neko_todo ソースリスト

美都

2024 年 12 月 21 日

目次

1 Rust ソース 2
1.1 メインモジュール　main.rs 2
1.2 ライブラリメインモジュール lib.rs 3
1.3 アプリケーションステータス app_status.rs 5
1.4 コンフィグ設定処理 config.rs 6
1.5 アプリケーション設定情報の処理 setup.rs 11
1.6 todo モデル処理 todo.rs 14
1.7 データベースアクセス database.rs 19
2 フロントエンド　React 関係 25
2.1 index.html 25
2.2 メイン CSS ファイル 26
2.3 main.jsx 29
2.4 アプリケーションメイン App.jsx 30
2.5 全体のベースページ BasePage.jsx 31
2.6 ユーザー登録画面 RegistUser.jsx 32
2.7 ログイン画面 Login.jsx 34
2.8 todo リストの表示 TodoList.jsx 36
2.9 todo アイテム表示 todoitem.jsx 38
3 データベース構成 39
3.1 テーブル生成スクリプト create_table.sql 39

1　Rust ソース

1.1　メインモジュール　main.rs

```rust
// Prevents additional console window on Windows in release, DO NOT REMOVE!!
#![cfg_attr(not(debug_assertions), windows_subsystem = "windows")]

use env_logger::builder;

fn main() {
    builder().init();

    neko_todo_lib::run()
}
```

1.2 ライブラリメインモジュール lib.rs

```rust
//! tauri メインプロセス
mod app_status;
mod config;
mod database;
mod setup;
mod todo;

use app_status::AppStatus;
use log::error;
use setup::setup;
use tauri::{command, State};
use todo::{TodoError, TodoItem};

#[tauri::command]
fn greet(name: &str) -> String {
    format!("Hello, {}! You've been greeted from Rust!", name)
}

#[cfg_attr(mobile, tauri::mobile_entry_point)]
pub fn run() {
    let app_status = match setup() {
        Ok(s) => s,
        Err(e) => {
            error!("{}", e);
            std::process::exit(1)
        }
    };

    tauri::Builder::default()
        .plugin(tauri_plugin_shell::init())
        .manage(app_status)
        .invoke_handler(tauri::generate_handler![
            greet,
            get_todo_list,
            regist_user,
            login
        ])
        .run(tauri::generate_context!())
        .expect("error while running tauri application");
}

#[tauri::command]
fn get_todo_list(app_status: State<'_, AppStatus>) -> Result<Vec<TodoItem>, String> {
```

```
    println!("todo 取得");

    Ok(app_status.todo().get_todo_list().unwrap())
}

#[tauri::command]
async fn regist_user(
    app_status: State<'_, AppStatus>,
    name: String,
    password: String,
) -> Result<(), String> {
    match app_status.todo().add_user(&name, &password).await {
        Ok(_) => Ok(()),
        Err(e) => match e {
            TodoError::DuplicateUser(_) => Err("DuplicateUser".to_string()),
            TodoError::HashUserPassword(e) => Err(format!("InvalidPassword.[{}]", e)),
            TodoError::AddUser(e) => Err(e.to_string()),
            _ => unimplemented!("[lib.rs regist_user] add_user 返り値異常"),
        },
    }
}

#[command]
async fn login(
    app_status: State<'_, AppStatus>,
    name: String,
    password: String,
) -> Result<String, String> {
    let session = app_status
        .todo()
        .login(&name, &password)
        .await
        .map_err(|e| match e {
            TodoError::NotFoundUser => "NotFoundUser.".to_string(),
            TodoError::HashUserPassword(e) => format!("InvalidPassword.[{}]", e),
            TodoError::WrongPassword => "WrongPassword".to_string(),
            TodoError::FailLogin(e) => format!("OtherError:{}", e),
            _ => unimplemented!("[lib.rs::login] login から予期しないエラー"),
        })?;

    let mut cnf = app_status.config().lock().unwrap();
    cnf.set_session_id(&session);
    cnf.save().map_err(|e| format!("OtherError:{}", e))?;
    Ok(session.to_string())
}
```

1.3 アプリケーションステータス app_status.rs

```rust
//! アプリケーション全体のステータスを保持する。

use crate::{config::NekoTodoConfig, todo::Todo};
use std::sync::{Arc, Mutex};

pub struct AppStatus {
    config: Arc<Mutex<NekoTodoConfig>>,
    todo: Todo,
}

impl AppStatus {
    pub fn new(config: NekoTodoConfig, todo: Todo) -> Self {
        Self {
            config: Arc::new(Mutex::new(config)),
            todo,
        }
    }

    pub fn config(&self) -> &Mutex<NekoTodoConfig> {
        &self.config
    }

    pub fn todo(&self) -> &Todo {
        &self.todo
    }
}
```

1.4 コンフィグ設定処理 config.rs

```rust
//! アプリケーション設定の取得関係

use directories::BaseDirs;
use std::{
    fs::OpenOptions,
    io::{BufWriter, ErrorKind, Result, Write},
    path::PathBuf,
};
use uuid::Uuid;

const CONF_FILE_NAME: &str = "neko_todo.conf";
const CONF_DIR_NAME: &str = "neko_todo";
const DB_HOST: &str = "NEKO_DB_DB_HOST";
const DB_USER: &str = "NEKO_DB_DB_USER";
const DB_PASS: &str = "NEKO_DB_DB_PASS";
const SESSION: &str = "NEKO_DB_SESSION_ID";

#[derive(Debug)]
pub struct NekoTodoConfig {
    db_host: String,
    db_user: String,
    db_pass: String,
    session_id: Option<Uuid>,
    dirty: bool,
}

impl NekoTodoConfig {
    pub fn new() -> dotenvy::Result<Self> {
        let file = Self::get_config_file_path().map_err(dotenvy::Error::Io)?;
        dotenvy::from_path(file)?;
        let session_id = std::env::var(SESSION)
            .ok()
            .map(|s| Uuid::parse_str(&s).expect("環境ファイル異常:SESSION_ID 不正"));

        Ok(Self {
            db_host: std::env::var(DB_HOST).unwrap_or_default(),
            db_user: std::env::var(DB_USER).unwrap_or_default(),
            db_pass: std::env::var(DB_PASS).unwrap_or_default(),
            session_id,
            dirty: false,
        })
    }

```

```rust
    pub fn get_db_host(&self) -> &str {
        &self.db_host
    }

    pub fn get_db_user(&self) -> &str {
        &self.db_user
    }

    pub fn get_db_pass(&self) -> &str {
        &self.db_pass
    }

    pub fn get_session_id(&self) -> Option<Uuid> {
        self.session_id
    }

    pub fn set_db_host(&mut self, val: &str) {
        self.db_host = val.to_string();
        self.dirty = true;
    }

    pub fn set_db_user(&mut self, val: &str) {
        self.db_user = val.to_string();
        self.dirty = true;
    }

    pub fn set_db_pass(&mut self, val: &str) {
        self.db_pass = val.to_string();
        self.dirty = true;
    }

    pub fn set_session_id(&mut self, uuid: &Uuid) {
        self.session_id = Some(*uuid);
        self.dirty = true;
    }

    pub fn save(&mut self) -> Result<()> {
        if !self.dirty {
            return Ok(());
        }
        let path = Self::get_config_file_path()?;
        let file = OpenOptions::new().write(true).truncate(true).open(&path)?;
        let mut buffer = BufWriter::new(file);
        writeln!(buffer, "{}={}", DB_HOST, self.get_db_host())?;
        writeln!(buffer, "{}={}", DB_USER, self.get_db_user())?;
```

```
89          writeln!(buffer, "{}={}", DB_PASS, self.get_db_pass())?;
90          if let Some(s) = self.session_id {
91              writeln!(buffer, "{}={}", SESSION, s)?;
92          }
93          self.dirty = false;
94          Ok(())
95      }
96
97      /// コンフィグファイルのファイル名を生成する
98      /// 必要に応じて、コンフィグファイル用のディレクトリ("neko_todo")を生成し
99      /// さらに、存在しなければ、空のコンフィグファイル("neko_todo.conf")を生成する。
100     fn get_config_file_path() -> Result<PathBuf> {
101         // 環境依存コンフィグ用ディレクトリの取得
102         let mut path: PathBuf = BaseDirs::new().unwrap().config_dir().into();
103         // 必要であれば、自分用のディレクトリを生成する。
104         // ここでエラーになるのは、OSシステムに問題がある。
105         path.push(CONF_DIR_NAME);
106         if let Err(e) = std::fs::create_dir(&path) {
107             if e.kind() != ErrorKind::AlreadyExists {
108                 return Err(e);
109             }
110         }
111
112         // コンフィグファイルがなければ、空のファイルを生成する。
113         path.push(CONF_FILE_NAME);
114         if let Err(e) = std::fs::File::create_new(&path) {
115             if e.kind() != ErrorKind::AlreadyExists {
116                 return Err(e);
117             }
118         }
119         Ok(path)
120     }
121 }
122
123 impl Drop for NekoTodoConfig {
124     fn drop(&mut self) {
125         if self.dirty {
126             self.save().unwrap();
127         }
128     }
129 }
130
131 #[cfg(test)]
132 mod tests {
133     use super::*;
```

```rust

    /// 環境設定の挙動テスト
    #[test]
    #[ignore]
    fn test_env_val() {
        let val_db_host = "test_host";
        let val_db_user = "test_user";
        let val_db_pass = "test_pass";
        save_curr_conf_file();
        {
            let mut conf = NekoTodoConfig::new().unwrap();
            // 初期状態では空文字列が返るはず
            assert_eq!(conf.get_db_host(), "");
            assert_eq!(conf.get_db_user(), "");
            assert_eq!(conf.get_db_pass(), "");
            // test_host をセットしてセットされているか確認。
            conf.set_db_host(val_db_host);
            conf.set_db_user(val_db_user);
            conf.set_db_pass(val_db_pass);
            assert_eq!(conf.get_db_host(), val_db_host);
            assert_eq!(conf.get_db_user(), val_db_user);
            assert_eq!(conf.get_db_pass(), val_db_pass);
        } // この時点で一旦環境ファイルを保存してみる。
          // 環境ファイルをもう一度ロードして、環境を確認
        delete_env_val();
        let conf = NekoTodoConfig::new().unwrap();
        assert_eq!(conf.get_db_host(), val_db_host);
        assert_eq!(conf.get_db_user(), val_db_user);
        assert_eq!(conf.get_db_pass(), val_db_pass);
        restore_curr_conf_file();
    }

    /// テスト環境のため、元の conf ファイルを退避
    fn save_curr_conf_file() {
        let file = NekoTodoConfig::get_config_file_path().unwrap();
        let mut save_file = file.clone();
        save_file.set_extension("save");
        if file.exists() {
            println!(
                "現在の環境ファイル [{:?}] を [{:?}] に退避します。",
                &file, &save_file
            );
            std::fs::rename(file, save_file).unwrap();
        }
    }
```

```
179
180     /// テスト環境のための一時ファイルを抹消し、元のファイルを復旧
181     fn restore_curr_conf_file() {
182         let file = NekoTodoConfig::get_config_file_path().unwrap();
183         let mut save_file = file.clone();
184         save_file.set_extension("save");
185         if save_file.exists() {
186             if file.exists() {
187                 println!("テスト用環境ファイル{:?}を削除します。", &file);
188                 std::fs::remove_file(&file).unwrap();
189             }
190             println!(
191                 "元の環境ファイル [{:?}] を [{:?}] に復元します。",
192                 &save_file, &file
193             );
194             std::fs::rename(save_file, file).unwrap();
195         }
196     }
197
198     /// テスト環境のため、環境変数をすべて消去する。
199     fn delete_env_val() {
200         std::env::remove_var(DB_HOST);
201         std::env::remove_var(DB_USER);
202         std::env::remove_var(DB_USER);
203     }
204 }
```

1.5 アプリケーション設定情報の処理 setup.rs

```rust
//! アプリケーション環境の構築を実施する
use clap::Parser;
use log::error;
use std::process::exit;
use tauri::async_runtime::block_on;
use thiserror::Error;

use crate::{
    app_status::AppStatus,
    config::NekoTodoConfig,
    todo::{Todo, TodoError},
};

/// アプリケーション環境の構築を行う。
pub fn setup() -> Result<AppStatus, SetupError> {
    let args = Args::parse();
    if args.setup {
        database_param_setup(&args)?;
    }

    let conf = NekoTodoConfig::new()?;

    if conf.get_db_host().is_empty()
        || conf.get_db_user().is_empty()
        || conf.get_db_pass().is_empty()
    {
        return Err(SetupError::Argument);
    }

    let todo = block_on(async {
        Todo::new(conf.get_db_host(), conf.get_db_user(), conf.get_db_pass()).await
    })?;

    Ok(AppStatus::new(conf, todo))
}

/// データベース接続パラメータの設定を設定ファイルに行い終了する。
fn database_param_setup(args: &Args) -> Result<(), SetupError> {
    let Some(ref host) = args.server else {
        return Err(SetupError::Argument);
    };
    let Some(ref user) = args.user else {
        return Err(SetupError::Argument);
```

```
    };
    let Some(ref pass) = args.pass else {
        return Err(SetupError::Argument);
    };

    // 一度試しに接続してみる。
    eprintln!("次のパラメータを使用します。");
    eprintln!("ホスト名:{}", host);
    eprintln!("ユーザー名:{}", user);
    eprintln!("パスワード:{}", pass);
    eprintln!("データベースへの接続を試行します。");
    block_on(async { Todo::new(host, user, pass).await })?;

    eprintln!("データベースへの接続に成功しました。");
    eprintln!("設定ファイルに接続情報を保存します。");
    {
        let mut conf = match NekoTodoConfig::new() {
            Ok(c) => Ok(c),
            Err(e) => Err(SetupError::SetupFile(e)),
        }?;

        conf.set_db_host(host);
        conf.set_db_user(user);
        conf.set_db_pass(pass);
    }
    eprintln!("アプリケーションを終了します。");
    exit(0);
}

/// アプリケーション引数の定義
#[derive(Parser, Debug)]
#[command(version, about)]
struct Args {
    /// データベース接続情報のセットアップを行う。
    #[arg(long)]
    setup: bool,
    /// データベースのサーバー名
    #[arg(short, long)]
    server: Option<String>,
    /// データベースのユーザー名
    #[arg(short, long)]
    user: Option<String>,
    /// データベースのパスワード
    #[arg(short, long)]
    pass: Option<String>,
```

```
89  }
90
91  #[derive(Error, Debug)]
92  pub enum SetupError {
93      #[error("設定ファイルへのアクセスに失敗")]
94      SetupFile(#[from] dotenvy::Error),
95      #[error("--setup 時には、server,user,pass の設定が必須です")]
96      Argument,
97      #[error("データベースへの接続に失敗")]
98      ConnectDatabase(#[from] TodoError),
99  }
```

1.6 todo モデル処理 todo.rs

```rust
use std::f32::consts::E;

use bcrypt::{hash, verify, DEFAULT_COST};
use chrono::{Days, Local, NaiveDate};
use log::error;
use serde::Serialize;
use thiserror::Error;
use uuid::Uuid;

use crate::database::*;

/// todo 一件の内容
#[derive(Serialize)]
pub struct TodoItem {
    title: String,
    work: String,
    update: NaiveDate,
    start: NaiveDate,
    end: NaiveDate,
    done: bool,
}

/// todo リストの処理全般
pub struct Todo {
    database: Database,
}

impl Todo {
    /// 初期化
    pub async fn new(host: &str, user: &str, pass: &str) -> Result<Self, TodoError> {
        let db = Database::new(host, user, pass).await.map_err(|e| match e {
            DbError::FailConnect(e2) => TodoError::DbInit(e2),
            e => unimplemented!("[ToDo::new] Database::new()[{e}]"),
        })?;
        Ok(Self { database: db })
    }

    /// todo の一覧を取得する。(仮実装。インターフェース未確定)
    pub fn get_todo_list(&self) -> Result<Vec<TodoItem>, String> {
        let now_date = Local::now().naive_local().date();
        let todo = TodoItem {
            title: "テスト 1".to_string(),
            work: "なにしようかな".to_string(),
```

```
            update: now_date,
            start: now_date + Days::new(1),
            end: now_date + Days::new(5),
            done: false,
        };
        let todo2 = TodoItem {
            title: "テスト 2".to_string(),
            work: "こんどはなにしよう。".to_string(),
            update: now_date + Days::new(1),
            start: now_date + Days::new(5),
            end: now_date + Days::new(20),
            done: true,
        };
        let ret = vec![todo, todo2];
        Ok(ret)
    }

    /// ユーザーの追加を行う。
    pub async fn add_user(&self, name: &str, password: &str) -> Result<(), TodoError> {
        let hashed_pass = hash(password, DEFAULT_COST)?;
        if let Err(e) = self.database.add_user(name, &hashed_pass).await {
            match e {
                DbError::DuplicateUserName(e) => return Err(TodoError::DuplicateUser(e)),
                DbError::FailDbAccess(e) => {
                    error!("[Todo::add_user]Database::add_user:[{e}]");
                    return Err(TodoError::AddUser(e));
                }
                _ => {}
            }
        }
        Ok(())
    }

    /// ログイン処理を行う。
    pub async fn login(&self, name: &str, password: &str) -> Result<Uuid, TodoError> {
        // 認証
        let user = self.database.get_user(name).await.map_err(|e| match e {
            DbError::NotFoundUser => TodoError::NotFoundUser,
            DbError::FailDbAccess(e) => TodoError::FailLogin(e),
            e => unimplemented!("[ToDo::login] Database::get_user:[{e}]"),
        })?;
        if !verify(password, &user.password)? {
            return Err(TodoError::WrongPassword);
        }
        // セッションの生成
```

```
        let session = self
            .database
            .make_new_session(&user.name)
            .await
            .map_err(|e| match e {
                DbError::NotFoundUser => TodoError::NotFoundUser,
                DbError::FailDbAccess(e) => TodoError::FailLogin(e),
                e => {
                    unimplemented!("[Todo::login] Database::make_new_session:[{e}]")
                }
            })?;
        Ok(session)
    }

    /// 現在のログインの有効性を確認し、セッション ID を更新する。
    /// もし指定されたセッション ID が無効な場合は、None を返す。
    /// セッションが有効な場合は、更新されたセッション ID を返す。
    pub async fn is_valid_session(&self, sess: &Uuid) -> Result<Option<Uuid>, TodoError> {
        let is_valid = self
            .database
            .is_session_valid(sess)
            .await
            .map_err(|e| match e {
                DbError::FailDbAccess(e) => TodoError::CheckValidSession(e),
                e => {
                    unimplemented!("[Todo::is_session_valid]is_session_valid:[{e}]")
                }
            })?;
        if is_valid {
            match self.database.update_session(sess).await {
                Ok(s) => Ok(Some(s)),
                Err(DbError::FailDbAccess(e)) => Err(TodoError::CheckValidSession(e)),
                Err(e) => {
                    unimplemented!("[Todo::is_session_valid]update_session:[{e}]")
                }
            }
        } else {
            Ok(None)
        }
    }
}

#[derive(Error, Debug)]
pub enum TodoError {
    #[error("データベース初期化失敗")]
```

```
134     DbInit(sqlx::Error),
135     #[error("新規ユーザーの登録に失敗")]
136     AddUser(sqlx::Error),
137     #[error("すでに、このユーザー名は使用されています。")]
138     DuplicateUser(sqlx::Error),
139     #[error("ユーザーパスワードのハッシュに失敗。")]
140     HashUserPassword(#[from] bcrypt::BcryptError),
141     #[error("ユーザーが見つかりません。")]
142     NotFoundUser,
143     #[error("パスワードが違います。")]
144     WrongPassword,
145     #[error("ログイン失敗 (汎用)")]
146     FailLogin(sqlx::Error),
147     #[error("[is_valid_session] データベースアクセスに失敗")]
148     CheckValidSession(sqlx::Error),
149 }
150
151 #[cfg(test)]
152 mod test {
153     use super::*;
154     use sqlx::MySqlPool;
155
156     impl Todo {
157         fn test_new(pool: MySqlPool) -> Self {
158             Self {
159                 database: Database::new_test(pool),
160             }
161         }
162     }
163
164     #[sqlx::test]
165     async fn new_user_and_login(pool: MySqlPool) {
166         let todo = Todo::test_new(pool);
167         // ユーザー生成
168         let user_name = "testdayo";
169         let user_pass = "passnano";
170         todo.add_user(user_name, user_pass).await.unwrap();
171
172         // 正しいユーザーでログイン
173         let _sess = todo.login(user_name, user_pass).await.unwrap();
174
175         // 間違ったユーザー名でログイン
176         let res = todo.login("detarame", user_pass).await;
177         match res {
178             Ok(_) => assert!(false, "こんなユーザーいないのに、なんでログインできたの?"),
```

```
179             Err(TodoError::NotFoundUser) => {}
180             Err(e) => assert!(false, "おなしなエラーが帰ってきた。{e}"),
181         }
182 
183         // 間違ったパスワードでログイン
184         let res = todo.login(user_name, "detarame").await;
185         match res {
186             Ok(_) => assert!(false, "間違ったパスワードでログインできちゃだめ"),
187             Err(TodoError::WrongPassword) => {}
188             Err(e) => assert!(false, "こんなえらーだめです。{e}"),
189         }
190     }
191 
192     #[sqlx::test]
193     async fn is_valid_session_test(pool: MySqlPool) {
194         let todo = Todo::test_new(pool);
195 
196         // テスト用ユーザーの生成及び、ログイン
197         let user_name = "testdayo";
198         let user_pass = "passwordnano";
199 
200         todo.add_user(user_name, &user_pass).await.unwrap();
201         let sess = todo.login(user_name, user_pass).await.unwrap();
202 
203         // 正しいセッションを検索する。
204         let new_sess = todo.is_valid_session(&sess).await.unwrap();
205         match new_sess {
206             Some(s) => assert_ne!(s, sess, "ログイン後のセッションが更新されていない。"),
207             None => assert!(false, "正しいセッションが見つからなかった。"),
208         };
209 
210         // 間違ったセッションを検索する。
211         let none_sess = todo.is_valid_session(&Uuid::now_v7()).await.unwrap();
212         match none_sess {
213             Some(_) => assert!(false, "こんなセッションがあるわけがない。"),
214             None => {}
215         }
216     }
217 }
```

1.7 データベースアクセス database.rs

```rust
//! データベースの操作を司る

use log::error;
use sqlx::{
    mysql::{MySqlPool, MySqlPoolOptions},
    prelude::*,
    query, query_as,
};
use thiserror::Error;
use uuid::Uuid;

/// neko_db データベース操作関数郡
#[derive(Clone, Debug)]
pub struct Database {
    pool: MySqlPool,
}

impl Database {
    /// 新規生成。
    pub async fn new(host: &str, user: &str, pass: &str) -> Result<Self, DbError> {
        let db_url = format!("mariadb://{}:{}@{}/nekotodo", user, pass, host);
        let pool = MySqlPoolOptions::new()
            .max_connections(10)
            .min_connections(3)
            .connect(&db_url)
            .await
            .map_err(DbError::FailConnect)?;
        Ok(Self { pool })
    }

    /// ユーザーの追加
    pub async fn add_user(&self, name: &str, pass: &str) -> Result<(), DbError> {
        let sql = "insert into users(name, password) values (?, ?);";
        query(sql)
            .bind(name)
            .bind(pass)
            .execute(&self.pool)
            .await
            .map_err(|e| match e {
                sqlx::Error::Database(ref db_err) => {
                    if db_err.kind() == sqlx::error::ErrorKind::UniqueViolation {
                        DbError::DuplicateUserName(e)
                    } else {
```

```
                    DbError::FailDbAccess(e)
                }
            }
            _ => DbError::FailDbAccess(e),
        })?;
        Ok(())
    }

    /// ユーザー名をキーとして、ユーザー情報を取得
    pub async fn get_user(&self, name: &str) -> Result<User, DbError> {
        let sql = "select name, password from users where name = ?;";
        query_as(sql)
            .bind(name)
            .fetch_one(&self.pool)
            .await
            .map_err(|e| match e {
                sqlx::Error::RowNotFound => DbError::NotFoundUser,
                e => DbError::FailDbAccess(e),
            })
    }

    /// セッション情報を新規作成する。
    ///   生成した uuid を返す。
    pub async fn make_new_session(&self, user_name: &str) -> Result<Uuid, DbError> {
        let sql = "insert into sessions(id, user_name) values (?,?);";
        // キー情報の作成
        let id = Uuid::now_v7();

        query(sql)
            .bind(id.to_string())
            .bind(user_name)
            .execute(&self.pool)
            .await
            .map_err(|err| match err {
                sqlx::Error::Database(ref e) => {
                    if e.is_foreign_key_violation() {
                        // 外部キーエラー。存在しないユーザーを指定した。
                        return DbError::NotFoundUser;
                    }
                    DbError::FailDbAccess(err)
                }
                _ => DbError::FailDbAccess(err),
            })?;

        Ok(id)
```

```rust
    }

    /// 指定されたセッションを新規セッションに更新する。
    /// 指定されたセッションは削除され、新たなセッション id を発行する。
    pub async fn update_session(&self, id: &uuid::Uuid) -> Result<Uuid, DbError> {
        let mut tr = self.pool.begin().await.map_err(DbError::FailDbAccess)?;
        // 期限切れのセッション削除
        let sql_old_del = "delete from sessions where expired < now();";
        query(sql_old_del)
            .execute(&mut *tr)
            .await
            .map_err(DbError::FailDbAccess)?;

        // ユーザー ID の特定
        let sql_query_user = "select user_name from sessions where id=?;";
        let user: String = query(sql_query_user)
            .bind(id.to_string())
            .fetch_one(&mut *tr)
            .await
            .map_err(|e| match e {
                sqlx::Error::RowNotFound => DbError::NotFoundSession,
                e => DbError::FailDbAccess(e),
            })?
            .get("user_name");

        // 旧セッションの削除
        let sql_del_curr_sess = "delete from sessions where id = ?;";
        query(sql_del_curr_sess)
            .bind(id.to_string())
            .execute(&mut *tr)
            .await
            .map_err(DbError::FailDbAccess)?;

        // 新セッションの生成
        let sql_create_sess = "insert into sessions(id, user_name) values (?, ?);";
        let id = Uuid::now_v7();
        query(sql_create_sess)
            .bind(id.to_string())
            .bind(user)
            .execute(&mut *tr)
            .await
            .map_err(DbError::FailDbAccess)?;

        tr.commit().await.map_err(DbError::FailDbAccess)?;
        Ok(id)
```

```
134     }
135
136     /// 指定されたセッション ID が有効であるか確認する。
137     /// データベースエラーが発生した場合は、Err(DbError::FailDbAccess) を返す。
138     pub async fn is_session_valid(&self, sess: &Uuid) -> Result<bool, DbError> {
139         // 期限切れのセッションを削除する。
140         let sql_old_del = "delete from sessions where expired < now();";
141         query(sql_old_del)
142             .execute(&self.pool)
143             .await
144             .map_err(DbError::FailDbAccess)?;
145         // 指定セッション ID の有無を確認する。
146         let sql_find_sess = "select count(*) as cnt from sessions where id = ?;";
147         let sess_cnt: i64 = query(sql_find_sess)
148             .bind(sess.to_string())
149             .fetch_one(&self.pool)
150             .await
151             .map_err(DbError::FailDbAccess)?
152             .get("cnt");
153         if sess_cnt == 1 {
154             Ok(true)
155         } else {
156             Ok(false)
157         }
158     }
159 }
160
161 #[derive(FromRow, Debug)]
162 pub struct User {
163     pub name: String,
164     pub password: String,
165 }
166
167 #[derive(Error, Debug)]
168 pub enum DbError {
169     #[error("データベースへの接続に失敗。")]
170     FailConnect(sqlx::Error),
171     #[error("データベース操作失敗 (一般)")]
172     FailDbAccess(sqlx::Error),
173     #[error("User 挿入失敗 (name 重複)")]
174     DuplicateUserName(sqlx::Error),
175     #[error("ユーザーが見つかりません。")]
176     NotFoundUser,
177     #[error("セッション情報更新失敗")]
178     FailUpdateSession(sqlx::Error),
```

```
179     #[error("指定されたセッション id が見つかりません。")]
180     NotFoundSession,
181 }
182
183 #[cfg(test)]
184 mod test {
185     use super::*;
186
187     /// テスト用の Database 生成。テスト用 Pool をインジェクション
188     impl Database {
189         pub(crate) fn new_test(pool: MySqlPool) -> Self {
190             Self { pool }
191         }
192     }
193
194     /// ユーザー生成のテスト
195     #[sqlx::test]
196     async fn test_add_user_and_get_user(pool: MySqlPool) {
197         let db = Database::new_test(pool);
198         db.add_user("hyara", "password").await.unwrap();
199         let user = db.get_user("hyara").await.unwrap();
200         assert_eq!(user.name, "hyara");
201         assert_eq!(user.password, "password");
202         let error_user = db.get_user("naiyo").await;
203         match error_user {
204             Ok(_) => assert!(false, "結果が帰ってくるはずがない。"),
205             Err(DbError::NotFoundUser) => assert!(true),
206             Err(e) => assert!(false, "このエラーはおかしい。{e}"),
207         }
208     }
209
210     /// セッション生成関係の一連のテスト。
211     #[sqlx::test]
212     async fn test_make_new_session(pool: MySqlPool) {
213         println!("まずはテスト用のユーザーの生成");
214         let db = Database::new_test(pool);
215         let user_name = "nekodayo";
216         let password = "password";
217         db.add_user(user_name, password).await.unwrap();
218
219         println!("次に、普通にセッションを作ってみる。");
220         let sess1 = db.make_new_session(user_name).await.unwrap();
221         println!("セッション生成成功 id=[{}]", sess1.to_string());
222
223         println!("次は、存在しないユーザーに対してセッションを生成してみる。");
```

```
224         let sess2 = db.make_new_session("detarame").await;
225         match sess2 {
226             Ok(_) => assert!(false, "このユーザーは存在しなかったはず。"),
227             Err(DbError::NotFoundUser) => assert!(true),
228             Err(e) => assert!(false, "このエラーもおかしい。[{}]", e),
229         }
230
231         println!("普通に、セッションを更新してみる。");
232         let sess3 = db.update_session(&sess1).await.unwrap();
233         assert_ne!(sess1, sess3);
234
235         println!("ないはずのセッションを更新しようとしてみる。");
236         let sess4 = Uuid::now_v7();
237         let sess5 = db.update_session(&sess4).await;
238         match sess5 {
239             Ok(_) => assert!(false, "このセッションはないはずなのに。"),
240             Err(DbError::NotFoundSession) => assert!(true),
241             Err(e) => assert!(false, "セッション更新２回め。失敗するにしてもこれはない{e}"),
242         }
243     }
244
245     /// セッションが有効かどうかを確認するテスト
246     #[sqlx::test]
247     async fn test_is_session_valid(pool: MySqlPool) {
248         let db = Database::new_test(pool);
249
250         println!("テスト用ユーザーの作成");
251         let name = "nekodayo";
252         let pass = "nekodamon";
253         db.add_user(name, pass).await.unwrap();
254
255         println!("新規セッションを生成する。");
256         let sess = db.make_new_session(name).await.unwrap();
257         println!("生成したセッション ID は、[{}] です。", &sess);
258
259         println!("今作ったセッション ID の妥当性を問い合わせてみる。");
260         assert!(db.is_session_valid(&sess).await.unwrap());
261
262         println!("偽セッション ID をいれて、問い合わせてみる。");
263         assert!(!db.is_session_valid(&Uuid::now_v7()).await.unwrap());
264     }
265 }
```

2　フロントエンド　React 関係

2.1　index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Tauri + React</title>
  </head>

  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

2.2 メイン CSS ファイル

```css
.logo.vite:hover {
  filter: drop-shadow(0 0 2em #747bff);
}

.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafb);
}
:root {
  font-family: Inter, Avenir, Helvetica, Arial, sans-serif;
  font-size: 16px;
  line-height: 24px;
  font-weight: 400;

  color: #0f0f0f;
  background-color: #f6f6f6;

  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  -webkit-text-size-adjust: 100%;
}

.container {
  margin: 0;
  padding-top: 10vh;
  display: flex;
  flex-direction: column;
  justify-content: center;
  text-align: center;
}

.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: 0.75s;
}

.logo.tauri:hover {
  filter: drop-shadow(0 0 2em #24c8db);
}

```

```
.row {
  display: flex;
  justify-content: center;
}

a {
  font-weight: 500;
  color: #646cff;
  text-decoration: inherit;
}

a:hover {
  color: #535bf2;
}

h1 {
  text-align: center;
}

input,
button {
  border-radius: 8px;
  border: 1px solid transparent;
  padding: 0.6em 1.2em;
  font-size: 1em;
  font-weight: 500;
  font-family: inherit;
  color: #0f0f0f;
  background-color: #ffffff;
  transition: border-color 0.25s;
  box-shadow: 0 2px 2px rgba(0, 0, 0, 0.2);
}

button {
  cursor: pointer;
}

button:hover {
  border-color: #396cd8;
}
button:active {
  border-color: #396cd8;
  background-color: #e8e8e8;
}

```

```
input,
button {
  outline: none;
}

#greet-input {
  margin-right: 5px;
}

@media (prefers-color-scheme: dark) {
  :root {
    color: #f6f6f6;
    background-color: #2f2f2f;
  }

  a:hover {
    color: #24c8db;
  }

  input,
  button {
    color: #ffffff;
    background-color: #0f0f0f98;
  }
  button:active {
    background-color: #0f0f0f69;
  }
}
```

2.3 main.jsx

```jsx
import React from "react";
import ReactDOM from "react-dom/client";
import { UIProvider, extendTheme } from "@yamada-ui/react";
import App from "./App";
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";

const semantics = {
    colors: {
        primary: "red.500",
    },
    colorSchemes: {
        primary: "blue",
    },
};

const globalStyle = {
    body: {
        bg: "#DCB879",
    },
}

const customTheme = extendTheme({ semantics, styles: {globalStyle} })();

const query_client = new QueryClient();

ReactDOM.createRoot(document.getElementById("root")).render(
    <React.StrictMode>
        <UIProvider theme={customTheme}>
            <QueryClientProvider client={query_client}>
                <App />
            </QueryClientProvider>
        </UIProvider>
    </React.StrictMode>,
);
```

2.4 アプリケーションメイン App.jsx

```
//import reactLogo from "./assets/react.svg";
import "./App.css";
import { createBrowserRouter ,createRoutesFromElements, Route, RouterProvider, } from
↪  "react-router-dom";

import BasePage from "./BasePage.jsx";
import TodoList from "./TodoList.jsx";
import Login from "./Login.jsx";
import RegistUser from "./RegistUser.jsx";

export const routes = createBrowserRouter(
    createRoutesFromElements(
        <>
            <Route element={ <BasePage/> }>
                <Route path="/" element={<TodoList/>}/>
                <Route path="/login" element={<Login/>}/>
                <Route path="/regist_user" element={<RegistUser/>}/>
            </Route>
        </>
    ));

function App() {
    return (
        <RouterProvider router={routes}/>
    );
}
export default App;
```

2.5 全体のベースページ BasePage.jsx

```jsx
import { Outlet } from "react-router-dom";
import "./App.css";


function BasePage() {

    return (
        <>
            <h1> 猫 todo </h1>
            <Outlet/>
        </>
    );
}


export default BasePage;
```

2.6 ユーザー登録画面 RegistUser.jsx

```jsx
/* ユーザー登録画面 */

import { useForm } from "react-hook-form";
import { VStack, FormControl, Input, Button, Text } from "@yamada-ui/react";
import { invoke } from "@tauri-apps/api/core";
import { useState } from 'react';
import { useNavigate } from "react-router-dom";

function RegistUser() {
    const { register, handleSubmit, formState: {errors} } = useForm();
    const [ sendMessage, setSendMessage ] = useState('');
    const navi = useNavigate();
    const onSubmit = async (data) => {
        try {
            setSendMessage('送信中です。');
            await invoke('regist_user', { name: data.name, password: data.pass });
            navi('/login');
        } catch (e) {
            setSendMessage('エラーが発生しました。{'+e+'}');
            console.log(e);
        }
    };

    return (
        <>
            <h1> 新規ユーザー登録 </h1>
            <p> すべての欄を入力してください。 </p>
            <VStack as="form" onSubmit={handleSubmit(onSubmit)}>
                <FormControl
                    isInvalid={!!errors.name}
                    label="ユーザー名"
                    errorMessage={errors?.name?.message}
                >
                    <Input {...register("name", {required: "入力は必須です。"},)}/>
                </FormControl>
                <FormControl
                    isInvalid={!!errors.pass}
                    label="パスワード"
                    errorMessage={errors?.pass?.message}
                >
                    <Input {...register("pass", {required: "入力は必須です。"},)}/>
                </FormControl>
                <Button type="submit"> 送信 </Button>
```

```
44                  <Text>{sendMessage}</Text>
45              </VStack>
46          </>
47      );
48  }
49
50  export default RegistUser;
51
```

2.7 ログイン画面 Login.jsx

```
/* ログイン画面 */

import { useForm } from "react-hook-form";
import { VStack, FormControl, Input, Button, Text } from "@yamada-ui/react";
import { invoke } from "@tauri-apps/api/core";
import { Link } from "react-router-dom";
import { useState } from "react";

function Login() {
    const { register, handleSubmit, formState: {errors} } = useForm();
    const [ sendMessage, setSendMessage ] = useState('');
    const onSubmit = async (data) => {
        try {
            setSendMessage('処理中です。');
            const sess = await invoke('login', { name: data.name, password: data.pass });
            setSendMessage('ログインできました。sessionid=' + sess);
        } catch (e) {
            setSendMessage('エラーが発生しました。{' + e + '}');
            console.log(e);
        }
    };

    return (
        <>
            <Link to="/regist_user">新規ユーザー登録</Link>
            <h1> ログイン </h1>
            <VStack as="form" onSubmit={handleSubmit(onSubmit)}>
                <FormControl
                    isInvalid={!!errors.name}
                    label="ユーザー名"
                    errorMessage={errors?.name?.message}
                >
                    <Input {...register("name", {required: "入力は必須です。"},)}/>
                </FormControl>
                <FormControl
                    isInvalid={!!errors.pass}
                    label="パスワード"
                    errorMessage={errors?.pass?.message}
                >
                    <Input {...register("pass", {required: "入力は必須です。"},)}/>
                </FormControl>
                <Button type="submit" w="30%" ml="auto" mr="auto"> ログイン </Button>
                <Text>{sendMessage}</Text>
```

```
44                </VStack>
45            </>
46        );
47    }
48
49    export default Login;
50
```

2.8 todo リストの表示 TodoList.jsx

```
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { useQuery } from "@tanstack/react-query";
import { Grid, GridItem} from "@yamada-ui/react";
import { invoke } from "@tauri-apps/api/core";
import "./App.css";
import TodoItem from "./todoitem";

const get_todo_list = async () => invoke('get_todo_list') ;

function TodoList() {

    const { data: todos, isLoading: isTodoListLoading , isError, error} = useQuery({
        queryKey: ['get_todo_list'],
        queryFn: get_todo_list,
    });

    const navi = useNavigate();
    const handleClick = () => navi('/login');

    if (isTodoListLoading) {
        return ( <p> loading... </p>);
    }

    if (isError) {
        return ( <p> エラーだよ。{error}</p> );
    }

    console.log(todos);
    return (
        <>
            <button type="button" onClick={handleClick}> 入力欄への遷移 </button>
            <h1>テスト</h1>
            <Grid templateColumns="repeat(4, 1fr)" gap="md">
                {todos?.map( todo_item => {
                    return (
                        <GridItem key={todo_item.title} w="full" rounded="md" bg="primary">
                            <TodoItem item={todo_item}/>
                        </GridItem>
                    )}
                )}
            </Grid>
        </>
```

```
44      );
45  }
46
47
48  export default TodoList;
```

2.9 todo アイテム表示 todoitem.jsx

```jsx
// todo リストの各アイテム
import { SimpleGrid, GridItem } from "@yamada-ui/react";

export default function TodoItem({item}) {
    return (
        <>
            <SimpleGrid w="full" columns={{base: 2, md: 1}} gap="md">
                <GridItem> <p> {item.done? '済': '未'} </p></GridItem>

                <GridItem>
                    <div style={{textAlign:'right', fontSize:'0.7em'}}>
                        {item.update}
                    </div>
                </GridItem>
            </SimpleGrid>
            <p style={{textAlign:'center', fontSize:'1.1em'}}><strong>{item.title}</strong></p>
            <p>{item.work}</p>
            <div style={{fontSize:'0.9em'}}>
                <p>{item.start} ～ {item.end}</p>
            </div>
        </>
    );
}

```

3　データベース構成

3.1　テーブル生成スクリプト create_table.sql

```
# 猫todo 関係のすべての mariadb オブジェクトの生成

create database if not exists nekotodo;

use nekotodo;

create table if not exists users (
    name varchar(128) primary key,
    password varchar(61)
    );

create table if not exists todo (
    id int unsigned auto_increment primary key,
    user_name varchar(128) not null references users(name),
    title varchar(128) not null,
    work varchar(2048),
    update_date date not null,
    start_date date,
    end_date date,
    done bool
    ) ;

create table if not exists tag (
    name varchar(128) primary key
    );

create table if not exists todo_tag (
    todo_id int unsigned references todo(id),
    tag_name varchar(128) references tag(name),
    primary key(todo_id, tag_name)
    );

create table if not exists sessions (
    id varchar(40) primary key,
    user_name varchar(128) references users(name),
    expired timestamp default date_add(current_timestamp, interval 48 hour)
    );

```