. 図 10 にページ左端の余白を比較するための画像を示す. 図 10 では, 「test さんの記事一覧です」という文字を左側の余白を変えて表示している. 余白の設定は上が 0px, 下が 8px である. 余白が 0px のとき, 1 文字目の「t」が画面の左端にくっついていて読みにくい印象を受ける. このため, 左に 8px の余白を設けている.

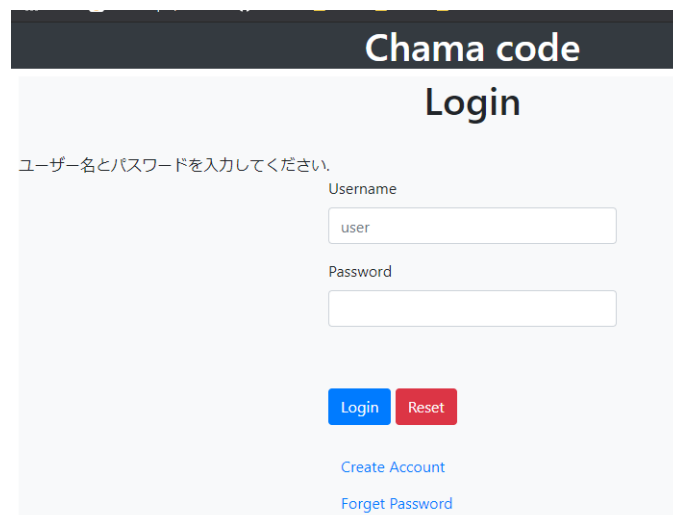
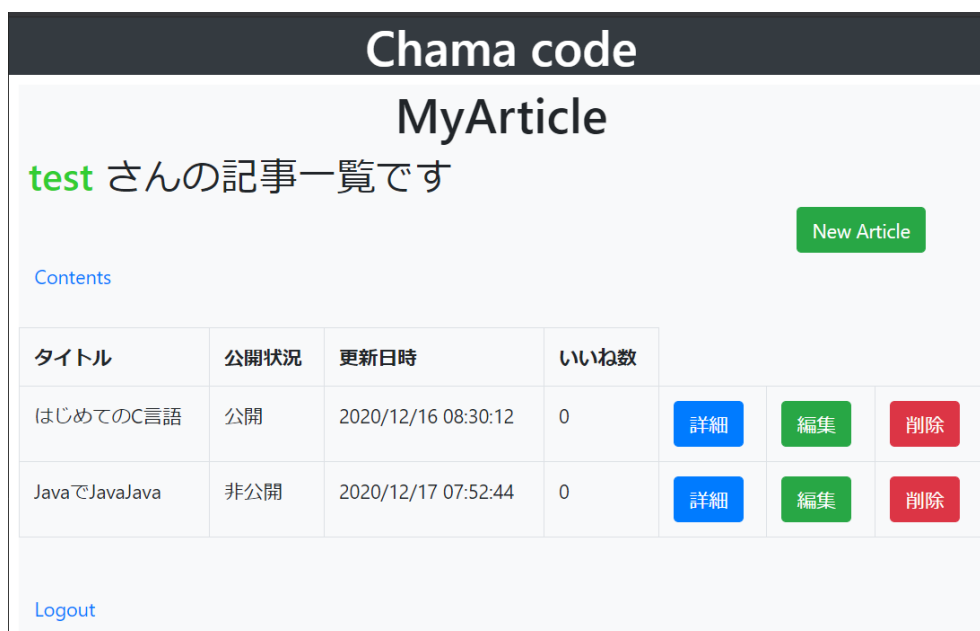


図 8: ログインページ



タイトル	公開状況	更新日時	いいね数			
はじめてのC言語	公開	2020/12/16 08:30:12	0	詳細	編集	削除
JavaでJavaJava	非公開	2020/12/17 07:52:44	0	詳細	編集	削除

図 9: ユーザーの記事一覧を表示するページ

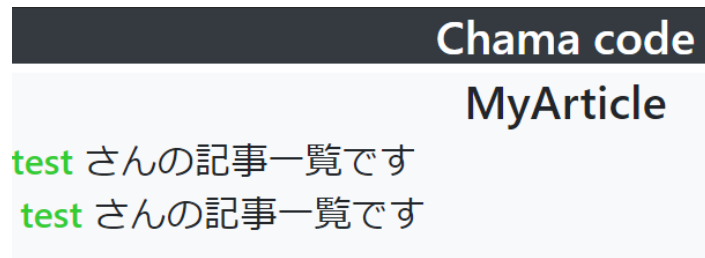


図 10: ページ左端の余白の比較

全てのページで共通するレイアウトを、それぞれのコードに記述すると、プログラムが冗長になり、メンテナンス性が落ちる。このため、全てのページに共通するレイアウトを 1 箇所に記述することで全体に反映させる仕組みがある。「views/layout.erb」に共通するレイアウトを記述することで全てのページにレイアウトが反映される。リスト 8 に layout.erb を示す。本アプリでは、ページのレイアウトを整えるために、Bootstrap[2]を用いている。Bootstrap はブラウザでのレイアウトを調整するためのフレームワークである。ただし、これは外部のライブラリであるため、使用はボタンや表、余白のレイアウトを整えることのみに用いる。Bootstrap の読み込みはリスト 8 の 3 行目から 5 行目のリンクで行っている。

黒背景に白文字で「Chama code」と表示するコードは、リスト 8 の 12 行目から 16 行目である。12 行目の div タグで「bg-dark text-white」という Bootstrap を用いたレイアウト設定を行っている。「bg-dark」は背景色 (background color) を dark, つまり黒に近い色に設定するという意味である。「text-white」は文字 (text) を白に設定するという意味である。この設定は div タグの開始 (<div>) から div タグの終わり (</div>) まで有効である。

左端の余白を開けるコードは、リスト 8 の 18 行目から 20 行目までである。18 行目の div タグでは背景を白、文字を黒に設定している。余白の設定は「mx-2」という記述である。これは左右のマージンを 8px 開けるという意味である。19 行目の「<%= yield %>」は他の erb ファイルの記述を埋め込むための記述である。layout.erb はページに共通するレイアウトを記述するファイルであるから、ページの HTML を記述する場所を指定する必要がある。それが 19 行目のコードである。

リスト 8: views/layout.erb

```
1 <!DOCTYPE html>
2 <html>
3 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/
4 4.5.0/css/bootstrap.min.css" integrity="sha384-9aIt2nRpC12Uk9gS9baD1411NQA
5 pFmC26EwA0H8WgZl5MYXfFc+NcPb1dKGj7Sk" crossorigin="anonymous">
6 <head>
7 <meta charset="utf-8" />
8 <title>Chama code</title>
9 </head>
10
11 <body>
12 <div class="bg-dark text-white">
13 <center>
14 <h1>Chama code</h1>
15 </center>
16 </div>
17
18 <div class="bg-light text-black mx-2">
19 <%= yield %>
20 </div>
21 </body>
22 </html>
```

5.3 アカountの作成

本章では、アカウントの作成について次に示す5つの内容を述べる。

1. アカountの作成方法
2. アカountの作成に失敗する例 (ユーザー名の重複)
3. アカountの作成に失敗する例 (不正な入力)
4. アカountの作成に失敗する例 (パスワードの不一致)
5. 実装部分の説明

5.3.1 アカountの作成方法

アカウントの作成方法について説明する。まず、アカウントの作成ページにアクセスする。アカウントの作成ページへのアクセスは、ログインページ (図8) の「Create Account」をクリックする、もしくは「/createaccount」にアクセスすると行える。図11がアカウント作成のページである。図11からわかるように、アカウントの作成には、ユーザー名とパスワードが必要である。さらに、パスワードは2回入力するようになっている。パスワードを2回入力させることで、誤ったパスワードでアカウント登録を行って、ログインできなくなる不具合が起きることを防いでいる。

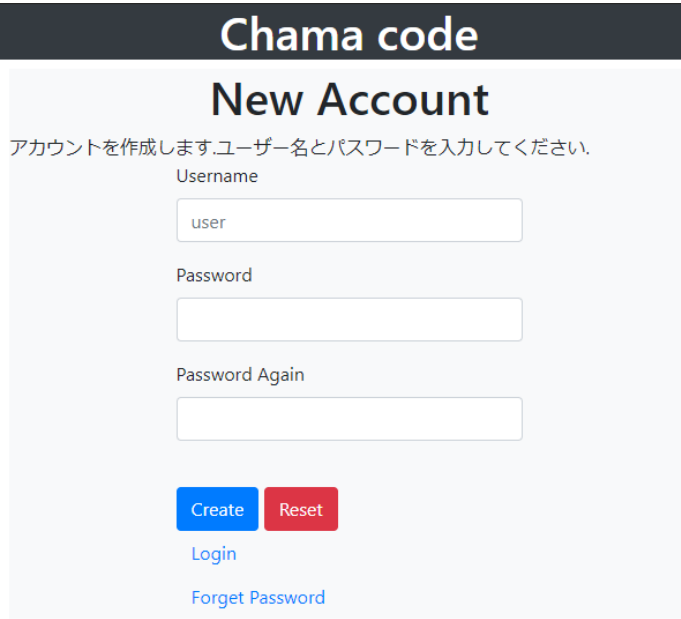


図 11: アカount作成ページ

ユーザー名が「test」、パスワードが「abcdefg1234」のアカウントを作成してみる。図12がユーザー名とパスワードをフォームに入力した状態である。ユーザー名は表示されるが、パスワードはどちらのフォームも「*」(アスタリスク)で表示される。フォームの入力内容を消したい場合、「Reset」というボタンを押すと、フォームの入力が消去される。

図 12: 新規アカウントの入力

フォームへの入力完了したら、「Create」のボタンを押す。「Create」ボタンを押すことで、サーバに入力内容が送信され、アカウント登録の処理が行われる。図 13 に「Create」ボタンを押したときの結果を示す。アカウントの作成に成功すると、「/successResister」にリダイレクトする。

図 13: アカウントの作成成功

この新規アカウント作成の方法は、一人のユーザーが複数のアカウントを作成することが容易に可能である。多くのサービスでは新規アカウント作成時にメールアドレスと連携することで、一人のユーザーが作れるアカウントの数を頭打ちにしている。開発時間の関係上、メールアドレスとの連携は実装できなかったため、このアプリでは、このような仕様になっている。実装という面では Ruby のプログラムからメールを送信する方法は、授業で学習したため、メールアドレスとの連携は可能であると考えられる。

5.3.2 アカウントの作成に失敗した例 (ユーザー名の重複)

4.1 章の users テーブルの定義で述べたように、ユーザー名は重複しない仕様になっている。このため、すでにユーザー登録がされているユーザー名で新規アカウントを作成しようとした場合、アカウントの作成を失敗する仕様になっている。

この仕様が実装できていることを確認する。アカウント作成ページ（「/createaccount」）にアクセスし、前項で作成した「test」というユーザー名で新規アカウント登録を行う。パスワードは「qwerty5678」とする。図 14 がアカウント作成フォームにユーザー名とパスワードを入力した状態である。図 15 に、図 14 の状態で「Create」ボタンを押してアカウント登録をしたときの結果を示す。アカウント登録に成功したときは、「/successResister」にリダイレクトしたが、この場合は「/failure1Resister」にリダイレクトする。図 15 の表示は「ユーザー名がすでに存在しているため、アカウント登録に失敗した」ということを意味している。これによって、ユーザー名を重複してアカウント登録ができないことが確認できた。

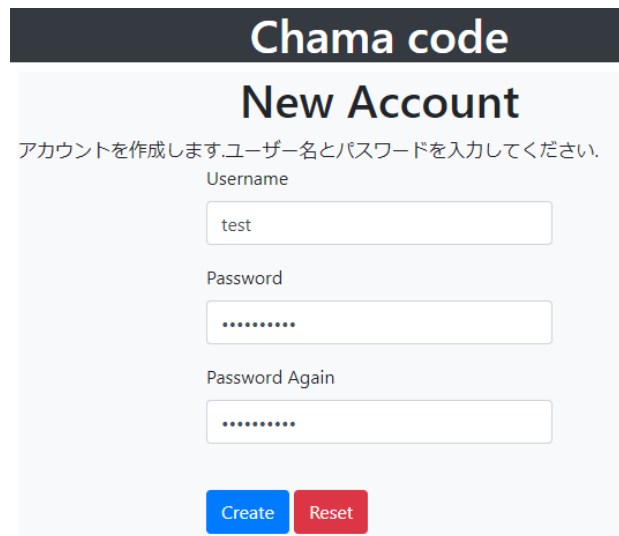


図 14: アカウントを重複して登録

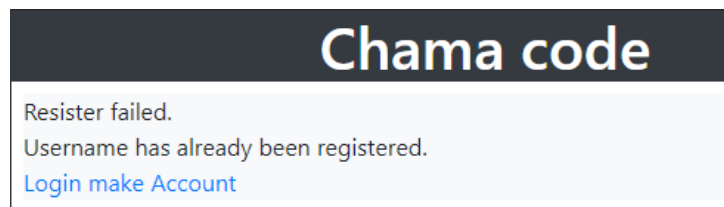
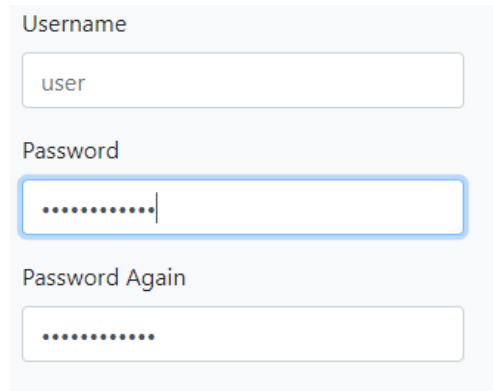


図 15: アカウントの作成失敗 (ユーザー名の重複)

5.3.3 アカウントの作成に失敗した例 (不正な入力)

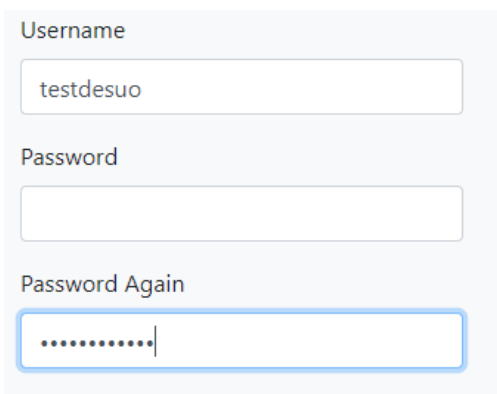
4.1 章で述べたように、ユーザー名およびパスワードは 40 文字以内という仕様であった。アカウント入力のフォームに何も入力せずに登録しようとした場合や、40 文字を超えた入力をサーバが受け取った場合は、アカウント登録に失敗する仕様になっている。

この仕様が実装されていることを確認する。まず、ユーザー名もしくはパスワードに何も入力せずにアカウント登録を試みる。図 16 から図 18 に示す 3 つの入力フォームでアカウント作成を行う。図 16 は、ユーザー名を入力せず、パスワードに「passdesu1234」と入力した状態である。図 17 はユーザー名に「testdesuo」と入力し、上側のパスワードフォームに「passdesu1234」と入力した状態である。図 18 はユーザー名に「testdesuo」と入力し、下側のパスワードフォームに「passdesu1234」と入力した状態である。図 19 に 図 16 から図 18 のフォームでアカウント登録を行った結果を示す。アカウント登録に成功したときは、「/successResister」にリダイレクトしたが、3 つのどの場合でも「/failure3Resister」にリダイレクトする。図 19 の表示は「不正な入力があったためアカウントの作成が失敗した」ということを意味している。これによって、入力が 0 文字のとき、アカウントの作成が失敗することが確認できた。



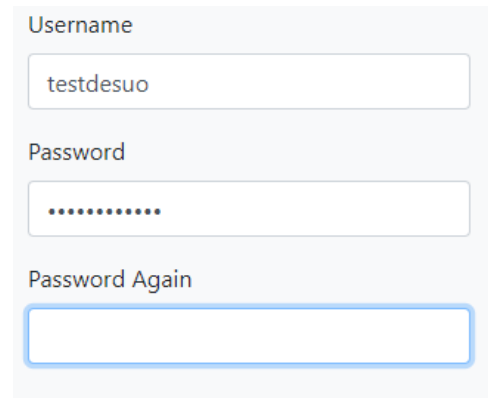
A screenshot of a user registration form. It contains three input fields: 'Username' with the text 'user', 'Password' with masked characters '.....', and 'Password Again' with masked characters '.....'. The 'Password' field is highlighted with a blue border.

図 16: ユーザー名を未入力でアカウントを作成



A screenshot of a user registration form. It contains three input fields: 'Username' with the text 'testdesuo', 'Password' which is empty, and 'Password Again' with masked characters '.....'. The 'Password Again' field is highlighted with a blue border.

図 17: パスワードを未入力でアカウントを作成 1



A screenshot of a user registration form. It contains three input fields: 'Username' with the text 'testdesuo', 'Password' with masked characters '.....', and 'Password Again' which is empty. The 'Password Again' field is highlighted with a blue border.

図 18: パスワードを未入力でアカウントを作成 2

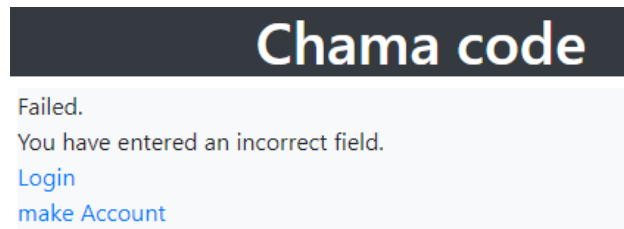


図 19: アカウントの作成失敗 (入力不正)

入力フォームに 40 文字以上の文字を入力してアカウント登録したときに、登録に失敗する仕様を確認する。確認はユーザー名を 40 文字以上にしてフォームを送信することで行う。ユーザー名、パスワードに 40 文字以上の入力があった場合のサーバの処理は共通しているから、パスワードに 40 文字以上の入力があった場合の確認は省略する。入力フォームは HTML で文字数制限をしているから、手入力では 40 文字までしか入力できない。開発者ツールを用いると 40 文字以上の入力が可能である。Google Chrome の場合は「F12」で開発者ツールを開き、左上のマウスカーソルアイコンをクリックする。そしてユーザー名のフォームをクリックすると、図 20 に示すようにページのソースが表示される。図 20 の「maxlength="40"」の 40 の部分を 40 より大きい数字に変更することで、入力フォームの文字数制限が変更されるため、40 文字以上の入力が可能になる。

```
<div class="form-group">
  <label for="exampleFormControlInput1">Username</label>
  <input type="text" class="form-control" name="uname" maxlength="40" placeholder="user"> == $0
</div>
<div class="form-group">_</div>
<div class="form-group">_</div>
```

図 20: 開発者ツールでの操作

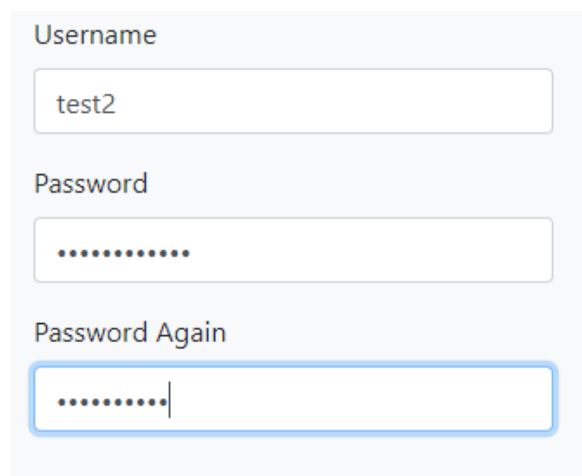
図 21 は maxlength を変更してユーザー名のフォームに 40 文字以上の入力を行った状態である。パスワードは「passdesu1234」とした。図 21 の状態でフォームを送信すると、入力が 0 文字のとき同様に「/failure2Resister」にリダイレクトし図 19 の画面が表示された。このことから、40 文字以上の入力が行われたときも、アカウントの作成に失敗することが確認出来た。

図 21: 40 文字以上の入力

5.3.4 アカウントの作成に失敗した例 (パスワードの不一致)

アカウント作成ページにはパスワードの入力フォームが2つある。入力された2つのパスワードが一致していないと、どちらのパスワードを登録すればよいかわからない。このため、2つのパスワードが一致しないと、アカウント作成に失敗する仕様になっている。

この仕様が実装されていることを確認する。ユーザー名を「test2」、パスワード1を「passdesu1234」、パスワード2を「qwerty5678」としてアカウント作成を行ってみる。図22がフォームにユーザー名とパスワードを入力した状態である。図23に、図22の状態でもう一度フォームを送信した結果を示す。アカウント登録に成功したときは、「/successResister」にリダイレクトしたが、この場合は「/failure2Resister」にリダイレクトする。図23の表示は「入力されたパスワードが一致しなかったため、アカウント作成に失敗した」ということを示している。これより、入力された2つのパスワードが不一致であると、アカウント作成が失敗することが確認できた。



The image shows a web form for account creation. It has three input fields: 'Username' with the value 'test2', 'Password' with masked characters, and 'Password Again' with masked characters. The 'Password Again' field is highlighted with a blue border, indicating it is the current focus or where an error might be occurring.

図 22: パスワードが一致しない入力

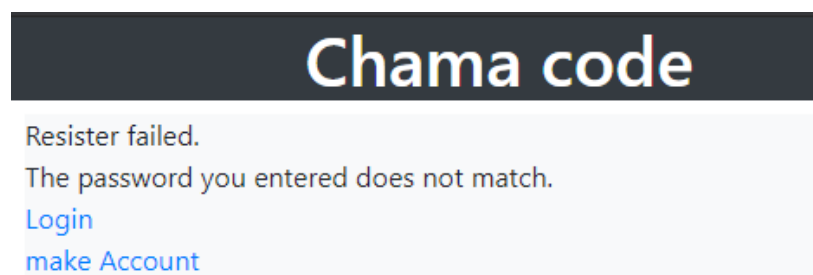


図 23: パスワードの不一致によるエラー画面

5.3.5 実装部分の説明

アカウント作成の実装部分について説明する。リスト9にアカウント作成に関連する部分のコードを示す。リスト9には次に示す6つのリンクにアクセスしたときの処理が記述されている。

- '/createaccount'
- '/newaccount'

- '/successResister'
- '/failure1Resister'
- '/failure2Resister'
- '/failure3Resister'

リスト 9: アカウント作成の実装 (main.rb)

```

1  # 名前の最大長
2  username_max = 40
3  # パスワードの最大長
4  password_max = 40
5
6  # 入力文字列が1文字以上,最大長以下であることを確認する関数
7  # 1 <= inputstr < maxlen : true
8  # else : false
9  def checkstr(inputstr,maxlen)
10     if inputstr.size==0 then
11         return false
12     elsif inputstr.size>maxlen then
13         return false
14     else
15         return true
16     end
17 end
18
19 # アカウント作成フォームを表示
20 get '/createaccount' do
21     erb :makeAccount
22 end
23
24 # アカウント作成管理
25 # 成功 : successResister
26 # 失敗1(usernameが既に存在) : failure1Registerにリダイレクト
27 # 失敗2(pass1,pass2が不一致) : failure2Registerにリダイレクト
28 # 失敗3(入力エラー) : failure3Registerにリダイレクト
29 post '/newaccount' do
30     checkflg=true
31     username = CGI.escapeHTML(params[:uname])
32     pass1 = CGI.escapeHTML(params[:pass1])
33     pass2 = CGI.escapeHTML(params[:pass2])
34     if !checkstr(username,username_max) then
35         checkflg=false
36     elsif !checkstr(pass1,password_max) then
37         checkflg=false
38     elsif !checkstr(pass2,password_max) then
39         checkflg=false
40     end
41
42     if checkflg==false then
43         redirect '/failure3Resister'
44     else
45
46     begin
47         a = User.find(username)
48         redirect '/failure1Resister'
49     rescue => e
50         if pass1==pass2 then
51             r = Random.new
52             algorithm = "1"
53             salt = Digest::MD5.hexdigest(r.bytes(20))
54             hashed = Digest::MD5.hexdigest(salt+pass1)
55             s = User.new
56             s.id = username
57             s.salt = salt
58             s.hashed = hashed

```

```

59     s.algo = algorithm
60     s.save
61     redirect '/successResister'
62   else
63     redirect '/failure2Resister'
64   end
65 end
66 end
67 end
68
69 # アカウ ント作成成功の表示
70 get '/successResister' do
71   erb :successResister
72 end
73
74 # usernameが既に存在しているエラーを表示
75 get '/failure1Resister' do
76   erb :failure1Resister
77 end
78
79 # pass1, pass2が不一致のエラーを表示
80 get '/failure2Resister' do
81   erb :failure2Resister
82 end
83
84 # 入力エラーの表示
85 get '/failure3Resister' do
86   erb :failure3Resister
87 end

```

リスト 9 において「/newaccount」以外の 5 つのリンクにアクセスがきたときの処理は、HTML を返すだけである。それぞれの HTML の内容は付録を参照してほしい。「/newaccount」は「/createaccount」、つまりアカウント作成ページで入力フォームの送信を行ったときにリダイレクトされるページである。フォームの送信によって「/newaccount」にリダイレクトされるのは、付録のリスト 48「/views/makeAccount.erb」の 8 行目で、「action="/newaccount"」という記述によってフォームの送信先を指定しているからである。

「/newaccount」にアクセスが来た時の処理を説明する。リスト 9 では 29 行目から 467 行目である。31 行目から 33 行目ではフォームの内容をサニタイジングして、変数に代入する処理を行っている。フォームの内容は「params[:hoge]」という形式で取得できる。取得する要素の名前は付録のリスト 48「/views/makeAccount.erb」の 11,16,20 行目で宣言している。取得したフォームの内容をそのまま保持することは危険であるからサニタイジングを行う。サニタイジングとは、ユーザーが入力した文字をサーバにとって無害な文字に置き換えることである。HTML のタグとして意味を持つ「<」や「>」の文字はページの見た目を崩す可能性がある。さらに、SQL にとって特殊な文字を入力されると SQL インジェクション攻撃が起き、データベース内容が漏洩する可能性もある。これを防ぐためにサニタイジングをおこなう。サニタイジングには CGI.escapeHTML[3] を用いている。

リスト 9 の 34 行目から 40 行目では、入力文字が不正な文字数でないかの確認を行っている。不正な文字数でないか確認を行う関数として checkstr 関数を作成した。関数の定義は 9 行目から 17 行目である。checkstr 関数は引数として、入力文字列と、その文字列が許されている最大の長さを引数とする。戻り値は不正な入力の場合 false、そうでない場合は true である。checkstr 関数の内部では、引数 inputsize のサイズを確認することで不正な文字列か否かの判定を行っている。関数の呼び出し側では、ユーザー名、パスワードをそれぞれ checkstr 関数に投入している。ユーザー名、パスワードの最大文字数はリスト 9 の 2 行目および 4 行目で定義している。文字列のチェックが終了したから、不正な文字列の場合は「/failure3Resister」にリダイレクトして処理を終了する。不正な入力があった場合 checkflg が false になっているから、42 行目の if 文で「/failure3Resister」にリダイレクトされる。これによって不正な文字列が入力されたときにアカウント作成に失敗する処理を実現している。

不正な入力を検知して「/failure3Resister」にリダイレクトする処理は、ログインや記事の作成、編集でも

同様に用いている。本節で、不正な入力に対する処理と実装について説明したため、以後の章では、不正な入力に対する実行結果は省略する。

入力文字列の不正を取り除き、サニタイジングによる無害化を行ったからデータベースへの登録を行う。リスト 9 では 46 行目から 65 行目の処理である。まず、47 行目で入力されたユーザー名で User データベースに検索をかける。もしユーザー名が存在すれば、エラーが起きずに変数 a に該当するレコードが代入される。ユーザー名は固有の値であるから、この場合、48 行目に示すように「/failure1Resister」にリダイレクトする処理を行う。これによって、ユーザー名が重複したときにアカウント作成が失敗する処理を実現している。もし、ユーザー名が存在しなければ SQLite3 からは「該当するレコードがありません」という意味のエラーが返ってくる。これを受け取る部分が 49 行目である。エラーを受け取った場合はアカウントが重複していないから、パスワードの一致を確認してアカウント作成の処理を行う。リスト 9 の 50 行目で pass1 と pass2 が等しいか確認し、等しくないときは「/failure2Resister」にリダイレクトする (63 行目)。pass1 と pass2 が等しいときは、アカウント作成の処理を行う。

アカウント作成の処理を行っているのは、リスト 9 の 51 行目から 60 行目である。52 行目ではハッシュアルゴリズムを登録している。ここでは、ハッシュアルゴリズムは MD5 のみを使用するから、変数 algorithm は常に 1 である。53 行目では、20 文字のランダムな salt を生成している。そして 54 行目で、生成した salt とパスワード pass1 を結合してハッシュ値を計算している。これでデータベースに登録に必要なデータが準備できたからデータベースに登録する。データベースにデータを登録しているのは、リスト 9 の 55 行目から 60 行目である。

5.4 ログイン/ログアウト

本節では、Web アプリケーションへのログイン/ログアウトについて、次に示す 4 つの内容を述べる。

1. ログインの方法
2. ログインに失敗した例
3. ログアウトの方法
4. サニタイジング処理の確認
5. 実装部分の説明

5.4.1 ログインの方法

ログインの方法について説明する。ここでは、5.3 章で作成したアカウントでログインを行う。5.3 章で作成したアカウントはユーザー名が「test」、パスワードが「abcdefg1234」である。まず、ログインページにアクセスする。ログインページは 5.1 章で示した「/login」のページである。図 24 にログインページのフォームにユーザー名とパスワードを入力した状態を示す。図 24 の状態で「Login」のボタンを押すとログインの処理が行われて「/contentspage」のページにリダイレクトされる。図 25 に「Login」ボタンを押したときの実行結果を示す。図 25 はログインが成功して「/contentspage」にリダイレクトした状態である。

Chama code

Login

ユーザー名とパスワードを入力してください。

Username

Password

Login Reset

図 24: ログインフォームに入力した状態

Chama code

Welcome test !

[MyArticle](#) [ランキング](#) [初めての方へ](#)

タイトル名:

[Logout](#)

図 25: 「Login」をクリックしたときの処理

ログインせずに「/contentspage」にアクセスした場合の処理を確認する。ログインせずに「/contentspage」にアクセスした場合、コンテンツページを表示せず、サービスへのログインを促す表示を行う仕様になっている。ログインせずに「/contentspage」にアクセスした場合の結果を図 26 に示す。図 26 の画面が表示は「バッドリクエストです。先にサービスにログインしてください。」という意味である。このことから、ログインせずにコンテンツページにアクセスすると、コンテンツページではなくログインを促す画面が表示され仕様になっていることがわかる。

Chama code

Bad request. Please login this service first.

[Back to login screen.](#)

図 26: ログインせずにコンテンツページにアクセスした結果

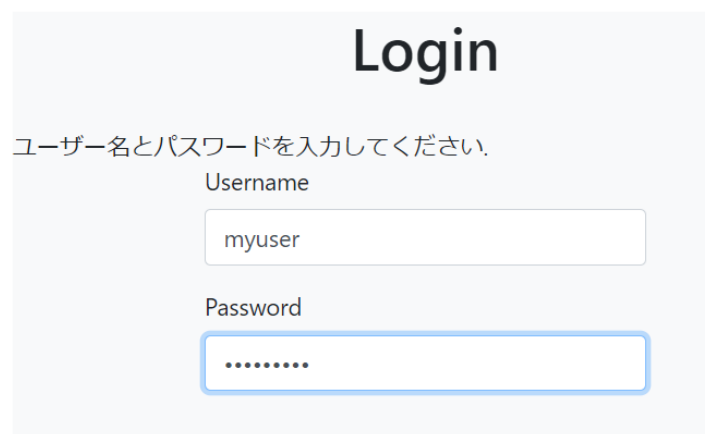
5.4.2 ログインに失敗する例

存在しないユーザー名でログインしようとした場合と、ユーザー名は存在するがパスワードが間違っている場合にログインできないことを確認する。この仕様はログイン機能としては当然の仕様であるかもしれないが、確認を行う。まず、存在しないユーザー名でログインしようとした場合の処理を確認する。いま、データベースに登録されているユーザーは図 27 のようになっている。図 27 からユーザーとして「test」というユーザーが登録されていることがわかる。ここでは、存在しないユーザーとして、ユーザー名は「myuser」、パスワードは「mypass111」でログインしてみる。

```
sqlite> select * from users;  
test|45b05f7db22ae7c335b9f70ca47c6204|fcaa6e13910d5e6cb0bb96bf363c3ab2|1  
sqlite> █
```

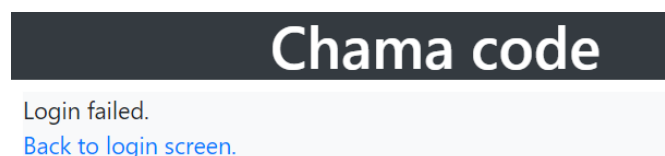
図 27: users テーブルの内容

図 28 はログインフォームに存在しないアカウントを入力した状態である。図 29 に図 28 の状態で「Login」ボタンを押した結果を示す。ログインに成功した場合は「/contentspage」にリダイレクトしたが、存在しないユーザーでログインしようとした場合は「/failure」にリダイレクトする。図 29 の画面表示は「ログインに失敗しました」という意味である。



The image shows a login form titled "Login". Below the title, it says "ユーザー名とパスワードを入力してください。" (Please enter your username and password). There are two input fields: "Username" with the value "myuser" and "Password" with masked characters ".....".

図 28: 存在しないアカウントを入力した状態



The image shows a dark header with the text "Chama code". Below it, on a light background, it says "Login failed." followed by a blue link "Back to login screen."

図 29: ログイン失敗

次にユーザー名は存在するが、パスワードが間違っている場合の処理を確認する。図 27 から、「test」というユーザー名が存在することが分かっているから、パスワードを適当なもの（例えば「zxcv0987」）でログインしてみる。図 30 はユーザー名およびパスワードをログインフォームに入力した状態である。図 30 の状態でログインをすると、図 29 と同様に「/failure」にリダイレクトした。これによって、存在しないユーザー名や、間違ったパスワードでログインできないことが確認できた。

The screenshot shows a web interface with a dark header bar containing the text "Chama code". Below the header, the word "Login" is centered in a large, bold font. Underneath "Login", there is a line of text: "ユーザー名とパスワードを入力してください。". Below this text are two input fields. The first field is labeled "Username" and contains the text "test". The second field is labeled "Password" and contains a series of dots ".....". The password field has a blue border, indicating it is the active element.

図 30: ログイン失敗 (パスワードが間違っている場合)

5.4.3 ログアウトの方法

ログアウトの方法について説明する。ログアウトは、コンテンツページまたは,MyArticle ページから行うことができる。図 31 がコンテンツページ, 図 32 が MyArticle ページである。ログアウトはコンテンツページ,MyArticle ページのどちらにおいても,「Logout」というリンクから行うことができる。図 33 に「Logout」をクリックしたときの処理を示す。ログアウトが完了すると,「/logout」に自動でリダイレクトされる。ページの内容は,「ログアウトしました」という意味である。これより,ログアウトが行えたことが確認できる。

The screenshot shows a web interface with a dark header bar containing the text "Chama code". Below the header, the text "Welcome test !" is displayed in a large, bold font, with "test" in green. Below this text, there are three links: "MyArticle", "ランキング", and "初めての方へ". Below these links, there is a text input field labeled "タイトル名:" with the placeholder text "find article...". At the bottom of the page, there is a link labeled "Logout".

図 31: コンテンツページ (再掲)



図 32: MyArticle ページ

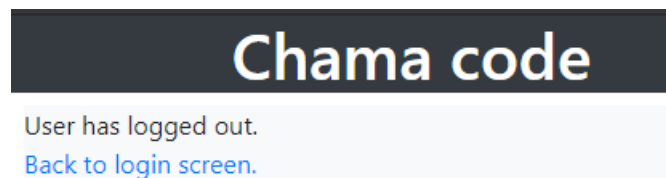


図 33: ログアウト時の処理

5.4.4 サニタイジング処理の確認

アカウント作成およびログインで、ユーザー名のサニタイジングが行われていることを確認する。ログインしたときのユーザー名は、コンテンツページ (図 31) で「Welcome ユーザー名!」というように表示される。これを利用して、サニタイジングが行われていることを確認する。実験としてユーザー名が「<h6>big</h6>」、パスワードが「wed1357」のユーザーを作成してログインしてみる。もし、サニタイジングの処理が行われていなければ h タグによってユーザー名を表示する部分の文字の大きさが変更され、「big」という文字だけが表示される。サニタイジングが行われていれば「<h6>big</h6>」という文字がそのまま表示される。

図 34 にユーザー名が「<h6>big</h6>」、パスワードが「wed1357」のユーザーを作成して、ログインした結果を示す。図 34 から、ユーザー名が「<h6>big</h6>」そのまま表示されていることが読み取れる。このことから、ユーザー名のサニタイジング処理が行われていることが確認できる。



図 34: ユーザー名のサニタイジング処理の確認

5.4.5 実装部分の説明

ログイン/ログアウト処理の実装部分について説明する. リスト 10 にログイン/ログアウトの実装部分のコードを示す. リスト 10 には, 次に示す 5 つのリンクにアクセスしたときの処理が記述されている.

- '/login'
- '/auth'
- '/failure'
- '/contentspage'
- '/logout'

リスト 10: ログイン/ログアウトの実装 (main.rb)

```
1 # パスワードを確認する関数
2 # 入力とDBが一致 -> true
3 # 不一致 -> false
4 def checkpass(trial_username, trial_passwd)
5   # Search recorded info
6   begin
7     a = User.find(trial_username)
8     db_username = a.username
9     db_salt = a.salt
10    db_hashed = a.hash
11    db_algo = a.algo
12  rescue => e
13    puts "User #{trial_username} is not found."
14    puts e.message
15    return false
16  end
17
18  # Generate a hashed value
19  if db_algo == "1"
20    trial_hashed = Digest::MD5.hexdigest(db_salt+trial_passwd)
21  else
22    puts "Unknown algorithm is used for user #{trial_username}."
23    return false
24  end
25
26  if db_hashed == trial_hashed
27    return true
28  else
```

```

29     return false
30 end
31 end
32
33 # ログインフォームを表示
34 get '/login' do
35     erb :loginscr
36 end
37
38 # ログイン管理
39 # 成功 : contentspageにリダイレクト
40 # 失敗 : failureにリダイレクト
41 post '/auth' do
42     checkflg=true
43     username = CGI.escapeHTML(params[:uname])
44     pass = CGI.escapeHTML(params[:pass])
45     if !checkstr(username,username_max) then
46         checkflg=false
47     elsif !checkstr(pass,password_max) then
48         checkflg=false
49     end
50
51     if checkflg==false then
52         redirect '/failure3Resister'
53     else
54         if(checkpass(username,pass))
55             session[:login_flag] = true
56             session[:username] = username
57             redirect '/contentspage'
58         else
59             session[:login_flag] = false
60             redirect '/failure'
61         end
62     end
63 end
64
65 # ログイン失敗の表示
66 get '/failure' do
67     erb :failure
68 end
69
70 # ログイン必須
71 # コンテンツページを表示
72 # 未ログイン時 : badrequestにリダイレクト
73 get '/contentspage' do
74     if (session[:login_flag]==true)
75         @a = session[:username]
76         erb :contents
77     else
78         erb :badrequest
79     end
80 end
81
82 # ログアウトの処理
83 get '/logout' do
84     session.clear
85     erb :logout
86 end

```

リスト 10 の内容を説明する。リスト 10 において、「/login」および「/failure」にアクセスがきた場合の処理は、HTML を返すだけである。HTML の内容は付録を参照してほしい。「/auth」は「login」、つまりログインフォームの送信を行ったときにリダイレクトされ、ログイン処理を行うページである。

「/auth」での処理について説明する。リスト 10 では 41 行目から 63 行目である。42 行目から 52 行目では、アカウントの作成処理と同様に、入力文字列の不正をチェックする処理と、サニタイジングの処理を行っている。入力文字列に問題がある場合は「/failure3Resister」にリダイレクトする。入力されたパスワードと、データベースのパスワードのハッシュ値の照合はリスト 10 の 54 行目の checkpass 関数で行っている。checkpass

関数の定義はリスト 10 の 4 行目から 31 行目である。checkpass 関数はユーザー名 username とパスワード pass を引数として受け取り、パスワードが一致した場合 true、一致しない場合は false を戻り値として返す。

パスワードの照合に成功した場合、checkpass 関数から true が返ってくるから、リスト 10 の 55 行目から 57 行目の処理が行われる。55 行目および 56 行目では、cookie にログインフラグを true(成功) とユーザー名を書き込む処理を行っている。この処理によって、ログアウトを行うまでログイン状態が継続する。パスワードの照合が失敗した場合、checkpass 関数から false が返ってくるから、リスト 10 の 59 行目および 60 行目の処理が行われる。59 行目では、cookie にログインフラグを false(失敗) で記述し、60 行目では「/failure」にリダイレクトする処理を行っている。これによってログインに失敗したページが表示される。

コンテンツページ「/contentspage」にアクセスがきたときの処理について説明する。リスト 10 では 73 行目から 80 行目である。74 行目では cookie からログインフラグを取得し、true であることを確認している。コンテンツページはログイン済みのユーザーのみアクセス可能である必要がある。このためログインフラグを確認している。もし、ログインフラグが false であれば badrequest のページを表示する (78 行目)。ログインフラグが true であればコンテンツページでユーザー名を表示するために、変数@a に cookie から取得したユーザー名を代入する処理を行っている。

ログアウトの処理について説明する。リスト 10 では 83 行目から 86 行目である。cookie の消去は 84 行目の「session.clear」によって行われる。cookie が削除されると、ログイン状態が保持されていない状態になるため、ログアウトした状態になる。

ログインフラグの確認は、Web アプリケーションのメインコンテンツである記事の作成、削除を代表とするページでも行っている。ページのアクセス権限は、ソースコードの「get '/hoge' do」という記述の上の行のコメントアウトに記述している。例えばリスト 10 では「/contentspage」の場合、コメントアウトで「ログイン必須」と記述し、「/logout」の場合は何も記述していない。以下、実行結果として非ログイン状態でアクセスしたときの処理については説明を省略するが、アクセス権限はソースコードのコメントアウトとしてすべて記述しているため、これを参照してほしい。また、サニタイジングが実装されていることの確認と実装方法についても本節で説明した。このため、記事の作成や編集の部分でも、同様の処理を行っているが、確認は省略する。

5.5 パスワードの変更

本節では、パスワードを変更する機能について、次に示す 4 つの内容を述べる。

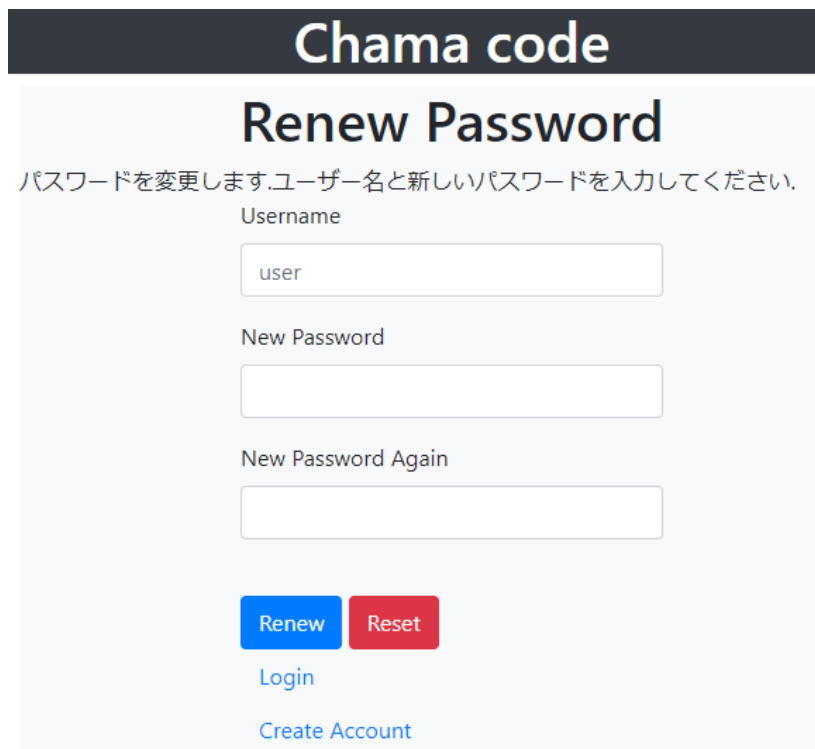
1. パスワードの変更方法
2. パスワードの変更に失敗する例 (ユーザーが存在しない)
3. パスワードの変更に失敗する例 (パスワードの不一致)
4. 実装部分の説明

5.5.1 パスワードの変更方法

パスワードの変更方法について説明する。5.3 章で作成したユーザー名が「test」、パスワードが「abcdefg1234」のユーザーのパスワードを「hijk5678」に変更してみる。まず、パスワード変更フォームにアクセスする。パスワード変更ページはログインページまたはアカウント作成ページの「Forget Password」をのリンクをクリックするか、「/forgetpass」にアクセスすることで表示される。図 35 がパスワード変更ページである。パスワードの変更には、ユーザー名とパスワードを入力する必要がある。パスワードはアカウント作成のときと同様に 2 回入力する。

変更前のパスワードは入力を必要としない仕様とした。この場合、ユーザー名だけ分かればパスワードの

変更ができるため、ユーザーを乗っ取ることが簡単にできる。乗っ取りを防ぐためには、メールアドレスとの連携が効果的であると考えられるが、開発時間と処理の複雑さを考慮してこのような仕様になっている。



Chama code

Renew Password

パスワードを変更します.ユーザー名と新しいパスワードを入力してください.

Username

New Password

New Password Again

[Renew](#) [Reset](#)

[Login](#)

[Create Account](#)

図 35: パスワードの変更フォーム

パスワードの変更を行う。図 36 はパスワード変更フォームにユーザー名と新しいパスワードを入力した状態である。図 36 の状態で「Renew」のボタンをクリックすると、パスワードを更新する処理が行われる。図 37 に「Renew」ボタンをクリックしたときの実行結果を示す。パスワードの更新に成功すると「/successRenewpass」にリダイレクトされる。図 37 のページの意味は、「パスワードの更新に成功しました」という意味である。

Chama code

Renew Password

パスワードを変更します.ユーザー名と新しいパスワードを入力してください.

Username

New Password

New Password Again

[Renew](#) [Reset](#)

[Login](#)

[Create Account](#)

図 36: パスワード変更フォームへの入力

Chama code

Renew Password succeeded.

[Login](#)

図 37: パスワードの更新に成功した画面

5.5.2 パスワードの変更に失敗する例 (ユーザーが存在しない)

存在しないユーザーのパスワードを変更しようとした場合は、パスワードの変更を行わず、変更に失敗した趣旨のページを表示する仕様になっている。この仕様を確認するために、存在しないユーザー名「myuser」のパスワードを適当なものに変更してみる。図 38 はパスワード変更フォームで、存在しないユーザー名と適当なパスワードを入力した状態である。図 39 に、図 38 の状態で「Renew」を押した結果を示す。パスワードの変更に成功した場合「/successRenewpass」にリダイレクトするが、この場合「/unknownUser」にリダイレクトする。図 39 の表示の意味は「パスワードの変更に失敗しました。入力したユーザーは存在しません。」という意味である。これによって、存在しないユーザーのパスワードを変更しようすると、失敗することが確認できた。

Chama code

Renew Password

パスワードを変更します.ユーザー名と新しいパスワードを入力してください.

Username

New Password

New Password Again

Renew Reset

図 38: 存在しないユーザーのパスワードを変更

Chama code

Renew Password failed.
No user entered was found.
[Login make Account](#)

図 39: パスワードの変更失敗

5.5.3 パスワードの変更に失敗する例 (パスワードの不一致)

パスワードの変更を行うときに、パスワードの入力欄が2つあることは図 35 で確認した。パスワードの入力欄にそれぞれ別の文字列を入力したときに、パスワードの変更が失敗する仕様を確認する。ユーザー名を「test」、パスワード1を「qwerty1234」、パスワード2を「asdfg4567」としてパスワードの変更を行ってみる。図 40 にユーザー名とパスワードを入力した状態を示す。図 40 の状態で「Renew」を押すと、図 23 の画面にリダイレクトする。これによって、パスワードが不一致の場合、パスワードの更新が行われないことが確認できた。

Chama code

Renew Password

パスワードを変更します。ユーザー名と新しいパスワードを入力してください。

Username

test

New Password

New Password Again

Renew

Reset

図 40: 入力したパスワードが不一致の状態

5.5.4 実装部分の説明

パスワードの変更処理の実装部分について説明する。リスト 11 にパスワードの変更処理の実装部分のコードを示す。リスト 11 には、次に示す 4 つのリンクにアクセスしたときの処理が記述されている。

- `’/forgetpass’`
- `’/newpass’`
- `’/successRenewpass’`
- `’/unknownUser’`

リスト 11: パスワード変更の実装 (main.rb)

```

1 # パスワードの再発行フォームを表示
2 get '/forgetpass' do
3   erb :forgetpass
4 end
5
6 # パスワードを再発行する処理
7 # 成功 : successRenewpassにリダイレクト
8 # 失敗(usernameが存在しない) : unknownUserにリダイレクト
9 # 失敗(pass1,pass2が不一致) : failure2Resisterにリダイレクト
10 post '/newpass' do
11   checkflg=true
12   username = CGI.escapeHTML(params[:uname])
13   pass1 = CGI.escapeHTML(params[:pass1])
14   pass2 = CGI.escapeHTML(params[:pass2])
15   if !checkstr(username,username_max) then
16     checkflg=false
17   elsif !checkstr(pass1,password_max) then
18     checkflg=false
19   elsif !checkstr(pass2,password_max) then
20     checkflg=false
21   end
22
23   if checkflg==false then
24     redirect '/failure3Resister'
25   else

```

```

26   begin
27     a = User.find(username)
28     if pass1==pass2 then
29       r = Random.new
30       algorithm = "1"
31       salt = Digest::MD5.hexdigest(r.bytes(20))
32       hashed = Digest::MD5.hexdigest(salt+pass1)
33       a.salt = salt
34       a.hashed = hashed
35       a.algo = algorithm
36       a.save
37       redirect '/successRenewpass'
38     else
39       redirect '/failure2Resister'
40     end
41     rescue => e
42       redirect '/unknownUser'
43     end
44   end
45 end
46
47 # パスワードの再発行成功を表示
48 get '/successRenewpass' do
49   erb :successRenewpass
50 end
51
52 # usernameが見つからないことを表示
53 get '/unknownUser' do
54   erb :unknownUser
55 end

```

リスト 11 の内容について説明する。リスト 11 において「/forgetpass」,「/successRenewpass」,「/unknownUser」の 3 つにアクセスがきた場合の処理は、HTML を返すだけである。HTML の内容は付録を参照してほしい。「/newpass」は「/forgetpass」のフォームを送信したときにリダイレクトされ、パスワードを更新する処理を行うページである。

パスワードの更新処理について説明する。リスト 11 では 10 行目から 45 行目である。11 行目から 24 行目では、アカウントの作成やログインの処理と同様に、入力文字列のサニタイジングや文字数のチェックの処理を行っている。入力文字列に不正がない場合は、入力された 2 つのパスワードが一致しているか確認する。この処理を行っているのは、リスト 11 の 28 行目である。29 行目から 36 行目では、データベースから該当するユーザーを検索し、パスワードを更新する処理を行っている。ユーザーが存在しない場合は 42 行目にあるように、「/unknownUser」にリダイレクトして処理を終了する。これらによって、パスワードの更新を実現している。

5.6 記事の作成

本節では Web アプリケーションのメインコンテンツである記事の作成について、次に示す 2 つの内容を述べる。

1. 記事の作成方法
2. 実装部分の説明

5.6.1 記事の作成方法

記事の作成方法について説明する。ここでは、「test」というユーザーでログインして記事を作成する。記事を作成するためには、まず MyArticle のページ「/myarticle」にアクセスする。MyArticle のページは図 41

のようなページである。記事の新規作成フォームは図 41 の「New Article」ボタンを押すと、「/newarticle」にリダイレクトして開かれる。「New Article」ボタンは文字色が白、背景色が緑である。



図 41: MyArticle ページ (再掲)

図 42 および図 43 に「New Article」ボタンを押したときの結果を示す。記事の内容、ソースコード、実行結果はそれぞれ 1000 字まで入力できるようにするために、フォームの入力欄を広くしている。このため、一枚の画像に収めることができない。図 42 と図 43 は縦に繋がっているものとして見てほしい。

Chama code
新しい記事を作成します!
<input type="checkbox"/> 公開 タイトル
<input type="text" value="title"/>
記事の内容
<div></div>
ソースコード
<div></div>

図 42: 記事の新規作成ページ 1

実行結果

Submit

図 43: 記事の新規作成ページ 2

新しい記事として, 図 44 および図 45 に示す記事を作成した. この記事は図 44 で「公開」というチェックボックスにチェックを入れているため, 記事を提出すると公開状態, つまり他のユーザーに見える状態で登録が行われる. もしチェックボックスにチェックを入れなければ, 非公開, つまり記事を作成したユーザーのみが閲覧できる状態で登録が行われる. 記事の入力が完了したら, 図 45 の最下部にある「Submit」というボタンを押すことで, データベースに記事を登録する処理が行われ, 記事が作成される.

Chama code

新しい記事を作成します!

☒ 公開

タイトル

はじめてのC言語

記事の内容

C言語のHello Worldのコードです.

ソースコード

```
#include<stdio.h>

int main(void){
    printf("hello world\n");
    return 0;
}
```

図 44: 記事の作成例 1

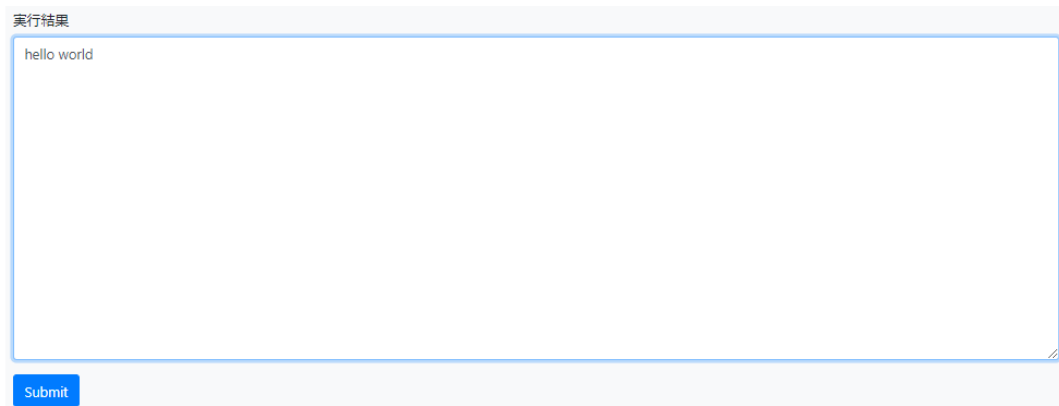


図 45: 記事の作成例 2

図 46 に「Submit」ボタンを押したときの実行結果を示す。記事の登録に成功すると「/successarticle」にリダイレクトされる。図 46 の表示は、「記事の提出に成功しました」という意味である。

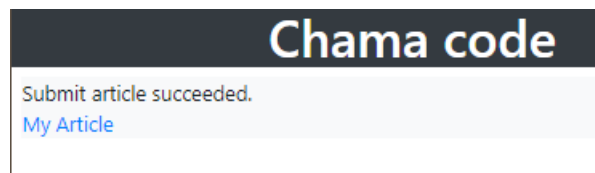


図 46: 記事の登録に成功した画面

5.6.2 実装部分の説明

記事の作成の実装部分について説明する。リスト 12 に記事作成の実装部分のコードを示す。リスト 12 に、次に示す 3 つのリンクにアクセスしたときの処理が記述されている。

- '/newarticle'
- '/autharticle'
- '/successarticle'

リスト 12: 記事作成の実装 (main.rb)

```
1 # 記事の最大長
2 article_max = 1000
3
4 # ログイン必須
5 # 記事の新規作成フォームを表示
6 # 未ログイン時 : badrequestにリダイレクト
7 get '/newarticle' do
8   if (session[:login_flag]==true)
9     erb :newarticle
10   else
11     erb :badrequest
12   end
13 end
14
15 # ログイン必須
16 # 記事の新規作成処理
17 # 成功 : successarticleにリダイレクト
```

```

18 # 失敗 : !
19 # 未ログイン時 : badrequestにリダイレクト
20 post '/autharticle' do
21   if (session[:login_flag]==true)
22     # サニタイジング処理を行う場所
23     checkflg=true
24     title = CGI.escapeHTML(params[:title])
25     description = CGI.escapeHTML(params[:description])
26     code = CGI.escapeHTML(params[:code])
27     result = CGI.escapeHTML(params[:result])
28     if !checkstr(title,title_max) then
29       checkflg=false
30     elsif !checkstr(description,article_max) then
31       checkflg=false
32     elsif !checkstr(code,article_max) then
33       checkflg=false
34     elsif !checkstr(result,article_max) then
35       checkflg=false
36     end
37
38     if checkflg==false then
39       redirect '/failure3Resister'
40     else
41
42       authtime = Time.now.strftime("%Y/%m/%d %T")
43       @a = session[:username]
44       r = Random.new
45       s = Content.new
46       s.id = Digest::MD5.hexdigest(r.bytes(40))
47       s.username = @a
48       s.date = authtime
49       s.title = title
50       s.description = description
51       s.code = code
52       s.result = result
53       s.good=0
54
55       if params[:publicbutton] then
56         s.open=1
57       else
58         s.open=0
59       end
60       s.save
61       redirect '/successarticle'
62     end
63   else
64     erb :badrequest
65   end
66 end
67
68 # ログイン必須
69 # 記事の新規作成成功を表示
70 # 未ログイン時 : badrequestにリダイレクト
71 get '/successarticle' do
72   if (session[:login_flag]==true)
73     erb :successarticle
74   else
75     erb :badrequest
76   end
77 end

```

リスト 12 の内容について説明する. リスト 12 において「/newarticle」および「/successarticle」にアクセスがきた場合の処理は,HTML を返すだけである.HTML の内容は付録を参照してほしい.「/autharticle」は「/newarticle」のフォームを送信したときにリダイレクトされ,記事の登録処理を行うページである.

記事の登録処理について説明する. リスト 12 では 20 行目から 66 行目である.21 行目から 39 行目では,セッション管理, 入力文字列のチェック, サニタイジングの 3 つの処理を行っている.43 行目から 60 行目では contents テーブルに記事を登録する処理を行っている. 55 行目から 59 行目では, チェックボックスのフォー

ムの「publicbutton」というキーから記事を公開にするか、非公開にするかを保持する open カラムの値を決定し、その値を格納する処理を行っている。

5.7 記事の閲覧

本節では、自分の記事の閲覧について次に示す 4 つの内容を述べる。

1. 自分の記事を見る方法
2. ユーザーによって表示内容が変化することの確認
3. 細かい仕様の説明
4. 実装部分の説明

5.7.1 自分の記事を見る方法

記事を見る方法を説明するために、ユーザー「test」に記事を追加する。非公開の記事として、タイトル「Python の Hello World」、内容は適当なもので記事を作成した。自分が作成した記事を見る方法について説明する。自分が作成した記事の一覧は MyArticle ページ「/myarticle」で見ることができる。図 47 にユーザー「test」の MeArticle ページを示す。図 47 から、ユーザー「test」の記事として「はじめての C 言語」と「Python の Hello World」の 2 つが表示されていることがわかる。

Chama code

MyArticle

test さんの記事一覧です

New Article

Contents

タイトル	公開状況	更新日時	いいね数			
はじめてのC言語	公開	2020/12/18 18:14:48	0	詳細	編集	削除
PythonのHello World	非公開	2020/12/18 19:22:28	0	詳細	編集	削除

Logout

図 47: 記事の一覧の例 (ユーザー「test」)

図 47 では、自分の記事の一覧に、タイトル、公開状況、更新日時、いいね数の 4 つ記事の情報が表示されている。一方で、記事の内容 (説明, ソースコード, 実行結果) は表示されていない。アプリの利用者側としては、自分が作成した記事の内容を閲覧したい場合があると考えられる。このニーズに答えるために、閲覧したい記事に対応する「詳細」ボタンを押すと、記事の内容が表示できるようになっている。「詳細」ボタンは文字色が白、背景色が青になっている。

図 48 および図 49 に詳細をクリックしたときの実行結果を示す。図 48 は「はじめての C 言語」という記

事の「詳細」ボタンを押したときの処理であり、図 49 は「Python の Hello World」という記事の「詳細」ボタンを押したときの処理である。どちらのページもリンクは「/detail」である。図 48 および図 49 から、クリックした記事に対応した記事の詳細が表示されていることがわかる。

Chama code

[戻る](#)

はじめてのC言語 @test

更新日: 2020/12/18 18:14:48
いいね数: 0

C言語のHello Worldのコードです。

ソースコード

```
#include<stdio.h>

int main(void){
    printf("hello world\n");
    return 0;
}
```

実行結果

```
hello world
```

[戻る](#)

図 48: 自分の記事の閲覧例 1

Chama code

[戻る](#)

PythonのHello World @test

更新日: 2020/12/18 19:22:28
いいね数: 0

PythonのHello Worldのコードです。

ソースコード

```
print("hello world")
```

実行結果

```
hello world
```

[戻る](#)

図 49: 自分の記事の閲覧例 2

5.7.2 ユーザーによって表示内容が変化することの確認

ユーザーによって表示内容が変化することの確認する。確認はユーザー「test」とは別のユーザーを作成し、そのユーザーで記事を作成することで行う。別のユーザーとしてユーザー「pfy」を作成する。そしてユーザー「pfy」で、非公開記事として「Say hello to Java」、公開記事として「Ruby にまつわるエトセトラ」を作成する。内容は適当なものとする。図 50 にユーザー「pfy」の MyArticle ページを示す。図 50 からユーザー「pfy」の場合、「Say hello to Java」と「Ruby にまつわるエトセトラ」という 2 つの記事が表示されていることがわかる。このことから MyArticle ページで、ユーザーごとに自分の記事の一覧が表示されることが確認できる。

Chama code

MyArticle

pfy さんの記事一覧です

[Contents](#)

[New Article](#)

タイトル	公開状況	更新日時	いいね数			
Say hello to Java	非公開	2020/12/18 20:11:11	0	詳細	編集	削除
Rubyにまつわるエトセトラ	公開	2020/12/18 20:12:15	0	詳細	編集	削除

[Logout](#)

図 50: 記事の一覧の例 (ユーザー「pfy」)

「詳細」ボタンを押すと記事の詳細が表示されることも確認する。図 51 および図 52 に詳細を押したときの実行結果を示す。図 51 は「Say hello to Java」という記事の「詳細」ボタンを押したときの処理であり、図 52 は「Ruby にまつわるエトセトラ」という記事の「詳細」ボタンを押したときの処理である。図 51 および図 52 から、ユーザー「pfy」の場合もクリックした記事に対応した記事の詳細が表示されていることがわかる。

Chama code

[戻る](#)

Say hello to Java @pfy

更新日 : 2020/12/18 20:11:11
いいね数 : 0

JavaのHello Worldのコードです.

ソースコード

```
public class HelloWorld{
    public static void main(String[] args){
        System.out.println("hello world");
    }
}
```

実行結果

```
hello world
```

[戻る](#)

図 51: 自分の記事の閲覧例 3

Chama code

[戻る](#)

Rubyにまつわるエトセトラ @pfy

更新日 : 2020/12/18 20:12:15
いいね数 : 0

Rubyのhello worldのコードです.

ソースコード

```
puts "hello world"
```

実行結果

```
hello world
```

[戻る](#)

図 52: 自分の記事の閲覧例 4

5.7.3 細かい仕様の説明

記事の一覧表示の細かい仕様として次の 2 つの仕様がある。本項では、この 2 つの仕様について実装できていることを確認する。

1. タイトルが 20 文字を超えたときの表示の仕様
2. 他のユーザーが「/detail」にアクセスした場合の仕様

まず、タイトルが 20 文字を超えたときの表示の仕様について確認する。タイトルは 60 文字まで入力可能だが、記事の一覧として 60 文字全てを表示すると見映えが悪くなってしまう。そこでタイトルが 20 文字を超えている場合、20 文字目までのタイトルと「...」という文字を表示する仕様にした。実装の確認としてユーザー「test」でタイトルが「ルンゲクッタ法による連立微分方程式および高階微分方程式の数値計算」という記事を投稿してみる。このタイトルの文字数は 32 文字であるため、仕様が実装されていれば「ルンゲクッタ法による連立微分方程式および...」と表示される。

図 53 に長いタイトルを入力したときの、記事一覧での表示結果を示す。タイトルの表示は「ルンゲクッタ法による連立微分方程式および...」というふうに表示されているから、この仕様が実装できていることがわかる。



Chama code						
MyArticle						
test さんの記事一覧です				New Article		
Contents						
タイトル	公開状況	更新日時	いいね数			
はじめてのC言語	公開	2020/12/18 18:14:48	0	詳細	編集	削除
PythonのHello World	非公開	2020/12/18 19:22:28	0	詳細	編集	削除
ルンゲクッタ法による連立微分方程式および...	非公開	2020/12/18 21:03:23	0	詳細	編集	削除

[Logout](#)

図 53: タイトルが長い場合の表示

次にあるユーザーが「/detail」にアクセスしているときに、他のユーザーが「/detail」にアクセスしても、無関係に処理が行われることを確認する。この仕様を確認するために、まずユーザー 1 の端末でユーザー「test」でログインしたうえで、「はじめての C 言語」にアクセスする。次に、ユーザー 2 の端末でユーザー「pfy」でログインしたうえで「Say hello to Java」にアクセスする。もし、ユーザー 1 とユーザー 2 の処理が無関係に行われているなら、どちらのページも問題なく表示されるはずである。実際に実験を行ったところ、どちらのページも問題なく表示できた。これより、同じ「/detail」にアクセスしても、処理が無関係に行われていることが確認できる。

5.7.4 実装部分の説明

自分の記事の閲覧の実装部分について説明する. リスト 13 およびリスト 14 に自分の記事の閲覧の実装部分のコードを示す. リスト 13 は main.rb から自分の記事の閲覧を行う部分のコードを抜き出したものである. リスト 13 には「/myarticle」および「/detail」にアクセスがきたときの処理が記述されている. リスト 14 は「/myarticle」のページの HTML である.

リスト 13: 自分の記事の閲覧の実装 (main.rb)

```
1 # ログイン必須
2 # ログインしているユーザーの記事だけを表示
3 # 未ログイン時 : badrequestにリダイレクト
4 get '/myarticle' do
5   @isarticle=0
6   if (session[:login_flag]==true)
7     @a = session[:username]
8     # userの記事の件数取得
9     @c = Content.select('*').where('username == \''+@a+'\'').count
10
11     if @c>=1 then # 記事があるとき読み込み
12       @s = Content.select('*').where('username == \''+@a+'\'')
13       @isarticle=1
14     end
15
16     erb :myarticle
17   else
18     erb :badrequest
19   end
20 end
21
22 # ログイン必須
23 # 記事の詳細を表示
24 # 未ログイン時 : badrequestにリダイレクト
25 post '/detail' do
26   if (session[:login_flag]==true)
27     @cookieu = session[:username]
28     @a = Content.find(params[:id])
29     if @cookieu.eql?(@a.username) then
30       @user_flg=true
31     else
32       @user_flg=false
33     end
34
35     erb :detail
36   else
37     erb :badrequest
38   end
39 end
```

リスト 14: 自分の記事の閲覧の実装 (myarticle.erb)

```
1 <script src="deleteCheck.js"></script>
2 <center>
3 <h1>MyArticle</h1>
4 </center>
5 <span class="row mx-2">
6 <h2><font color="#32cd32"><%= @a %> </font>さんの記事一覧です</h2>
7 </span>
8
9 <div class="text-right mr-5" >
10 <a class="btn btn-success" data-toggle="collapse" href="/newarticle" role="button"
11 aria-expanded="false" aria-controls="collapseExample">
12 New Article
13 </a>
14 </div>
15
```

```

16 <a class="btn btn-link" data-toggle="collapse" href="/contentspage" role="button"
17 aria-expanded="false" aria-controls="collapseExample">
18 Contents
19 </a>
20 <br>
21
22 <br>
23 <% if @isarticle>=1 then %>
24
25 <table class="table table-bordered">
26 <tr>
27 <th>タイトル</th>
28 <th>公開状況</th>
29 <th>更新日時</th>
30 <th>いいね数</th>
31
32 </tr>
33 <% @s.each do |a| %>
34 <tr>
35 <% if a.title.size>20 %>
36 <td><%= a.title[0,20]+"..." %></td>
37 <% else %>
38 <td><%= a.title %></td>
39 <% end %>
40 <td>
41 <% if a.open==1 then%>
42 公開
43 <% else %>
44 非公開
45 <%end%>
46 </td>
47 <td><%= a.date %></td>
48 <td><%= a.good %></td>
49 <form method="post" action="detail">
50 <td><button type="submit" class="btn btn-primary">詳細</button></td>
51 <input type="hidden" name="id" value="<%= a.id %>">
52 </form>
53
54 <form method="post" action="edit">
55 <td><button type="submit" class="btn btn-success">編集</button></td>
56 <input type="hidden" name="id" value="<%= a.id %>">
57 </form>
58
59 <form method="post" action="del">
60 <td><button type="submit" class="btn btn-danger" onclick="return deleteCheck();">
61 削除</button></td>
62 <input type="hidden" name="id" value="<%= a.id %>">
63 <input type="hidden" name="_method" value="delete">
64 </form>
65 </tr>
66 <% end %>
67 </table>
68 <% else %>
69 <div class="row mx-2">
70 you have no article.
71 </div>
72 <% end %>
73
74 <br>
75 <a class="btn btn-link" data-toggle="collapse" href="/logout" role="button"
76 aria-expanded="false" aria-controls="collapseExample">
77 Logout
78 </a>

```

まず、「/myarticle」のページで、自分の記事一覧が表示される処理を実装する方法について説明する。リスト13では、4行目から20行目である。「/myarticle」でのサーバの処理は、cookie から取得したユーザー名を用いて、contents テーブルからユーザーの記事を取得することで行っている。myarticle.erb では、main.rb

で取得した記事の一覧を表形式で画面に表示する処理を行っている。リスト 14 で表の出力を行う記述は 23 行目から 72 行目である。23 行目では、記事が存在することを示すフラグが立っていることを確認してから表の出力を行っている。フラグが立っていない場合 70 行目の「you have no article.」が表示される。記事がある場合は、ループ処理を用いて記事を表示する。タイトル文字列が長い場合に、タイトルを 20 文字で区切り、「...」を表示する処理は 35 行目から 39 行目で行っている。

次に Myarticle ページから閲覧したい記事の詳細をクリックすると、記事の内容が見れる処理について説明する。「詳細」フォームはリスト 14 の 49 行目から 52 行目である。詳細フォームの提出時にユーザーは「詳細」ボタンをクリックするだけであるが、51 行目に示すように内部で記事の id の情報を提出している。これによってユーザーがどの記事の詳細をクリックしたのかを判別している。サーバ側では、詳細フォームが提出されると、リスト 13 の 25 行目から 39 行目の処理が行われる。具体的には、28 行目でフォームから受け取った id の記事を Contents テーブルから取得して、オブジェクト@a に代入している。そして「/detail」ページの HTML でオブジェクト@a の内容を表示することで、記事の詳細を表示している。

5.8 記事の編集

本節では記事の編集について、次に示す 2 つの内容を述べる。

1. 記事の編集を行う方法.
2. 実装部分の説明.

5.8.1 記事の編集を行う方法

記事の編集を行う方法について説明する。ユーザー「test」の「はじめての C 言語」という記事を編集してみる。まず、MyArticle ページ「/myarticle」にアクセスする。MyArticle ページは図 54 のページであった。記事の編集は、「詳細」ボタンの右側にある「編集」ボタンをクリックすることで行える。「編集」ボタンは文字色が白、背景色は「New Article」ボタンと同じ緑である。いま編集したい記事は「はじめての C 言語」だから、「はじめての C 言語」のレコードの「編集」ボタンをクリックする。



図 54: MyArticle ページの再掲 (ユーザー「test」)

図 55 および図 56 に「編集」ボタンをクリックした結果を示す。記事の新規作成ページと同様に、入力フォームを広くとっているため、2 枚の画像に分けて掲載している。また、記事の新規作成フォームとレイアウトはほとんど同じであるが、新規作成フォームは「/newarticle」、記事の編集フォームは「/edit」である。図 55 および図 56 から、記事の編集をクリックすると、入力フォームにすでに投稿した記事の内容が入力された状態でフォームが開くことがわかる。ただし、公開/非公開のチェックボックスは、もとの記事の状態を保持しない。つまり、チェックボックスにチェックのない非公開の状態になるため、公開したい場合は、再度チェックボックスにチェックを入れる必要がある。

Chama code

記事を更新します!

☐ 公開

タイトル

はじめてのC言語

記事の内容

C言語のHello Worldのコードです。

ソースコード

```
#include<stdio.h>

int main(void){
    printf("hello world\n");
    return 0;
}
```

図 55: 記事の編集ページの例 1

実行結果

hello world

Submit

図 56: 記事の編集ページの例 2

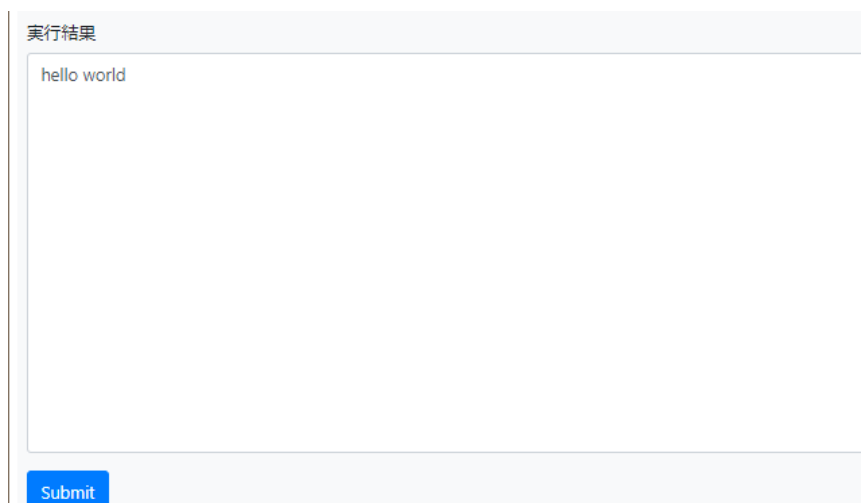
記事の編集画面が開けたから, 記事の内容を更新して提出する. 例として, 「はじめての C 言語」の記事を図

57 および図 58 に示すように「はじめての JavaScript」に更新した。図 57 および図 58 の状態で「Submit」をクリックすると記事の更新が行われる。図 59 に「Submit」をクリックした結果を示す。記事の更新に成功すると「/successarticle」にリダイレクトされる。図 59 のページは「記事の更新に成功しました」という内容を表示している。



The screenshot shows a web form titled "Chama code" with a dark header. Below the header, the text "記事を更新します!" (Update article!) is displayed. There are two input fields: "公開" (Public) with a checkbox and "タイトル" (Title) with a text box containing "はじめてのJavaScript". Below these is a large text area for "記事の内容" (Article content) containing "JavaScriptのHello Worldのコードです。". At the bottom, there is a "ソースコード" (Source code) section with a text box containing the JavaScript code `alert("hello world");`.

図 57: 記事の編集例 1



The screenshot shows the same web form as Figure 57, but with the "実行結果" (Execution result) section at the top displaying "hello world". The "Submit" button is visible at the bottom left of the form.

図 58: 記事の編集例 2

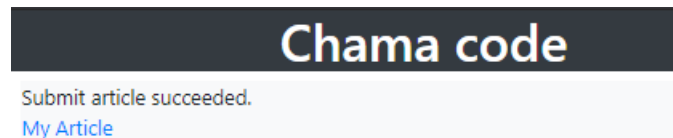


図 59: 記事の更新成功

MyArticle ページで記事の編集が行われたことを確認する。図 60 に記事を更新して MyArticle ページにアクセスした結果を示す。図 60 から「はじめての C 言語」という記事がなくなり、「はじめての JavaScript」という記事ができたことがわかる。さらに、更新日時が図 54 では「2020/12/18 18:14:48」になっていたが、記事を更新したことで、「2020/12/20 08:58:13」に更新されていることがわかる。「いいね」機能についてはまだ説明していないが、記事を更新しても、その記事のいいね数は保持される。



図 60: 記事の更新結果 (MyArticle ページ)

記事の内容が更新されていることを確認する。記事の内容が更新されていることは、更新した「はじめての JavaScript」という記事の詳細を確認することで行える。図 61 に「はじめての JavaScript」という記事の詳細を表示した結果を示す。図 61 から記事の内容も更新されていることが読み取れる。これらによって、記事の編集が行えることが確認できた。



図 61: 「はじめての JavaScript」の詳細

5.8.2 実装部分の説明

記事の編集の実装部分について説明する. リスト 15 およびリスト 16 に記事の編集の実装部分のコードを示す. リスト 15 は main.rb から, 記事の編集を行う部分のコードを抜き出したものである. リスト 15 のコードには, 「/edit」および「/authedit」にアクセスがきたときの処理が記述されている. リスト 16 は記事の編集ページ「/edit」の HTML である.

リスト 15: 自分の記事の編集の実装 (myarticle.erb)

```

1  # ログイン必須
2  # 記事の編集画面を表示
3  # 未ログイン時 : badrequestにリダイレクト
4  post '/edit' do
5    if (session[:login_flag]==true)
6      @a = Content.find(params[:id])
7      erb :edit
8    else
9      erb :badrequest
10   end
11 end
12
13 # ログイン必須
14 # 記事の更新処理
15 # 成功 : successarticleにリダイレクト
16 # 失敗 : !
17 # 未ログイン時 : badrequestにリダイレクト
18 post '/authedit' do
19   if (session[:login_flag]==true)
20     checkflg=true
21     title = CGI.escapeHTML(params[:title])
22     description = CGI.escapeHTML(params[:description])
23     code = CGI.escapeHTML(params[:code])
24     result = CGI.escapeHTML(params[:result])

```

```

25     if !checkstr(title,title_max) then
26         checkflg=false
27     elsif !checkstr(description,article_max) then
28         checkflg=false
29     elsif !checkstr(code,article_max) then
30         checkflg=false
31     elsif !checkstr(result,article_max) then
32         checkflg=false
33     end
34
35     if checkflg==false then
36         redirect '/failure3Resister'
37     else
38         a = Content.find(params[:id])
39         authtime = Time.now.strftime("%Y/%m/%d %T")
40         a.date = authtime
41         a.title = title
42         a.description = description
43         a.code = code
44         a.result = result
45         if params[:publicbutton] then
46             a.open=1
47         else
48             a.open=0
49         end
50         a.save
51         redirect '/successarticle'
52     end
53     else
54         erb :badrequest
55     end
56 end

```

リスト 16: 自分の記事の編集の実装 (edit.erb)

```

1  <h3>記事を更新します!</h3>
2
3  <div class="ml-2" style="width: 1200px;">
4  <form action="/authedit" method="post">
5  <div class="form-check">
6      <input class="form-check-input" name="publicbutton[]" value="checked"
7      type="checkbox" id="defaultCheck1">
8      <label class="form-check-label" for="defaultCheck1">
9          公開
10     </label>
11 </div>
12 <div class="form-group">
13     <label for="exampleFormControlInput1">タイトル</label>
14     <input type="text" class="form-control" name="title" maxlength="60"
15     value="<%= @a.title %>">
16 </div>
17 <br>
18 <div class="form-group">
19     <label for="exampleFormControlTextarea1">記事の内容</label>
20     <textarea class="form-control" name="description" id="exampleFormControlTextarea1"
21     rows="15" maxlength="1000"><%= @a.description %></textarea>
22 </div>
23
24 <div class="form-group">
25     <label for="exampleFormControlTextarea1">ソースコード</label>
26     <textarea class="form-control" name="code" id="exampleFormControlTextarea1"
27     rows="15" maxlength="1000"><%= @a.code %></textarea>
28 </div>
29 <div class="form-group">
30     <label for="exampleFormControlTextarea1">実行結果</label>
31     <textarea class="form-control" name="result" id="exampleFormControlTextarea1"
32     rows="15" maxlength="1000"><%= @a.result %></textarea>
33 </div>
34 <button type="submit" class="btn btn-primary">Submit</button>

```

```

35 <input type="hidden" name="id" value="<%= @a.id %>">
36 </form>
37 </div>

```

「編集」ボタンをクリックすると、もとの記事の内容を保持したフォームが開く仕組みについて説明する。まず、MyArticle ページの「編集」ボタンをクリックすると、「詳細」ボタンのときと同様にクリックした記事に対応する記事 id が送信される。「詳細」ボタンは図 14 の 54 行目から 57 行目である。フォームは「/edit」に送信される。サーバの「/edit」の処理はリスト 15 の 4 行目から 11 行目である。処理内容はセッション管理を行ったのちに、6 行目で、受け取った id から contents テーブルの該当するレコードを取得している。

リスト 16 に示した edit.erb は、フォームの内容自体は新規作成ページの HTML(newarticle.erb) と同じである。新規作成ページとの違いは、タイトルでは value タグ、それ以外の投稿フォームでは「<%= 文字列 %>」という記述を行うことで、フォームの初期値として、すでに作成した記事の内容を埋め込んでいることである。

更新後の記事が提出されたときの処理について説明する。投稿した記事には id の情報があるから、該当する id のレコードを更新することで、すでに作成した記事の編集を行うことができる。レコードの更新は、編集フォームを提出した「/authedit」で行っている。リスト 15 では 18 行目から 56 行目である。「/authedit」では 19 行目から 36 行目でセッション管理および入力文字列の確認とサニタイジングの処理を行った後に、レコードを更新する処理を行っている。レコードの更新処理を行っているのは 38 行目から 51 行目である。レコードの更新が完了した場合、51 行目に示すように「/successarticle」にリダイレクトされる。

5.9 記事の削除

本節では記事の削除について、次に示す 2 つの内容を述べる。

1. 記事の削除を行う方法.
2. 実装部分の説明.

5.9.1 記事の削除

記事の削除を行う方法を説明する。ユーザー「test」の「Python の Hello World」という記事を削除してみる。記事の削除は MyArticle ページから行うことができるから、まず MyArticle ページにアクセスする。MyArticle ページは図 54 に示したページである。記事の削除は「削除」というボタンをクリックすることで行える。いま「Python の Hello World」という記事を削除したいから、対応するレコードの「削除」ボタンをクリックする。図 62 および図 63 に「削除」ボタンをクリックした結果を示す。図 62 はユーザー 1 の端末での実行結果、図 63 はユーザー 2 での端末での実行結果である。どちらの場合も、「削除」ボタンをクリックすると「Really delete this article?」と表示される。図 62 はホスト端末であるため「127.0.0.1:9998 内容です」と表示され、図 63 は「192.168.0.102:9998 の内容です」と表示されている。記事の削除は、周りに詳細、編集というボタンがあるためワンクリックで削除できる仕様にすると、間違っって押してしまうことがあると考えた。このため、「本当に削除します?」というダイアログを挟むことで、間違っって記事を削除するというヒューマンエラーを防いでいる。削除確認ダイアログで「OK」をクリックすると図 64 に示すように「Python の Hello World」という記事が削除される。「キャンセル」をクリックした場合は図 65 に示すように「Canceled」というダイアログが表示される。図 65 で「OK」をクリックすると MyArticle ページに戻る。

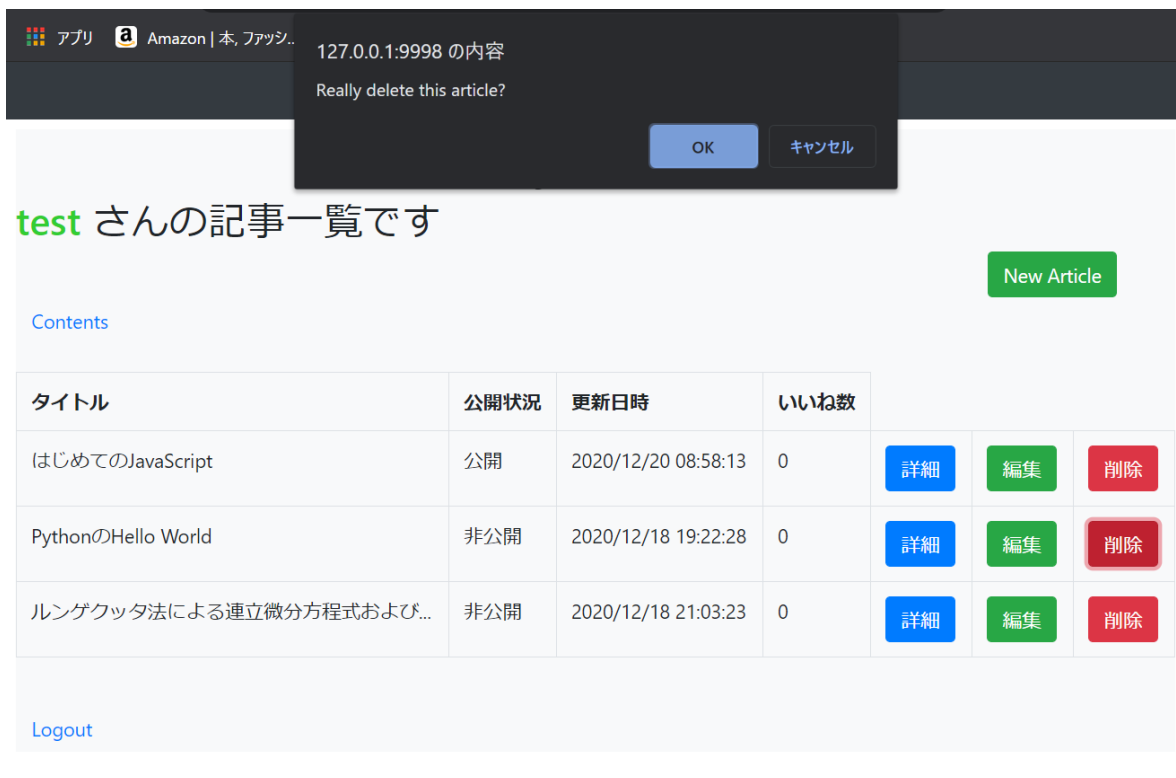


図 62: 削除確認ダイアログ (ユーザー 1)



図 63: 削除確認ダイアログ (ユーザー 1)



図 64: 「OK」をクリックした場合



図 65: 「キャンセル」をクリックした場合

5.9.2 実装部分の説明

記事の削除の実装部分について説明する. リスト 17 およびリスト 18 に記事の削除の実装部分のコードを示す. リスト 17 は「削除」ボタンをクリックして, さらに確認ダイアログで「OK」をクリックしたときの処理を行う部分を `main.rb` から抜粋したものである. リスト 18 は確認ダイアログの JavaScript のコードである.

リスト 17: 記事の削除の実装 (`main.rb`)

```

1 # ログイン 必須
2 # 記事の削除処理をしてmyarticleにリダイレクト

```

```

3 # 未ログイン時 : badrequestにリダイレクト
4 delete '/del' do
5   if (session[:login_flag]==true)
6     s=Content.find(params[:id])
7     s.destroy
8     redirect '/myarticle'
9   else
10    erb :badrequest
11  end
12 end

```

リスト 18: /public/deleteChecke.js

```

1 function deleteCheck(){
2   if(window.confirm('Really delete this article?')){
3     return true;
4   }else{
5     window.alert('Canceled. ');
6     return false;
7   }
8 }

```

まず、削除確認ダイアログが表示される仕組みについて説明する。「削除」ボタンは「詳細」ボタン、「編集」ボタンと同様に、削除する記事の id をサーバに送信する仕組みになっている。このことは、リスト 14 の 59 行目から 64 行目で確認できる。「詳細」、「編集」ボタンとの違いはリスト 14 の 60 行目の「onClick="return deleteCheck();"」という記述である。ただしこの記述を行う他に、リスト 14 の 1 行目で「deleteCheck.js」を読み込んでいることに注意が必要である。リスト 18 に示した deleteCheck.js には、deleteCheck という関数が定義されている。deleteCheck 関数は window.confirm という関数でダイアログを表示し、「OK」の場合は HTML に true、「キャンセル」の場合はアラートを表示して HTML に false を返す処理を行っている。HTML 側では、「削除」ボタンが押されたときに、deleteCheck 関数を呼び出して、その戻り値が true であればフォームを「/del」に送信する処理を行っている。

削除フォームを受け取ったサーバの処理について説明する。リスト 17 では 6 行目で id に該当する記事を contents テーブルから取得している。そして 7 行目で取得した記事を削除している。これによって削除をクリックした記事が contents テーブルから削除される。

5.10 記事の検索

本節では記事の検索について、次に示す 3 つの内容を述べる。

1. 記事の検索方法
2. 非公開の記事が表示されないことの確認
3. 実装部分の説明

5.10.1 記事の検索方法

記事の検索方法について説明する。記事の検索はコンテンツページ「/contentspage」から行うことができる。図 66 にコンテンツページを示す。記事の検索は「find article...」というフォームに検索したい記事名を入力することで行える。「検索」ボタンはないのか疑問に思うかもしれないが、このフォームは検索したい記事名を入力すると、フォームの下側に候補が表示される仕組みになっている。また、記事の検索は記事のタイトルのみで行っている。記事の内容で検索をかけられないのは不便であるが、開発時間の都合上このような仕様になっている。

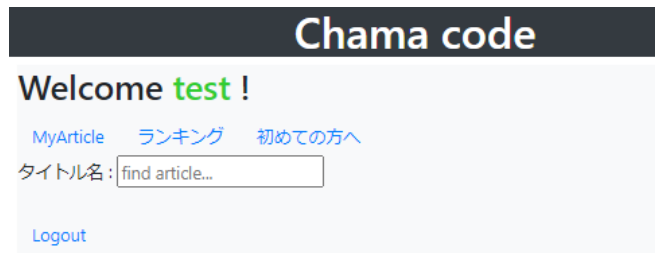


図 66: コンテンツページ

準備として, ユーザー「pfy」の「Say hello to Java」という記事を非公開から公開に変更した. いまユーザー「test」でログインしているものとして, 自分の記事を検索してみる. 図 64 からユーザー「test」には「はじめての JavaScript」という記事があることが分かるから, これを検索してみる. 図 67 は検索フォームに「は」を入力したときの結果である. 図 67 から「はじめての JavaScript」の記事が表示されていることが読みとれる.



図 67: 検索フォームの使用例 1

検索フォームに「J」と入力した場合も「はじめての JavaScript」という記事が表示されることを確認する. 図 68 に検索フォームに「J」と入力した結果を示す. 図 68 から「はじめての JavaScript」と「Say hello to Java」の 2 つの記事が表示されていることがわかる. どちらの記事のタイトルにも「J」が含まれているから, この結果は正しいことがわかる.

The screenshot shows the Chama code application interface. At the top, there's a dark header with 'Chama code' in white. Below it, a light blue box contains the text 'Welcome test !' and navigation links: 'MyArticle', 'ランキング', and '初めての方へ'. A search input field labeled 'タイトル名:' contains the character 'J'. Below the input, it says '2件見つかりました.' (2 items found). A table displays the search results:

タイトル	投稿者	更新日時	いいね数	
はじめてのJavaScript	test	2020/12/20 08:58:13	0	詳細
Say hello to Java	pfy	2020/12/20 12:35:10	0	詳細

At the bottom left of the light blue box, there is a 'Logout' link.

図 68: 検索フォームの使用例 2

検索結果が 0 件の場合の表示も確認する。タイトルに「C++」を含む記事は作成していないから、検索フォームで「C++」と入力してみる。図 69 は「C+」まで入力した結果である。図 69 から検索結果が 0 件の場合は、検索結果に「見つかりませんでした。」と表示されることがわかる。

The screenshot shows the Chama code application interface. At the top, there's a dark header with 'Chama code' in white. Below it, a light blue box contains the text 'Welcome test !' and navigation links: 'MyArticle', 'ランキング', and '初めての方へ'. A search input field labeled 'タイトル名:' contains the text 'C+'. Below the input, it says '見つかりませんでした.' (Not found). At the bottom left of the light blue box, there is a 'Logout' link.

図 69: 検索フォームの使用例 3

5.10.2 非公開の記事が表示されないことの確認

非公開の記事が検索結果に表示されないことを確認する。準備として、ユーザー「pfy」の「Ruby にまつわるエトセトラ」の記事を非公開に更新した。また、ユーザー「test」で「R 言語 入門」という非公開の記事を新規作成した。内容は適当なものとする。仕様の確認はユーザー「test」で「エトセトラ」、「R」の 2 単語を検索することで行う。非公開の記事が検索結果に表示されない仕様が実装されていれば、「エトセトラ」の場合は検索結果は 0 件、「R」の場合は「はじめての JavaScript」の記事のみが表示されるはずである。

まず「エトセトラ」を入力した場合について確認する。図 70 にユーザー「test」で「エトセトラ」を入力したときの結果を示す。図 70 では「エ」を入力しただけで、検索結果が 0 件になった。次に「R」を入力した場合について確認する。図 71 にユーザー「test」で「R」を入力したときの結果を示す。図 71 から「はじめての JavaScript」の記事のみが表示され、「R 言語 入門」の記事が表示されていないことがわかる。これらから、非公開の状態の記事は検索結果に表示されないことが確認できた。ちなみに、大文字小文字を区別しないのは、SQLite3 の LIKE 句の仕様である。

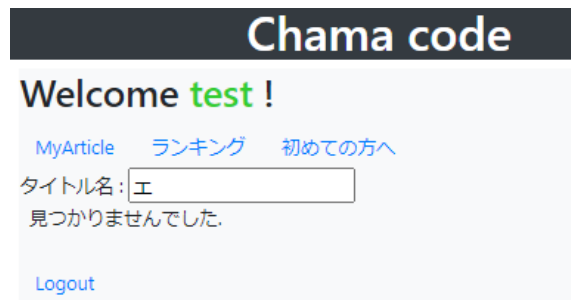


図 70: 「エトセトラ」を入力した場合



図 71: 「R」を入力した場合

5.10.3 実装部分の説明

記事の検索機能の実装部分について説明する。リスト 19, リスト 20, およびリスト 21 に記事の検索機能の実装部分のコードを示す。リスト 19 は「/search/:val」にアクセスがきたときの処理が記述されている。リスト 20 はコンテンツページの HTML が記述されている contents.erb である。リスト 21 はユーザーが検索欄に入力を行ったときに、呼び出される JavaScript のコードである。

リスト 19: 記事の検索の実装 (main.rb)

```

1 # ログイン 必須
2 # contents ページにおける記事の検索処理
3 # 未ログイン時 : badrequest にリダイレクト
4 get '/search/:val' do
5   if (session[:login_flag] == true)
6     key = params[:val]
7     s = Content.where("open==1 AND title LIKE ?", "%#{key}%")
8     puts s
9     l = s.length
10    r = []
11
12    r.push(kensu: "#{l}")
13    if l != 0 && l <= 100
14      s.each do |a|
15        d = {
16          id: "#{a.id}",

```

```

17         title: "#{a.title}",
18         username: "#{a.username}",
19         date: "#{a.date}",
20         good: "#{a.good}",
21     }
22     r.push(d)
23 end
24 else
25     d={
26         id: "none",
27         title:"none",
28         username:"none",
29         date:"none",
30         good:"none",
31     }
32     r.push(d)
33 end
34
35 r.to_json
36 else
37     erb :badrequest
38 end
39 end

```

リスト 20: 記事の検索の実装 (contents.rb)

```

1 <script type="text/javascript" src="search.js"></script>
2 <h2>Welcome <font color="#32cd32"><%= @a %> </font> !</h2>
3
4 <a class="btn btn-link" data-toggle="collapse" href="/myarticle" role="button"
5   aria-expanded="false" aria-controls="collapseExample">
6   MyArticle
7 </a>
8
9 <a class="btn btn-link" data-toggle="collapse" href="/ranking" role="button"
10  aria-expanded="false" aria-controls="collapseExample">
11 ランキング
12 </a>
13
14 <a class="btn btn-link" data-toggle="collapse" href="/beginer" role="button"
15  aria-expanded="false" aria-controls="collapseExample">
16 初めの方へ
17 </a>
18 <br>
19
20 <form>
21   タイトル名 : <input type="text" id="searchform" onKeyUp="doSearch();"
22   placeholder="find article..." maxlength=40>
23 </form>
24 <div id="kekka"></div>
25
26 <br>
27 <a class="btn btn-link" data-toggle="collapse" href="/logout" role="button"
28  aria-expanded="false" aria-controls="collapseExample">
29 Logout
30 </a>

```

リスト 21: 記事の検索の実装 (search.js)

```

1 var xhr = new XMLHttpRequest();
2
3 function doSearch(){
4     t = document.getElementById("searchform").value;
5     if(t.length!=0){
6         xhr.onreadystatechange = checkStatus;
7         xhr.open('GET','http://'+window.location.hostname+":9998/search/"+t,true);
8         xhr.responseType = 'json';
9         xhr.send(null);

```

```

10     }
11 }
12
13 function checkStatus(){
14     s="";
15     if((xhr.readyState==4)&&(xhr.status==200)){
16         a=xhr.response;
17         l=a[0].kensu;
18         s="<div class=\"row mx-2\">"
19         if(l==0){
20             s=s+"見つかりませんでした.</div>";
21         }else{
22             s=s+l+"件見つかりました.</div><br>";
23             if(l<=100){
24                 s=s+"<table class=\"table table-bordered\">";
25                 s=s+"<tr>";
26                 s=s+"<th>タイトル</th>";
27                 s=s+"<th>投稿者</th>";
28                 s=s+"<th>更新日時</th>";
29                 s=s+"<th>いいね数</th>";
30                 s=s+"<tr>";
31                 for(i=1;i<=l;i++){
32                     s=s+"<tr>";
33                     if(a[i].title.length>20){
34                         s=s+"<td>"+a[i].title.substr(0,20)+"...</td>";
35                     }else{
36                         s=s+"<td>"+a[i].title+"</td>";
37                     }
38                     s=s+"<td>"+a[i].username+"</td>";
39                     s=s+"<td>"+a[i].date+"</td>";
40                     s=s+"<td>"+a[i].good+"</td>";
41                     s=s+"<td>";
42                     s=s+"<form method=\"post\" action=\"detail\">";
43                     s=s+"<button type=\"submit\" class=\"btn btn-primary\">詳細</button>";
44                     s=s+"<input type=\"hidden\" name=\"id\" value=\""+a[i].id+"\">";
45                     s=s+"</form>";
46                     s=s+"</td>";
47                     s=s+"</tr>";
48                 }
49                 s=s+"</table>";
50             }
51         }
52         document.getElementById("kekka").innerHTML=s;
53     }
54 }

```

検索機能がどのように実装されているか説明する。検索欄はリスト 20 の 20 行目から 23 行目である。21 行目の「onKeyUp="doSearch()"」という記述は、入力欄でキーが押して、離されたときに JavaScript の doSearch という関数が呼び出されることを示している。これによって入力欄に単語を入力すると、自動で検索が行われる。doSearch 関数はリスト 21 の 3 行目から 11 行目である。doSearch 関数では XMLHttpRequest オブジェクトを用いてサーバに検索欄で入力した文字をわたして、検索結果の JSON ファイルを要求する処理を行っている。XMLHttpRequest オブジェクトは、web ブラウザとサーバの間で非同期に HTTP 通信を行うためのオブジェクトである。JSON は 7 行目で「サーバの IP アドレス:9998/search/hoge」にアクセスを送ることで要求している。「hoge」の部分は検索欄に入力した文字列が入る。

サーバから JSON を返ってきた場合、checkStatus 関数が呼び出される。checkStatus 関数はリスト 21 の 13 行目から 54 行目である。checkStatus 関数はこの JSON を解読して画面に表示する処理を行っている。checkStatus 関数の 17 行目から 49 行目で変数 s に画面に表示する内容を書き込み、52 行目で id が kekka の div タグに文字列 s を埋め込むことで画面に結果表示している。id が kekka の div タグはリスト 20 の 24 行目にある。

「サーバの IP アドレス:9998/search/hoge」にアクセスしたときのサーバの処理について説明する。リスト 19 の 43 行目に示すようにアクセスする文字列に不定な部分がある場合「:val」という記述をすることで、

不定な文字列を変数 `val` に代入することができる。これによってサーバが、ユーザーが入力した文字列がわかる仕組みになっている。7 行目では、検索ワードで `contents` テーブルに検索をして、結果を取得している。データベースから検索結果を取得できたから、これを JSON 形式で返す処理を行う。この処理を行っているのはリスト 19 の 12 行目から 35 行目である。

5.11 「いいね」機能

本節では「いいね」機能について、次に示す 3 つの内容を述べる。

1. 「いいね」機能の説明
2. 自分の記事に「いいね」を押せないことの確認
3. 実装部分の説明

5.11.1 「いいね」機能の説明

他のユーザーの記事を検索できることは前節で説明した。他のユーザーの記事を閲覧した場合、「いいね」を押せる仕様を実装した。本項では他のユーザーの記事に「いいね」を押す方法について説明する。ユーザー「test」がユーザー「pfy」の記事「Say hello to Java」にいいねを押してみる。まず、ユーザー「test」でログインを行って、コンテンツページを開く。次に、図 72 のように検索欄に「Java」と入力する。これでユーザー「pfy」の記事「Say hello to Java」が検索できた。「いいね」ボタンは記事の詳細ページにあるため、「詳細」ボタンをクリックして記事の詳細を開く。図 73 が記事「Say hello to Java」を開いた状態である。図 73 において、「good」ボタンを押すと「いいね」数が増え、コンテンツページにリダイレクトされる。ボタンをページの最下部に設置しているのは、記事を読んだ上で「いいね」を押してほしいからである。



Chama code

Welcome test !

[MyArticle](#) [ランキング](#) [初めての方へ](#)

タイトル名:

2件見つかりました.

タイトル	投稿者	更新日時	いいね数	
はじめてのJavaScript	test	2020/12/20 08:58:13	0	詳細
Say hello to Java	pfy	2020/12/20 12:35:10	0	詳細

[Logout](#)

図 72: 「Java」を入力した状態

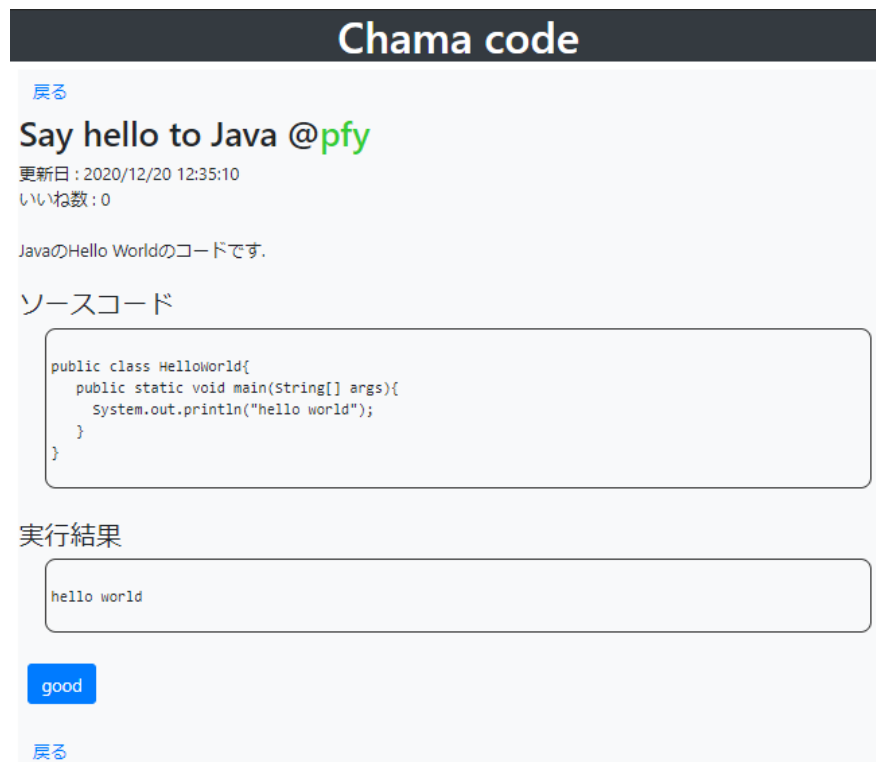


図 73: 記事「Say hello to Java」

「いいね」ボタンを押したことが反映されていることを確認する。反映されていることを確認するためには、コンテンツページの検索欄で「Java」と検索すればよい。図 74 にコンテンツページの検索欄で「Java」と検索した結果を示す。図 72 のときは、記事「Say hello to Java」のいいね数が「0」であったが、図 74 ではいいね数が「1」になっていることが読み取れる。



図 74: いいね数の変化の反映 (コンテンツページ)

詳細ページでもいいね数の変化が反映されていることを確認する。図 75 に記事「Say hello to Java」の詳細を表示したときの結果を示す。図 73 ではいいね数が「0」であったが、図 75 ではいいね数が「1」になっていることが読み取れる。

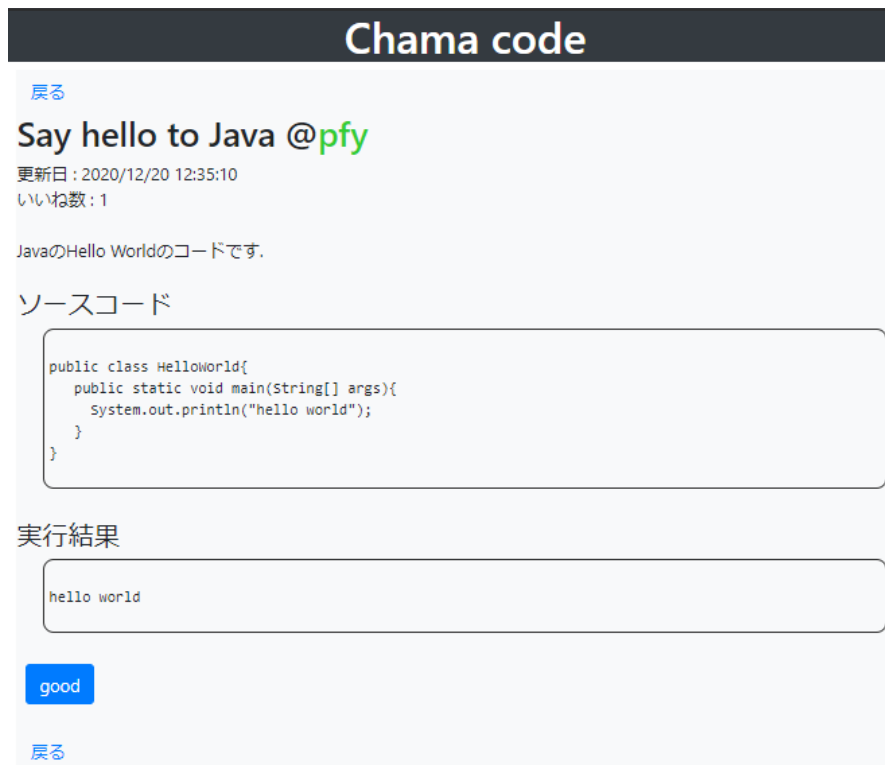


図 75: いいね数の変化の反映 (詳細ページ)

5.11.2 自分の記事に「いいね」を押せないことの確認

「いいね」機能には、他のユーザーの投稿に「いいね」を押せるが、自分の投稿には「いいね」を押せない仕様がある。この機能が実装されていることを確認する。例としてユーザー「test」の記事「はじめてのJavaScript」の詳細を、ユーザー「test」で開いてみる。図 76 にコンテンツページで検索した「はじめてのJavaScript」を投稿者本人が開いた場合の画面を示す。また比較のためにユーザー「pfy」が「はじめてのJavaScript」を開いた場合の画面を図 77 に示す。図 76 では「good」ボタンが表示されていないが、図 77 では「good」ボタンが表示されていることが読み取れる。このことから自分の記事にいいねが押せない仕様の実装できていることが確認できた。

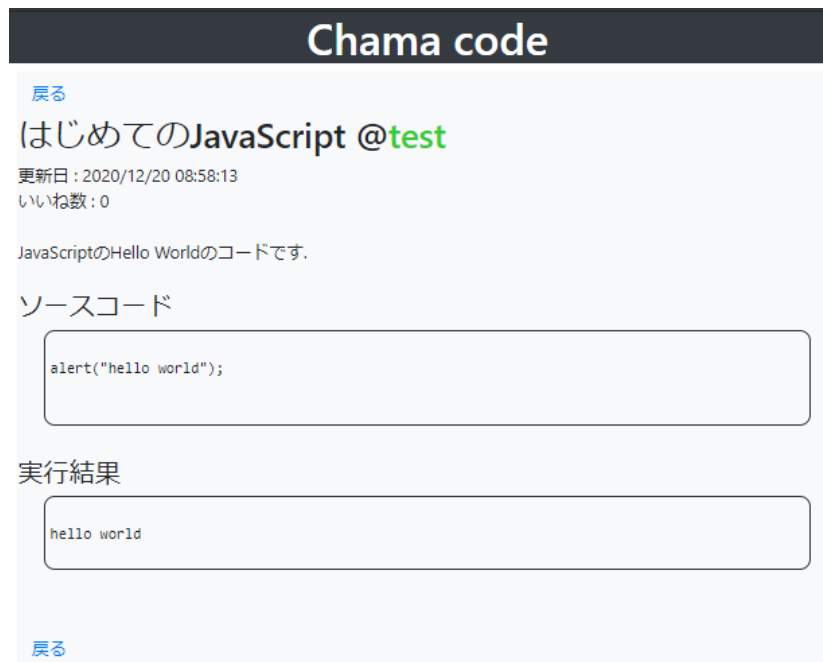


図 76: 自分の記事の詳細を開いた画面

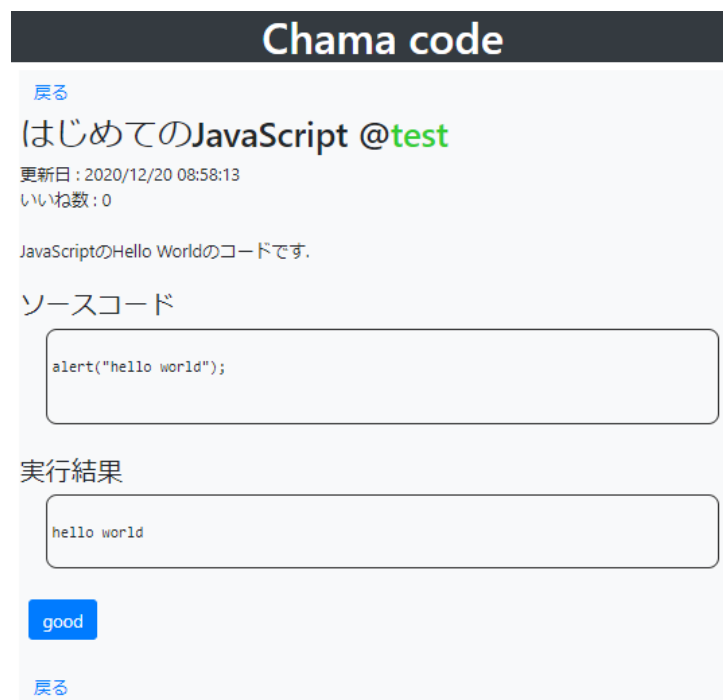


図 77: ユーザー「pfy」が記事の詳細を開いた画面

5.11.3 実装部分の説明

「いいね」機能の実装部分について説明する。リスト 22 に「いいね」ボタンをクリックしたときの処理を行う部分のコードを示す。「/detail」の「good」ボタンを押すと、記事の id を含むフォームが「/good」に提出される。リスト 22 の 6 行目から 8 行目では、contents テーブルから該当する記事を取得し、いいね数を保持するカラムである good をインクリメントしている。これによって、いいね数をカウントする処理を行っている。

リスト 22: 「いいね」機能の実装 (main.rb)

```
1 # ログイン必須
2 # いいねが押されたときの処理をして contentpage にリダイレクト
3 # 未ログイン時 : badrequest にリダイレクト
4 post '/good' do
5   if (session[:login_flag]==true)
6     a = Content.find(params[:id])
7     a.good +=1
8     a.save
9     redirect '/contentpage'
10  else
11    erb :badrequest
12  end
13 end
```

5.12 ランキングの表示

本節ではランキング機能について、次に示す 2 つの内容を述べる。

1. ランキング機能の説明
2. 実装部分の説明

5.12.1 ランキング機能の説明

いいね数が多い記事をランキングで表示する機能を実装した。本項では、この機能について説明する。準備として、ユーザー「test」の記事「はじめての JavaScript」のいいね数を「1」、ユーザー「pfy」の記事「Say hello to Java」のいいね数を「2」にする処理を行った。ランキングはコンテンツページの「ランキング」をクリックすることで見ることができる。図 78 はランキングページ「/ranking」にアクセスしたときの画面である。ランキングページはいいね数が 1 以上の記事を降順で表示する仕様になっている。

Chama code					
ランキング					
Contents					
順位	タイトル	投稿者	更新日時	いいね数	
1	Say hello to Java	pfy	2020/12/20 12:35:10	2	詳細
2	はじめてのJavaScript	test	2020/12/20 08:58:13	1	詳細
戻る					

図 78: ランキングの例 1

新たにユーザー「judy」を作成して、記事「Start SQL」という記事を投稿した。さらに、この記事に別のユーザーが5いいねをつけた。このときにランキングの1位に記事「Start SQL」が表示されることを確認する。図 79 にいいね数が多い記事を作成したときの、ランキングの変化を示す。図 79 から、いいね数が最も多い「Start SQL」がランキング1位になっていることがわかる。

Chama code					
ランキング					
Contents					
順位	タイトル	投稿者	更新日時	いいね数	
1	Start SQL	judy	2020/12/20 18:00:39	5	詳細
2	Say hello to Java	pfy	2020/12/20 12:35:10	2	詳細
3	はじめてのJavaScript	test	2020/12/20 08:58:13	1	詳細
戻る					

図 79: ランキングの例 2

5.12.2 実装部分の説明

ランキング機能の実装部分について説明する。リスト 23 に、ランキング機能の実装部分のコードを示す。リスト 23 は「/ranking」にアクセスがきたときのサーバの処理が記述されている。リスト 23 の 11 行目で、いいね数が 1 以上の公開記事を降順で contents テーブルから取得し、変数@s に代入している。この変数@s を ranking.erb で展開することで、ランキングが表示される。

リスト 23: ランキング機能の実装 (main.rb)

```

1 # ログイン 必須
2 # いいね数の多い記事を表示
3 # 未ログイン時 : badrequest にリダイレクト

```

```

4 get '/ranking' do
5   @isarticle=0
6   if (session[:login_flag]==true)
7     @a = session[:username]
8     # userの記事の件数を取得
9     @c = Content.select('*').count
10
11     if @c>=1 then # 記事があるとき読み込み
12       @s = Content.where("open==1 AND good>0").order('good desc')
13       @isarticle=1
14     end
15
16     erb :ranking
17   else
18     erb :badrequest
19   end
20 end

```

5.13 説明ページの表示

Web アプリケーションの使い方を紹介するページを作成した. このページはコンテンツページの「初めての方へ」というリンクからアクセスできる「/beginer」というリンクのページである. 図 80 および図 81 に説明ページを示す.

Chama code

Chama codeの使い方

[戻る](#)

初めまして,**test** さん!

このページでは,Chama codeの概要と使い方解説します!

目次

- [1. そもそもChama codeって何?](#)
- [2. 記事を投稿しよう!](#)
- [3. 記事を検索しよう!](#)
- [4. ランキングを見よう!](#)

そもそもChama codeって何?

Chama codeは4Iネットワークプログラミングの課題で作成したWebアプリです. 作成者は金澤雄大です.このアプリはユーザーのソースコードを共有することを目的に作られています. 公開したソースコードは他のユーザーが「いいね」をつけることができます. 「いいね」が多い投稿はランキングの上位に表示されます. 記事のランクインを目指しましょう!

記事を投稿しよう!

まずは,**test** さんの記事を投稿しましょう. 記事の投稿は次の手順で行うことができます.

- [MyArticle](#)の「New Article」から投稿フォームを開きます.
- 投稿内容(タイトル,説明,ソースコード,実行結果)を書きます.タイトルは60文字,説明,ソースコード,実行結果は100文字まで入力できます.ただし,HTMLやJavaScriptのタグは反映されません.また,記事は公開/非公開の 2つの状態があり,「公開」のボタンにチェックをいれると全てのユーザーが閲覧できるようになります. ボタンにチェックを入れない場合は投稿者本人のみがその記事を閲覧できます.

新しい記事を作成します!

☐ 公開

タイトル

title

記事の内容

3. 投稿フォームの一番下にある「Submit」をクリックして投稿しましょう.

図 80: 説明ページ 1

記事を投稿しよう!

まずは, [test](#) さんの記事を投稿しましょう. 記事の投稿は次の手順で行うことができます.

1. [MyArticle](#) の「New Article」から投稿フォームを開きます.
2. 投稿内容(タイトル, 説明, ソースコード, 実行結果)を書きます. タイトルは60文字, 説明, ソースコード, 実行結果は100文字まで入力できます. ただし, HTMLやJavaScriptのタグは反映されません. また, 記事は公開/非公開の2つの状態があり, 「公開」のボタンにチェックをいれると全てのユーザーが閲覧できるようになります. ボタンにチェックを入れない場合は投稿者本人のみがその記事を閲覧できます.

新しい記事を作成します!

☐ 公開

タイトル

title

記事の内容

3. 投稿フォームの一番下にある「Submit」をクリックして投稿しましょう.

4. 投稿した記事は [MyArticle](#) に表示されます. 投稿内容を表示したいときは, 「詳細」をクリックしましょう. 投稿内容を編集したいときは「編集」をクリックしましょう. 投稿内容を削除したいときは「削除」をクリックしましょう.

記事を検索しよう!

ユーザーが公開している記事は誰でも見ることができます. 公開されている記事を検索してみましょう. 検索は [Contents](#) のフォームから行えます. 読みたい記事にマッチしそうな単語を入力しましょう. 検索は入力した単語と記事のタイトルで行っています. 記事の内容で検索はできません. 他のユーザーの記事は, 記事の詳細の一番下に「いいね」ボタンがあります. 素晴らしい記事には「いいね」を押しましょう!

ランキングを見よう!

公開されている「いいね」が多い記事のランキングを見ることができます. ランキングは [こちら](#)

[戻る](#)

図 81: 説明ページ 2

アプリの説明ページの実装部分をリスト 24 に示す. リスト 24 のコードでは, 「/beginer」にアクセスがきたときに `beginer.erb` を表示する処理を行っている.

リスト 24: アプリの説明ページの実装 (main.rb)

```
1 # ログイン 必須
2 # 初めの方へのページを表示
3 # 未ログイン時 : badrequest にリダイレクト
4 get '/beginer' do
5   if (session[:login_flag]==true)
6     @a = session[:username]
7     erb :beginer
8   else
9     erb :badrequest
10  end
11 end
```

6 付録 ソースコード

本章では, 付録としてソースコードの一覧を示す.

6.1 Gemfile

リスト 25 に Gemfile のコードを示す.Gemfile は Ruby のライブラリ管理を行うためのファイルである. リスト 25 では,Ruby 用のフレームワークである Sinatra や ActiveRecord,SQLite のライブラリを Gemfile に記述している.

リスト 25: Gemfile

```
1 # frozen_string_literal: true
2
3 source "https://rubygems.org"
4
5 git_source(:github) {|repo_name| "https://github.com/#{repo_name}" }
6
7 gem 'sinatra'
8 gem 'activerecord'
9 gem 'sqlite3'
10 # gem "rails"
```

6.2 chamacode.sql3

リスト 26 に chamacode.sql3 のコードを示す.chamacode.sql3 はデータベース定義を行うクエリを記述したファイルである.

リスト 26: chamacode.sql3

```
1 create table users (
2   username char(40) primary key,
3   salt varchar(40),
4   hashed varchar(40),
5   algo char(5)
6 );
7
8 create table contents (
9   id char(40) primary key,
10  username char(40),
11  date char(19),
12  title char(60),
13  description char(1000),
14  code char(1000),
15  result char(1000),
16  open int,
17  good int
18 );
```

6.3 database.yml

リスト 27 に database.yml のコードを示す.database.yml は Ruby でデータベースを扱うためのアダプターとデータベース言語の設定を行うためのファイルである. なお, このファイルを記述するときはインデントに注意が必要である.

リスト 27: database.yml

```
1 development:
2   adapter: sqlite3
3   database: chamacode.db
```

6.4 main.rb

リスト 28 に main.rb のコードを示す.

リスト 28: main.rb

```
1 require 'sinatra'
2 require 'digest/md5'
3 require 'active_record'
4 require 'json'
5 require 'cgi/escape'
6
7 ActiveRecord::Base.configurations = YAML.load_file('database.yml')
8 ActiveRecord::Base.establish_connection :development
9
10 set :environment, :production
11 set :sessions,
12   expire_after: 7200,
13   secret: 'Endkq8ty2ll0hnt48sm1b'
14
15 class User < ActiveRecord::Base
16 end
17
18 class Content < ActiveRecord::Base
19 end
20
21 # 名前の最大長
22 username_max = 40
23 # パスワードの最大長
24 password_max = 40
25 # titleの最大長
26 title_max = 60
27 # 記事の最大長
28 article_max = 1000
29
30 # パスワードを確認する関数
31 # 入力とDBが一致 -> true
32 # 不一致 -> false
33 def checkpass(trial_username, trial_passwd)
34   # Search recorded info
35   begin
36     a = User.find(trial_username)
37     db_username = a.username
38     db_salt = a.salt
39     db_hashed = a.hashed
40     db_algo = a.algo
41   rescue => e
42     puts "User #{trial_username} is not found."
43     puts e.message
44     return false
45   end
46
47   # Generate a hashed value
48   if db_algo == "1"
49     trial_hashed = Digest::MD5.hexdigest(db_salt+trial_passwd)
50   else
51     puts "Unknown algorithm is used for user #{trial_username}."
52     return false
53   end
54
55   if db_hashed == trial_hashed
56     return true
57   else
58     return false
59   end
60 end
61
62 # 入力文字列が1文字以上, 最大長以下であることを確認する関数
63 # 1 <= inputstr < maxlen : true
64 # else : false
```



```

65 def checkstr(inputstr,maxlen)
66   if inputstr.size==0 then
67     return false
68   elsif inputstr.size>maxlen then
69     return false
70   else
71     return true
72   end
73 end
74
75 # loginにリダイレクト
76 get '/' do
77   redirect '/login'
78 end
79
80 # ログインフォームを表示
81 get '/login' do
82   erb :loginscr
83 end
84
85 # ログイン管理
86 # 成功 : contentspageにリダイレクト
87 # 失敗 : failureにリダイレクト
88 post '/auth' do
89   checkflg=true
90   username = CGI.escapeHTML(params[:uname])
91   pass = CGI.escapeHTML(params[:pass])
92   if !checkstr(username,username_max) then
93     checkflg=false
94   elsif !checkstr(pass,password_max) then
95     checkflg=false
96   end
97
98   if checkflg==false then
99     redirect '/failure3Resister'
100  else
101    if(checkpass(username,pass))
102      session[:login_flag] = true
103      session[:username] = username
104      redirect '/contentspage'
105    else
106      session[:login_flag] = false
107      redirect '/failure'
108    end
109  end
110 end
111
112 # ログイン失敗の表示
113 get '/failure' do
114   erb :failure
115 end
116
117 # アカウント作成フォームを表示
118 get '/createaccount' do
119   erb :makeAccount
120 end
121
122 # アカウント作成管理
123 # 成功 : successResister
124 # 失敗1(usernameが既に存在) : failure1Registerにリダイレクト
125 # 失敗2(pass1,pass2が不一致) : failure2Registerにリダイレクト
126 # 失敗3(入力エラー) : failure3Registerにリダイレクト
127 post '/newaccount' do
128   checkflg=true
129   username = CGI.escapeHTML(params[:uname])
130   pass1 = CGI.escapeHTML(params[:pass1])
131   pass2 = CGI.escapeHTML(params[:pass2])
132   if !checkstr(username,username_max) then
133     checkflg=false
134   elsif !checkstr(pass1,password_max) then

```

```

135     checkflg=false
136   elsif !checkstr(pass2,password_max) then
137     checkflg=false
138   end
139
140   if checkflg==false then
141     redirect '/failure3Resister'
142   else
143
144     begin
145       a = User.find(username)
146       redirect '/failure1Resister'
147     rescue => e
148       if pass1==pass2 then
149         r = Random.new
150         algorithm = "1"
151         salt = Digest::MD5.hexdigest(r.bytes(20))
152         hashed = Digest::MD5.hexdigest(salt+pass1)
153         s = User.new
154         s.id = username
155         s.salt = salt
156         s.hashed = hashed
157         s.algo = algorithm
158         s.save
159         redirect '/successResister'
160       else
161         redirect '/failure2Resister'
162       end
163     end
164   end
165 end
166
167 # アカウント作成成功の表示
168 get '/successResister' do
169   erb :successResister
170 end
171
172 # usernameが既に存在しているエラーを表示
173 get '/failure1Resister' do
174   erb :failure1Resister
175 end
176
177 # pass1,pass2が不一致のエラーを表示
178 get '/failure2Resister' do
179   erb :failure2Resister
180 end
181
182 # 入力エラーの表示
183 get '/failure3Resister' do
184   erb :failure3Resister
185 end
186
187 # パスワードの再発行フォームを表示
188 get '/forgetpass' do
189   erb :forgetpass
190 end
191
192 # パスワードを再発行する処理
193 # 成功 : successRenewpassにリダイレクト
194 # 失敗(usernameが存在しない) : unknownUserにリダイレクト
195 # 失敗(pass1,pass2が不一致) : failure2Resisterにリダイレクト
196 post '/newpass' do
197   checkflg=true
198   username = CGI.escapeHTML(params[:uname])
199   pass1 = CGI.escapeHTML(params[:pass1])
200   pass2 = CGI.escapeHTML(params[:pass2])
201   if !checkstr(username,username_max) then
202     checkflg=false
203   elsif !checkstr(pass1,password_max) then
204     checkflg=false

```

```

205     elsif !checkstr(pass2, password_max) then
206         checkflg=false
207     end
208
209     if checkflg==false then
210         redirect '/failure3Resister'
211     else
212         begin
213             a = User.find(username)
214             if pass1==pass2 then
215                 r = Random.new
216                 algorithm = "1"
217                 salt = Digest::MD5.hexdigest(r.bytes(20))
218                 hashed = Digest::MD5.hexdigest(salt+pass1)
219                 a.salt = salt
220                 a.hashed = hashed
221                 a.algo = algorithm
222                 a.save
223                 redirect '/successRenewpass'
224             else
225                 redirect '/failure2Resister'
226             end
227         rescue => e
228             redirect '/unknownUser'
229         end
230     end
231 end
232
233 # パスワードの再発行成功を表示
234 get '/successRenewpass' do
235     erb :successRenewpass
236 end
237
238 # usernameが見つからないことを表示
239 get '/unknownUser' do
240     erb :unknownUser
241 end
242
243 # ログイン必須
244 # コンテンツページを表示
245 # 未ログイン時 : badrequestにリダイレクト
246 get '/contentspage' do
247     if (session[:login_flag]==true)
248         @a = session[:username]
249         erb :contents
250     else
251         erb :badrequest
252     end
253 end
254
255 # ログイン必須
256 # いいね数の多い記事を表示
257 # 未ログイン時 : badrequestにリダイレクト
258 get '/ranking' do
259     @isarticle=0
260     if (session[:login_flag]==true)
261         @a = session[:username]
262         # userの記事の件数を取得
263         @c = Content.select('*').count
264
265         if @c>=1 then # 記事があるとき読み込み
266             @s = Content.where("open==1 AND good>0").order('good desc')
267             @isarticle=1
268         end
269
270         erb :ranking
271     else
272         erb :badrequest
273     end
274 end

```

```

275
276 # ログイン必須
277 # 初めの方へのページを表示
278 # 未ログイン時 : badrequestにリダイレクト
279 get '/beginer' do
280   if (session[:login_flag]==true)
281     @a = session[:username]
282     erb :beginer
283   else
284     erb :badrequest
285   end
286 end
287
288 # ログイン必須
289 # 記事の新規作成フォームを表示
290 # 未ログイン時 : badrequestにリダイレクト
291 get '/newarticle' do
292   if (session[:login_flag]==true)
293     erb :newarticle
294   else
295     erb :badrequest
296   end
297 end
298
299 # ログイン必須
300 # ログインしているユーザーの記事だけを表示
301 # 未ログイン時 : badrequestにリダイレクト
302 get '/myarticle' do
303   @isarticle=0
304   if (session[:login_flag]==true)
305     @a = session[:username]
306     # userの記事の件数取得
307     @c = Content.select('*').where('username == \''+@a+'\'').count
308
309     if @c>=1 then # 記事があるとき読み込み
310       @s = Content.select('*').where('username == \''+@a+'\'')
311       @isarticle=1
312     end
313
314     erb :myarticle
315   else
316     erb :badrequest
317   end
318 end
319
320 # ログイン必須
321 # 記事の新規作成フォームを表示
322 # 未ログイン時 : badrequestにリダイレクト
323 get '/newarticle' do
324   if (session[:login_flag]==true)
325     erb :newarticle
326   else
327     erb :badrequest
328   end
329 end
330
331 # ログイン必須
332 # 記事の新規作成処理
333 # 成功 : successarticleにリダイレクト
334 # 失敗 : !
335 # 未ログイン時 : badrequestにリダイレクト
336 post '/autharticle' do
337   if (session[:login_flag]==true)
338     # サニタイジング処理を行う場所
339     checkflg=true
340     title = CGI.escapeHTML(params[:title])
341     description = CGI.escapeHTML(params[:description])
342     code = CGI.escapeHTML(params[:code])
343     result = CGI.escapeHTML(params[:result])
344     if !checkstr(title,title_max) then

```

```

345     checkflg=false
346   elsif !checkstr(description,article_max) then
347     checkflg=false
348   elsif !checkstr(code,article_max) then
349     checkflg=false
350   elsif !checkstr(result,article_max) then
351     checkflg=false
352   end
353
354   if checkflg==false then
355     redirect '/failure3Resister'
356   else
357
358     authtime = Time.now.strftime("%Y/%m/%d %T")
359     @a = session[:username]
360     r = Random.new
361     s = Content.new
362     s.id = Digest::MD5.hexdigest(r.bytes(40))
363     s.username = @a
364     s.date = authtime
365     s.title = title
366     s.description = description
367     s.code = code
368     s.result = result
369     s.good=0
370
371     if params[:publicbutton] then
372       s.open=1
373     else
374       s.open=0
375     end
376     s.save
377     redirect '/successarticle'
378   end
379   else
380     erb :badrequest
381   end
382 end
383
384 # ログイン必須
385 # 記事の新規作成成功を表示
386 # 未ログイン時 : badrequestにリダイレクト
387 get '/successarticle' do
388   if (session[:login_flag]==true)
389     erb :successarticle
390   else
391     erb :badrequest
392   end
393 end
394
395 # ログイン必須
396 # 記事の削除処理をしてmyarticleにリダイレクト
397 # 未ログイン時 : badrequestにリダイレクト
398 delete '/del' do
399   if (session[:login_flag]==true)
400     s=Content.find(params[:id])
401     s.destroy
402     redirect '/myarticle'
403   else
404     erb :badrequest
405   end
406 end
407
408 # ログイン必須
409 # 記事の詳細を表示
410 # 未ログイン時 : badrequestにリダイレクト
411 post '/detail' do
412   if (session[:login_flag]==true)
413     @cookieu = session[:username]
414     @a = Content.find(params[:id])

```

```

415     if @cookieu.eql?(@a.username) then
416         @user_flg=true
417     else
418         @user_flg=false
419     end
420
421     erb :detail
422 else
423     erb :badrequest
424 end
425 end
426
427 # ログイン必須
428 # いいねが押されたときの処理をしてcontentspageにリダイレクト
429 # 未ログイン時 : badrequestにリダイレクト
430 post '/good' do
431     if (session[:login_flag]==true)
432         a = Content.find(params[:id])
433         a.good +=1
434         a.save
435         redirect '/contentspage'
436     else
437         erb :badrequest
438     end
439 end
440
441 # ログイン必須
442 # 記事の編集画面を表示
443 # 未ログイン時 : badrequestにリダイレクト
444 post '/edit' do
445     if (session[:login_flag]==true)
446         @a = Content.find(params[:id])
447         erb :edit
448     else
449         erb :badrequest
450     end
451 end
452
453 # ログイン必須
454 # 記事の更新処理
455 # 成功 : successarticleにリダイレクト
456 # 失敗 : !
457 # 未ログイン時 : badrequestにリダイレクト
458 post '/authedit' do
459     if (session[:login_flag]==true)
460         checkflg=true
461         puts params[:title]
462         puts params[:description]
463         puts params[:code]
464         puts params[:result]
465         title = CGI.escapeHTML(params[:title])
466         description = CGI.escapeHTML(params[:description])
467         code = CGI.escapeHTML(params[:code])
468         result = CGI.escapeHTML(params[:result])
469         if !checkstr(title,title_max) then
470             checkflg=false
471         elsif !checkstr(description,article_max) then
472             checkflg=false
473         elsif !checkstr(code,article_max) then
474             checkflg=false
475         elsif !checkstr(result,article_max) then
476             checkflg=false
477         end
478
479         if checkflg==false then
480             redirect '/failure3Resister'
481         else
482             a = Content.find(params[:id])
483             authtime = Time.now.strftime("%Y/%m/%d %T")
484             a.date = authtime

```

```

485     a.title = title
486     a.description = description
487     a.code = code
488     a.result = result
489     if params[:publicbutton] then
490       a.open=1
491     else
492       a.open=0
493     end
494     a.save
495     redirect '/successarticle'
496   end
497   else
498     erb :badrequest
499   end
500 end
501
502 # ログアウトの処理
503 get '/logout' do
504   session.clear
505   erb :logout
506 end
507
508 # ログイン必須
509 # contents ページにおける記事の検索処理
510 # 未ログイン時 : badrequest にリダイレクト
511 get '/search/:val' do
512   if (session[:login_flag]==true)
513     key=params[:val]
514     s=Content.where("open==1 AND title LIKE ?","%#{key}%")
515     puts s
516     l=s.length
517     r=[]
518
519     r.push(kensu: "#{l}")
520     if l != 0 && l <= 100
521       s.each do |a|
522         d = {
523           id: "#{a.id}",
524           title: "#{a.title}",
525           username: "#{a.username}",
526           date: "#{a.date}",
527           good: "#{a.good}",
528         }
529         r.push(d)
530       end
531     else
532       d={
533         id: "none",
534         title: "none",
535         username: "none",
536         date: "none",
537         good: "none",
538       }
539       r.push(d)
540     end
541
542     r.to_json
543   else
544     erb :badrequest
545   end
546 end

```

6.5 deleteCheck.js

リスト 29 に deleteCheck.js のコードを示す.

リスト 29: deleteCheck.js

```

1 function deleteCheck(){
2     if(window.confirm('Really delete this article?')){
3         return true;
4     }else{
5         window.alert('Canceled. ');
6         return false;
7     }
8 }

```

6.6 search.js

リスト 30 に search.js のコードを示す.

リスト 30: search.js

```

1 var xhr = new XMLHttpRequest();
2
3 function doSearch(){
4     t = document.getElementById("searchform").value;
5     if(t.length!=0){
6         xhr.onreadystatechange = checkStatus;
7         xhr.open('GET','http://'+window.location.hostname+":9998/search/"+t,true);
8         xhr.responseType = 'json';
9         xhr.send(null);
10    }
11 }
12
13 function checkStatus(){
14     s="";
15     if((xhr.readyState==4)&&(xhr.status==200)){
16         a=xhr.response;
17         l=a[0].kensu;
18         s="<div class=\"row mx-2\">"
19         if(l==0){
20             s=s+"見つかりませんでした.</div>";
21         }else{
22             s=s+l+"件見つかりました.</div><br>";
23             if(l<=100){
24                 s=s+"<table class=\"table table-bordered\">";
25                 s=s+"<tr>";
26                 s=s+"<th>タイトル</th>";
27                 s=s+"<th>投稿者</th>";
28                 s=s+"<th>更新日時</th>";
29                 s=s+"<th>いいね数</th>";
30                 s=s+"<tr>";
31                 for(i=1;i<=l;i++){
32                     s=s+"<tr>";
33                     if(a[i].title.length>20){
34                         s=s+"<td>"+a[i].title.substr(0,20)+"...</td>";
35                     }else{
36                         s=s+"<td>"+a[i].title+"</td>";
37                     }
38                     s=s+"<td>"+a[i].username+"</td>";
39                     s=s+"<td>"+a[i].date+"</td>";
40                     s=s+"<td>"+a[i].good+"</td>";
41                     s=s+"<td>";
42                     s=s+"<form method=\"post\" action=\"detail\">";
43                     s=s+"<button type=\"submit\" class=\"btn btn-primary\">詳細</button>";
44                     s=s+"<input type=\"hidden\" name=\"id\" value=\""+a[i].id+"\">";
45                     s=s+"</form>";
46                     s=s+"</td>";
47                     s=s+"</tr>";
48                 }
49                 s=s+"</table>";

```



```

50     }
51     }
52     document.getElementById("kekka").innerHTML=s;
53 }
54 }

```

6.7 badrequest.erb

リスト 31 に badrequest.erb のコードを示す.

リスト 31: badrequest.erb

```

1 Bad request. Please login this service first.<br>
2
3 <a href="/login">Back to login screen.</a>

```

6.8 contents.erb

リスト 32 に contents.erb のコードを示す.

リスト 32: contents.erb

```

1 <script type="text/javascript" src="search.js"></script>
2 <h2>Welcome <font color="#32cd32"><%= @a %> </font> !</h2>
3
4 <a class="btn btn-link" data-toggle="collapse" href="/myarticle" role="button"
5   aria-expanded="false" aria-controls="collapseExample">
6   MyArticle
7 </a>
8
9 <a class="btn btn-link" data-toggle="collapse" href="/ranking" role="button"
10  aria-expanded="false" aria-controls="collapseExample">
11 ランキング
12 </a>
13
14 <a class="btn btn-link" data-toggle="collapse" href="/beginer" role="button"
15  aria-expanded="false" aria-controls="collapseExample">
16 初めての方へ
17 </a>
18 <br>
19
20 <form>
21   タイトル名 : <input type="text" id="searchform" onKeyUp="doSearch();"
22   placeholder="find article..." maxlength=40>
23 </form>
24 <div id="kekka"></div>
25
26 <br>
27 <a class="btn btn-link" data-toggle="collapse" href="/logout" role="button"
28  aria-expanded="false" aria-controls="collapseExample">
29 Logout
30 </a>

```

6.9 edit.erb

リスト 33 に edit.erb のコードを示す.

リスト 33: edit.erb

```
1 <h3>記事を更新します!</h3>
2
3 <div class="ml-2" style="width: 1200px;">
4 <form action="/authedit" method="post">
5 <div class="form-check">
6   <input class="form-check-input" name="publicbutton[]" value="checked"
7   type="checkbox" id="defaultCheck1">
8   <label class="form-check-label" for="defaultCheck1">
9     公開
10  </label>
11 </div>
12 <div class="form-group">
13   <label for="exampleFormControlInput1">タイトル</label>
14   <input type="text" class="form-control" name="title" maxlength="60"
15   value="%= @a.title %">
16 </div>
17 <br>
18 <div class="form-group">
19   <label for="exampleFormControlTextarea1">記事の内容</label>
20   <textarea class="form-control" name="description" id="exampleFormControlTextarea1"
21   rows="15" maxlength="1000">%= @a.description %</textarea>
22 </div>
23
24 <div class="form-group">
25   <label for="exampleFormControlTextarea1">ソースコード</label>
26   <textarea class="form-control" name="code" id="exampleFormControlTextarea1"
27   rows="15" maxlength="1000">%= @a.code %</textarea>
28 </div>
29 <div class="form-group">
30   <label for="exampleFormControlTextarea1">実行結果</label>
31   <textarea class="form-control" name="result" id="exampleFormControlTextarea1"
32   rows="15" maxlength="1000">%= @a.result %</textarea>
33 </div>
34 <button type="submit" class="btn btn-primary">Submit</button>
35 <input type="hidden" name="id" value="%= @a.id %">
36 </form>
37 </div>
```

6.10 failure1Resister.erb

リスト 34 に failure1Resister.erb のコードを示す.

リスト 34: failure1Resister.erb

```
1 Resister failed.<br>
2 Username has already been registered.<br>
3
4 <a href="/login">Login</a>
5
6 <a href="/createaccount">make Account</a><br>
```

6.11 failure3Resister.erb

リスト 35 に failure3Resister.erb のコードを示す.

リスト 35: failure3Resister.erb

```
1 Failed.<br>
2 You have entered an incorrect field.<br>
3
4 <a href="/login">Login</a><br>
```

```
5 <a href="/createaccount">make Account</a>
6
```

6.12 layout.erb

リスト 36 に layout.erb のコードを示す.

リスト 36: layout.erb

```
1 <!DOCTYPE html>
2 <html>
3 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/
4 4.5.0/css/bootstrap.min.css" integrity="sha384-9aIt2nRpC12Uk9gS9baD1411NQA
5 pFmC26EwA0H8WgZl5MYXxFfc+NcPb1dKGj7Sk" crossorigin="anonymous">
6 <head>
7 <meta charset="utf-8" />
8 <title>Chama code</title>
9 </head>
10
11 <body>
12 <div class="bg-dark text-white">
13 <center>
14 <h1>Chama code</h1>
15 </center>
16 </div>
17
18 <div class="bg-light text-black mx-2">
19 <%= yield %>
20 </div>
21 </body>
22 </html>
```

6.13 logout.erb

リスト 37 に logout.erb のコードを示す.

リスト 37: logout.erb

```
1 User has logged out.<br>
2
3 <a href="/login">Back to login screen.</a>
```

6.14 myarticle.erb

リスト 38 に myarticle.erb のコードを示す.

リスト 38: myarticle.erb

```
1 <script src="deleteCheck.js"></script>
2 <center>
3 <h1>MyArticle</h1>
4 </center>
5 <span class="row mx-2">
6 <h2><font color="#32cd32"><%= @a %> </font>さんの記事一覧です</h2>
7 </span>
8
9 <div class="text-right mr-5" >
10 <a class="btn btn-success" data-toggle="collapse" href="/newarticle" role="button"
11 aria-expanded="false" aria-controls="collapseExample">
```

```

12 New Article
13 </a>
14 </div>
15
16 <a class="btn btn-link" data-toggle="collapse" href="/contentspage" role="button"
17 aria-expanded="false" aria-controls="collapseExample">
18 Contents
19 </a>
20 <br>
21
22 <br>
23 <% if @isarticle>=1 then %>
24
25 <table class="table table-bordered">
26 <tr>
27 <th>タイトル</th>
28 <th>公開状況</th>
29 <th>更新日時</th>
30 <th>いいね数</th>
31
32 </tr>
33 <% @s.each do |a| %>
34 <tr>
35 <% if a.title.size>20 %>
36 <td><%= a.title[0,20]+"..." %></td>
37 <% else %>
38 <td><%= a.title %></td>
39 <% end %>
40 <td>
41 <% if a.open==1 then%>
42 公開
43 <% else %>
44 非公開
45 <%end%>
46 </td>
47 <td><%= a.date %></td>
48 <td><%= a.good %></td>
49 <form method="post" action="detail">
50 <td><button type="submit" class="btn btn-primary">詳細</button></td>
51 <input type="hidden" name="id" value="<%= a.id %>">
52 </form>
53
54 <form method="post" action="edit">
55 <td><button type="submit" class="btn btn-success">編集</button></td>
56 <input type="hidden" name="id" value="<%= a.id %>">
57 </form>
58
59 <form method="post" action="del">
60 <td><button type="submit" class="btn btn-danger" onclick="return deleteCheck();">
61 削除</button></td>
62 <input type="hidden" name="id" value="<%= a.id %>">
63 <input type="hidden" name="_method" value="delete">
64 </form>
65 </tr>
66 <% end %>
67 </table>
68 <% else %>
69 <div class="row mx-2">
70 you have no article.
71 </div>
72 <% end %>
73
74 <br>
75 <a class="btn btn-link" data-toggle="collapse" href="/logout" role="button"
76 aria-expanded="false" aria-controls="collapseExample">
77 Logout
78 </a>

```

6.15 ranking.erb

リスト 39 に ranking.erb のコードを示す.

リスト 39: ranking.erb

```
1 <script src="deleteCheck.js"></script>
2
3 <center>
4 <h1>ランキング</h1>
5 </center>
6
7 <a class="btn btn-link" data-toggle="collapse" href="/contentspage" role="button"
8 aria-expanded="false" aria-controls="collapseExample">
9 Contents
10 </a>
11
12 <br>
13 <% if @isarticle>=1 then %>
14
15 <table class="table table-bordered">
16 <tr>
17 <th>順位</th>
18 <th>タイトル</th>
19 <th>投稿者</th>
20 <th>更新日時</th>
21 <th>いいね数</th>
22
23 <% rank=1 %>
24 </tr>
25 <% @s.each do |a| %>
26 <tr>
27 <td><%= rank %></td>
28 <% rank+=1 %>
29 <% if a.title.size>20 %>
30 <td><%= a.title[0,20]+"..." %></td>
31 <% else %>
32 <td><%= a.title %></td>
33 <% end %>
34 <td><%= a.username %></td>
35 <td><%= a.date %></td>
36 <td><%= a.good %></td>
37 <form method="post" action="detail">
38 <td><button type="submit" class="btn btn-primary">詳細</button></td>
39 <input type="hidden" name="id" value="<%= a.id %>">
40 </form>
41 </tr>
42 <% end %>
43 </table>
44 <% else %>
45 <div class="row mx-2">
46 No article.
47 </div>
48 <% end %>
49
50 <br>
51 <a class="btn btn-link" data-toggle="collapse" href="javascript:history.back()"
52 role="button" aria-expanded="false" aria-controls="collapseExample">
53 戻る
54 </a>
55 <br>
```

6.16 successResister.erb

リスト 40 に successResister.erb のコードを示す.

リスト 40: successResister.erb

```
1 Resister succeeded.<br>
2
3 <a href='/login'>Login</a>
```

6.17 unknownUser.erb

リスト 41 に unknownUser.erb のコードを示す.

リスト 41: unknownUser.erb

```
1 Renew Password failed.<br>
2 No user entered was found.<br>
3
4 <a href='/login'>Login</a>
5
6 <a href='/createaccount'>make Account</a><br>
```

6.18 beginner.erb

リスト 42 に beginner.erb のコードを示す.

リスト 42: beginner.erb

```
1 <center>
2 <h1>Chama codeの使い方</h1>
3 </center>
4
5 <a class="btn btn-link" data-toggle="collapse" href="javascript:history.back()"
6 role="button" aria-expanded="false" aria-controls="collapseExample">
7 戻る
8 </a>
9
10 <div class="mx-2">
11 <h2>初めまして,<font color="#32cd32"><%= @a %> </font>さん!</h2>
12
13 このページでは,Chama codeの概要と使い方解説します!
14 <br>
15 <hr>
16 <h3>目次</h3>
17 1. <a href="#check1">そもそもChama codeって何?</a><br>
18 2. <a href="#check2">記事を投稿しよう!</a><br>
19 3. <a href="#check3">記事を探索しよう!</a><br>
20 4. <a href="#check4">ランキングを見よう!</a><br>
21 <br>
22 <hr>
23 <h3 id="check1">そもそもChama codeって何?</h3>
24 Chama codeは4JネットワークプログラミングIの課題で作成したWebアプリです.
25 作成者は金澤雄大です.このアプリはユーザーのソースコードを共有することを目的に
26 作られています.公開したソースコードは他のユーザーが「いいね」をつけることができます.
27 「いいね」が多い投稿はランキングの上位に表示されます.記事のランクインを目指しましょう!
28 <br>
29 <hr>
30 <h3 id="check2">記事を投稿しよう!</h3>
31 まずは,<font color="#32cd32"><%= @a %> </font>さんの記事を投稿しましょう.
32 記事の投稿は次の手順で行うことができます.<br>
33 1. <a href="/myarticle">MyArticle</a>の「New Article」から投稿フォームを開きます.<br>
34 2. 投稿内容(タイトル,説明,ソースコード,実行結果)を書きます.タイトルは60文字,
35 説明,ソースコード,実行結果は100文字まで入力できます.ただし,HTMLやJavaScript
36 のタグは反映されません.また,記事は公開/非公開の2つの状態があり,「公開」のボ
37 タンにチェックをいれると全てのユーザーが閲覧できるようになります.
38 ボタンにチェックを入れない場合は投稿者本人のみがその記事を閲覧できます.<br>
```

```

39 <br>
40 3. 投稿フォームの一番下にある「Submit」をクリックして投稿しましょう.<br>
41 4. 投稿した記事は<a href="/myarticle">MyArticle</a>に表示されます.投稿内容を表示したい
42 ときは,「詳細」をクリックしましょう.投稿内容を編集したいときは「編集」をクリックしましょう
43 .投稿内容を削除したいときは「削除」をクリックしましょう.<br>
44 <br>
45 <hr>
46 <h3 id="check3">記事を検索しよう!</h3>
47 ユーザーが公開している記事は誰でも見ることができます.公開されている記事を検索してみま
48 しょう.検索は<a href="/contentspage">Contents</a>のフォームから行えます.読みたい記
49 事にマッチしそうな単語を入力しましょう.検索は入力した単語と記事のタイトルで行っています
50 .記事の内容で検索はできません.<br>他のユーザーの記事は,記事の詳細の一番下に「いいね」
51 ボタンがあります.素晴らしい記事には「いいね」を押しましょう!<br>
52 <br>
53 <hr>
54 <h3 id="check4">ランキングを見よう!</h3>
55 公開されている「いいね」が多い記事のランキング見ることができます.
56 ランキングは<a href="/ranking">こちら</a>
57 </div>
58
59 <hr>
60 <a class="btn btn-link" data-toggle="collapse" href="javascript:history.back()"
61 role="button" aria-expanded="false" aria-controls="collapseExample">
62 戻る
63 </a>

```

6.19 detail.erb

リスト 43 に detail.erb のコードを示す.

リスト 43: detail.erb

```

1 <a class="btn btn-link" data-toggle="collapse" href="javascript:history.back()"
2 role="button" aria-expanded="false" aria-controls="collapseExample">
3 戻る
4 </a>
5
6 <br>
7
8 <h2> <%= @a.title %> @<font color="#32cd32"><%= @a.username %></font> </h2>
9
10 更新日 : <%= @a.date %> <br>
11 いいね数 : <%= @a.good %>
12
13 <br>
14 <div style="white-space:pre-wrap;">
15 <%= @a.description %>
16 </div>
17
18 <br>
19
20 <h4>ソースコード</h4>
21
22 <div class="row mx-4" style="white-space:pre-wrap; padding: 5px; margin-bottom: 5px;
23 border: 1px solid #333333; border-radius: 10px;" >
24 <pre><code>
25 <%= @a.code %>
26 </code></pre>
27 </div>
28
29 <br>
30
31 <h4>実行結果</h4>
32
33 <div class="row mx-4" style="white-space:pre-wrap; padding: 5px; margin-bottom: 5px;
34 border: 1px solid #333333; border-radius: 10px;" >
35 <pre><code>

```

```

36 <%= @a.result %>
37 </code></pre>
38 </div>
39
40 <br>
41
42 <% if !@user_flg then %>
43 <div class="row mx-2">
44 <form method="post" action="good">
45 <button type="submit" class="btn btn-primary">good</button>
46 <input type="hidden" name="id" value="<%= @a.id %>">
47 </form>
48 </div>
49 <% end %>
50 <br>
51 <a class="btn btn-link" data-toggle="collapse" href="javascript:history.back()"
52 role="button" aria-expanded="false" aria-controls="collapseExample">
53 戻る
54 </a>

```

6.20 failure.erb

リスト 44 に failure.erb のコードを示す.

リスト 44: failure.erb

```

1 Login failed.<br>
2
3 <a href="/login">Back to login screen.</a>

```

6.21 failure2Resister.erb

リスト 45 に failure2Resister.erb のコードを示す.

リスト 45: failure2Resister.erb

```

1 Resister failed.<br>
2 The password you entered does not match.<br>
3
4 <a href="/login">Login</a><br>
5
6 <a href="/createaccount">make Account</a>

```

6.22 forgetpass.erb

リスト 46 に forgetpass.erb のコードを示す.

リスト 46: forgetpass.erb

```

1 <center>
2 <h1>Renew Password</h1>
3 </center>
4
5 パスワードを変更します. ユーザー名と新しいパスワードを入力してください.<br>
6
7 <div class="mx-auto" style="width: 300px;">
8 <form action="/newpass" method="post">
9 <div class="form-group">
10 <label for="exampleFormControlInput1">Username</label>

```



```

11     <input type="text" class="form-control" name="uname" maxlength="40"
12     placeholder="user">
13   </div>
14   <div class="form-group">
15     <label for="exampleFormControlInput1">New Password</label>
16     <input type="password" class="form-control" maxlength="40" name="pass1" >
17   </div>
18   <div class="form-group">
19     <label for="exampleFormControlInput1">New Password Again</label>
20     <input type="password" class="form-control" maxlength="40" name="pass2">
21   </div>
22   <br>
23
24   <button type="submit" class="btn btn-primary">Renew</button>
25   <button type="reset" class="btn btn-danger">Reset</button>
26 </form>
27 <a class="btn btn-link" data-toggle="collapse" href="/login" role="button"
28   aria-expanded="false" aria-controls="collapseExample">
29   Login
30 </a>
31 <br>
32 <a class="btn btn-link" data-toggle="collapse" href="/createaccount"
33   role="button" aria-expanded="false" aria-controls="collapseExample">
34   Create Account
35 </a>
36 </div>

```

6.23 loginscr.erb

リスト 47 に loginscr.erb のコードを示す.

リスト 47: loginscr.erb

```

1 <center>
2 <h1>Login</h1>
3 </center>
4
5 <br>
6 ユーザー名とパスワードを入力してください。
7
8 <div class="mx-auto" style="width: 300px;">
9   <form action="/auth" method="post">
10     <div class="form-group">
11       <label for="exampleFormControlInput1">Username</label>
12       <input type="text" class="form-control" name="uname" maxlength="40"
13       placeholder="user">
14     </div>
15     <div class="form-group">
16       <label for="exampleFormControlInput1">Password</label>
17       <input type="password" class="form-control" maxlength="40" name="pass" >
18     </div><br>
19   <br>
20   <button type="submit" class="btn btn-primary">Login</button>
21   <button type="reset" class="btn btn-danger">Reset</button>
22 </form>
23 <br>
24 <a class="btn btn-link" data-toggle="collapse" href="/createaccount"
25   role="button" aria-expanded="false" aria-controls="collapseExample">
26   Create Account
27 </a>
28 <br>
29 <a class="btn btn-link" data-toggle="collapse" href="/forgetpass"
30   role="button" aria-expanded="false" aria-controls="collapseExample">
31   Forget Password
32 </a>
33 </div>

```

6.24 makeAccount.erb

リスト 48 に makeAccount.erb のコードを示す.

リスト 48: makeAccount.erb

```
1 <center>
2 <h1>New Account</h1>
3 </center>
4
5 アカウントを作成します. ユーザー名とパスワードを入力してください.
6
7 <div class="mx-auto" style="width: 300px;">
8 <form action="/newaccount" method="post">
9 <div class="form-group">
10   <label for="exampleFormControlInput1">Username</label>
11   <input type="text" class="form-control" name="uname" maxlength="40"
12     placeholder="user">
13 </div>
14 <div class="form-group">
15   <label for="exampleFormControlInput1">Password</label>
16   <input type="password" class="form-control" maxlength="40" name="pass1" >
17 </div>
18 <div class="form-group">
19   <label for="exampleFormControlInput1">Password Again</label>
20   <input type="password" class="form-control" maxlength="40" name="pass2" >
21 </div>
22 <br>
23
24 <button type="submit" class="btn btn-primary">Create</button>
25 <button type="reset" class="btn btn-danger">Reset</button>
26 </form>
27 <a class="btn btn-link" data-toggle="collapse" href="/login" role="button"
28   aria-expanded="false" aria-controls="collapseExample">
29 Login
30 </a>
31 <br>
32 <a class="btn btn-link" data-toggle="collapse" href="/forgetpass" role="button"
33   aria-expanded="false" aria-controls="collapseExample">
34 Forget Password
35 </a>
36 </div>
```

6.25 newarticle.erb

リスト 49 に newarticle.erb のコードを示す.

リスト 49: newarticle.erb

```
1 <span class="row mx-2">
2 <h3>新しい記事を作成します!</h3>
3 </span>
4
5 <div class="ml-2" style="width: 1200px;">
6 <form action="/autharticle" method="post">
7 <div class="form-check">
8   <input class="form-check-input" name="publicbutton[]" value="checked"
9     type="checkbox" id="defaultCheck1">
10   <label class="form-check-label" for="defaultCheck1">
11     公開
12   </label>
13 </div>
14 <div class="form-group">
15   <label for="exampleFormControlInput1">タイトル</label>
16   <input type="text" class="form-control" name="title" maxlength="60"
17     placeholder="title">
```

```

18     </div>
19 <br>
20 <div class="form-group">
21   <label for="exampleFormControlTextarea1">記事の内容</label>
22   <textarea class="form-control" name="description" maxlength="1000"
23     id="exampleFormControlTextarea1" rows="15"></textarea>
24 </div>
25
26 <div class="form-group">
27   <label for="exampleFormControlTextarea1">ソースコード</label>
28   <textarea class="form-control" name="code" maxlength="1000"
29     id="exampleFormControlTextarea1" rows="15"></textarea>
30 </div>
31 <div class="form-group">
32   <label for="exampleFormControlTextarea1">実行結果</label>
33   <textarea class="form-control" name="result" maxlength="1000"
34     id="exampleFormControlTextarea1" rows="15"></textarea>
35 </div>
36 <button type="submit" class="btn btn-primary">Submit</button>
37 </form>
38 </div>

```

6.26 successRenewpass.erb

リスト 50 に successRenewpass.erb のコードを示す.

リスト 50: successRenewpass.erb

```

1 Renew Password succeeded.<br>
2
3 <a href="/login">Login</a>

```

6.27 successarticle.erb

リスト 51 に successarticle.erb のコードを示す.

リスト 51: successarticle.erb

```

1 Submit article succeeded.<br>
2
3 <a href="/myarticle">My Article</a>

```

参考文献

- [1] amazon,"<https://www.amazon.com/>", 閲覧日 2020 年 12 月 20 日
- [2] Bootstrap,"<https://getbootstrap.jp/>", 閲覧日 2020 年 12 月 20 日
- [3] Ruby 2.7.0 リファレンスマニュアル escapeHTML,"<https://docs.ruby-lang.org/ja/latest/method/CGI/s/escapeHTML.html>", 閲覧日 2020 年 12 月 20 日