neko_todo ソースリスト

美都

2025 年 2 月 1 日

目次

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 directories::ProjectDirs;

fn main() {
    let mut log_file: std::path::PathBuf = ProjectDirs::from("jp", "laki", "nekotodo")
        .unwrap()
        .config_dir()
        .into();
    log_file.push("nekotodo.log");

    fern::Dispatch::new()
        .format(|out, message, record| {
            out.finish(format_args!(
                "{} [{}] {}:{} {}",
                chrono::Local::now().format("%Y/%m/%d %H:%M:%S"),
                record.level(),
                record.file().unwrap(),
                record.line().unwrap(),
                message
            ))
        })
        //.level(log::LevelFilter::Info)
        .level(log::LevelFilter::Debug)
        .chain(std::io::stderr())
        .chain(fern::log_file(log_file).unwrap())
        .apply()
        .unwrap();

    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 database::ItemTodo;
use log::{debug, error, info};
use serde::Deserialize;
use setup::setup;
use tauri::{command, Manager, State};
use uuid::Uuid;

#[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)
        }
    };

    let app = tauri::Builder::default()
        .plugin(tauri_plugin_shell::init())
        .manage(app_status)
        .invoke_handler(tauri::generate_handler![
            greet,
            get_todo_list,
            get_todo_with_id,
            regist_user,
            login,
            is_valid_session,
            add_todo,
            update_done,
            edit_todo,
```

```
        ])
        .build(tauri::generate_context!())
        .expect("error thile build tauri application");
    app.run(|app, event| {
        if let tauri::RunEvent::Exit = event {
            info!("終了処理開始");
            let state = app.state::<AppStatus>();
            state.config().lock().unwrap().save().unwrap();
        }
    });
}

/// todo のリストを取得する。
#[tauri::command]
async fn get_todo_list(app_status: State<'_, AppStatus>) -> Result<Vec<ItemTodo>, String> {
    let sess = match get_curr_session(&app_status) {
        Some(u) => u,
        None => return Err("NotLogin".to_string()),
    };

    app_status
        .todo()
        .get_todo_list(sess, true)
        .await
        .map_err(Into::into)
}

/// todo アイテムを取得する
#[tauri::command]
async fn get_todo_with_id(app_status: State<'_, AppStatus>, id: u32) -> Result<ItemTodo, String> {
    let Some(sess) = get_curr_session(&app_status) else {
        return Err("NotLogin".to_string());
    };

    app_status
        .todo()
        .get_todo_with_id(id, sess)
        .await
        .map_err(Into::into)
}

/// todo を追加する。
#[tauri::command]
async fn add_todo(app_status: State<'_, AppStatus>, item: FormTodo) -> Result<(), String> {
    let sess = match get_cur_session_with_update(&app_status).await {
```

```
        Ok(Some(u)) => u,
        Ok(None) => return Err("NotLogin".to_string()),
        Err(e) => return Err(e),
    };

    debug!("input = {:?}", &item);
    app_status
        .todo()
        .add_todo(sess, &item.into())
        .await
        .map_err(Into::into)
}

/// todo の完了状態を変更する。
#[tauri::command]
async fn update_done(app_status: State<'_, AppStatus>, id: u32, done: bool) -> Result<(), String> {
    let sess = match get_cur_session_with_update(&app_status).await {
        Ok(Some(s)) => s,
        Ok(None) => return Err("NotLogin".to_string()),
        Err(e) => return Err(e),
    };
    app_status
        .todo()
        .change_done(id, sess, done)
        .await
        .map_err(Into::into)
}

/// todo の編集を行う。
#[tauri::command]
async fn edit_todo(
    app_status: State<'_, AppStatus>,
    id: u32,
    item: FormTodo,
) -> Result<(), String> {
    let sess = match get_cur_session_with_update(&app_status).await {
        Ok(Some(u)) => u,
        Ok(None) => return Err("NotLogin".to_string()),
        Err(e) => return Err(e),
    };

    debug!("input => id: {},  item: {:?}", id, &item);
    let mut item: ItemTodo = item.into();
    item.id = id;
    app_status
```

```
        .todo()
        .edit_todo(&item, sess)
        .await
        .map_err(Into::into)
}

/// ユーザー登録
#[tauri::command]
async fn regist_user(
    app_status: State<'_, AppStatus>,
    name: String,
    password: String,
) -> Result<(), String> {
    app_status
        .todo()
        .add_user(&name, &password)
        .await
        .map_err(Into::into)
}

/// ログイン
#[command]
async fn login(
    app_status: State<'_, AppStatus>,
    name: String,
    password: String,
) -> Result<String, String> {
    let session = app_status.todo().login(&name, &password).await?;

    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())
}

/// 現在、有効なセッションが存在するかどうか確認。(ユーザ I/F用)
#[command]
async fn is_valid_session(app_status: State<'_, AppStatus>) -> Result<bool, String> {
    let sess = get_cur_session_with_update(&app_status)
        .await
        .map(|i| i.is_some());
    match sess {
        Ok(sess) => info!("セッション確認 ({})", if sess { "有効" } else { "無効" }),
        Err(ref e) => info!("セション確認エラー ({})", e),
    }
```

```
179     sess
180 }
181
182 /// 現在、有効なセッションを返す。
183 /// 有効なセッションが存在すれば、セッションの更新を行い、期限を延長する。
184 async fn get_cur_session_with_update(app_status: &AppStatus) -> Result<Option<Uuid>, String> {
185     let cur_session = get_curr_session(app_status);
186     let Some(cur_session) = cur_session else {
187         return Ok(None);
188     };
189
190     match app_status.todo().is_valid_session(&cur_session).await {
191         Ok(Some(s)) => {
192             // 更新されたセッションを再登録
193             let mut cnf = app_status.config().lock().unwrap();
194             cnf.set_session_id(&s);
195             //cnf.save().map_err(|e| format!("FailSession:{e}"))?;
196             Ok(Some(s))
197         }
198         Ok(None) => Ok(None),
199         Err(e) => Err(format!("FailSession:{e}")),
200     }
201 }
202
203 /// 現在のセッションを取得する。
204 fn get_curr_session(app_status: &AppStatus) -> Option<Uuid> {
205     let conf = app_status.config().lock().unwrap();
206     conf.get_session_id()
207 }
208
209 /// Todo 項目追加画面データ取得用
210 #[derive(Deserialize, Debug, Clone)]
211 struct FormTodo {
212     title: String,
213     work: Option<String>,
214     start: Option<String>,
215     end: Option<String>,
216 }
217
218 impl From<FormTodo> for ItemTodo {
219     fn from(val: FormTodo) -> Self {
220         let start = val.start.map(|d| d.replace("/", "-").parse().unwrap());
221         let end = val.end.map(|d| d.replace("/", "-").parse().unwrap());
222         ItemTodo {
223             id: 0,
```

```
            user_name: "".to_string(),
            title: val.title,
            work: val.work,
            update_date: None,
            start_date: start,
            end_date: end,
            done: false,
        }
    }
}
```

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

const CONF_FILE_NAME: &str = "neko_todo.conf";
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,
        })
    }

    pub fn get_db_host(&self) -> &str {
```

```
44          &self.db_host
45      }
46
47      pub fn get_db_user(&self) -> &str {
48          &self.db_user
49      }
50
51      pub fn get_db_pass(&self) -> &str {
52          &self.db_pass
53      }
54
55      pub fn get_session_id(&self) -> Option<Uuid> {
56          self.session_id
57      }
58
59      pub fn set_db_host(&mut self, val: &str) {
60          self.db_host = val.to_string();
61          self.dirty = true;
62      }
63
64      pub fn set_db_user(&mut self, val: &str) {
65          self.db_user = val.to_string();
66          self.dirty = true;
67      }
68
69      pub fn set_db_pass(&mut self, val: &str) {
70          self.db_pass = val.to_string();
71          self.dirty = true;
72      }
73
74      pub fn set_session_id(&mut self, uuid: &Uuid) {
75          self.session_id = Some(*uuid);
76          self.dirty = true;
77      }
78
79      pub fn save(&mut self) -> Result<()> {
80          if !self.dirty {
81              return Ok(());
82          }
83          let path = Self::get_config_file_path()?;
84          let file = OpenOptions::new().write(true).truncate(true).open(&path)?;
85          let mut buffer = BufWriter::new(file);
86          writeln!(buffer, "{}={}", DB_HOST, self.get_db_host())?;
87          writeln!(buffer, "{}={}", DB_USER, self.get_db_user())?;
88          writeln!(buffer, "{}={}", DB_PASS, self.get_db_pass())?;
```

```
89          if let Some(s) = self.session_id {
90              writeln!(buffer, "{}={}", SESSION, s)?;
91          }
92          self.dirty = false;
93          Ok(())
94      }
95
96      /// コンフィグファイルのファイル名を生成する
97      /// 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
98      /// 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。
99      fn get_config_file_path() -> Result<PathBuf> {
100         use std::io;
101         // 環境依存コンフィグ用ディレクトリの取得
102         // 必要であれば、自分用のディレクトリを生成する。
103         // ここでエラーになるのは、OSシステムに問題がある。
104         let mut path: PathBuf = ProjectDirs::from("jp", "laki", "nekotodo")
105             .ok_or(io::Error::new(ErrorKind::Other, "Not Found Home"))?
106             .config_dir()
107             .into();
108         if let Err(e) = std::fs::create_dir(&path) {
109             if e.kind() != ErrorKind::AlreadyExists {
110                 return Err(e);
111             }
112         }
113
114         // 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。
115         path.push(CONF_FILE_NAME);
116         if let Err(e) = std::fs::File::create_new(&path) {
117             if e.kind() != ErrorKind::AlreadyExists {
118                 return Err(e);
119             }
120         }
121         Ok(path)
122     }
123 }
124
125 impl Drop for NekoTodoConfig {
126     fn drop(&mut self) {
127         if self.dirty {
128             self.save().unwrap();
129         }
130     }
131 }
132
133 #[cfg(test)]
```

```rust
mod tests {
    use super::*;

    /// 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     }
181
182     /// 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
183     fn restore_curr_conf_file() {
184         let file = NekoTodoConfig::get_config_file_path().unwrap();
185         let mut save_file = file.clone();
186         save_file.set_extension("save");
187         if save_file.exists() {
188             if file.exists() {
189                 println!("テスト用環境ファイル{:?}を削除します。", &file);
190                 std::fs::remove_file(&file).unwrap();
191             }
192             println!(
193                 "元の環境ファイル [{:?}] を [{:?}] に復元します。",
194                 &save_file, &file
195             );
196             std::fs::rename(save_file, file).unwrap();
197         }
198     }
199
200     /// テスト環境のため、環境変数をすべて消去する。
201     fn delete_env_val() {
202         std::env::remove_var(DB_HOST);
203         std::env::remove_var(DB_USER);
204         std::env::remove_var(DB_USER);
205     }
206 }
```

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, info};
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。
    info!("次のパラメータを使用します。");
    info!("ホスト名:{}", host);
    info!("ユーザー名:{}", user);
    info!("パスワード:{}", pass);
    info!("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 })?;

    info!("データベースへの接続に成功しました。");
    info!("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 bcrypt::{hash, verify, DEFAULT_COST};
use chrono::Local;
use log::error;
use thiserror::Error;
use uuid::Uuid;

use crate::database::*;

/// 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 => unreachable!("[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 async fn get_todo_list(
        &self,
        sess: Uuid,
        only_imcomplete: bool,
    ) -> Result<Vec<ItemTodo>, TodoError> {
        let ref_date = Local::now().date_naive();
        self.database
            .get_todo_item(sess, ref_date, only_imcomplete)
            .await
            .map_err(|e| match e {
                DbError::FailDbAccess(e) => TodoError::FailDbAccess(e),
                e => unreachable!("[get_todo_list]get_todo_item[{e}]"),
            })
    }

    /// 新規の todo を追加する
    /// 引数 item の id, user_name, update_date, update_date は無視される。
    pub async fn add_todo(&self, sess: Uuid, item: &ItemTodo) -> Result<(), TodoError> {
        // ユーザー名を取得
```

```
        let user = self
            .database
            .get_user_from_sess(sess)
            .await
            .map_err(|e| match e {
                DbError::NotFoundSession => TodoError::NotFoundSession,
                DbError::FailDbAccess(e) => {
                    error!("[Todo::add_todo]get_user_from_sess:[{e}]");
                    TodoError::FailDbAccess(e)
                }
                e => unreachable!("[add_todo]get_user_from_sess[{e}]"),
            })?;
        // アイテムを登録
        let mut item = item.clone();
        item.user_name = user.name.clone();
        if let Some(ref s) = item.work {
            if s.trim().is_empty() {
                item.work = None;
            }
        }
        self.database
            .add_todo_item(&item)
            .await
            .map_err(|e| match e {
                DbError::FailDbAccess(e) => {
                    error!("[Todo::add_todo]add_todo_item:[{e}]");
                    TodoError::FailDbAccess(e)
                }
                e => unreachable!("[add_todo]add_todo_item[{e}]"),
            })
    }

    /// id と sess を指定して todo を取得する。
    /// 一致する todo がなければ、エラー、TodoError::NotFoundTodo を返す。
    pub async fn get_todo_with_id(&self, id: u32, sess: Uuid) -> Result<ItemTodo, TodoError> {
        self.database
            .get_todo_item_with_id(id, sess)
            .await
            .map_err(|e| match e {
                DbError::NotFoundTodo => TodoError::NotFoundTodo,
                DbError::FailDbAccess(e) => {
                    error!("[Todo::get_todo_with_id]get_todo_item_with_id:[{e}])");
                    TodoError::FailDbAccess(e)
                }
                e => unreachable!("[Todo::get_todo_with_id]get_todo_item_with_id[{e}]"),
```

```
            })
    }

    /// Todo の完了状態を変更する
    pub async fn change_done(&self, id: u32, sess: Uuid, done: bool) -> Result<(), TodoError> {
        self.get_todo_with_id(id, sess).await?;
        self.database
            .change_done(id, done)
            .await
            .map_err(|e| match e {
                DbError::FailDbAccess(e) => {
                    error!("[Todo::change_done]change_done:[{e}]");
                    TodoError::FailDbAccess(e)
                }
                DbError::NotFoundTodo => TodoError::NotFoundTodo,
                e => unreachable!("[change_done]change_done[{e}]"),
            })
    }

    /// Todo の編集を行う。
    pub async fn edit_todo(&self, item: &ItemTodo, sess: Uuid) -> Result<(), TodoError> {
        let mut item = item.clone();
        if let Some(ref s) = item.work {
            if s.trim().is_empty() {
                item.work = None;
            }
        }
        self.get_todo_with_id(item.id, sess).await?;
        self.database.edit_todo(&item).await.map_err(|e| match e {
            DbError::FailDbAccess(e) => {
                error!("[Todo::edit_todo]edit_todo:[{e}]");
                TodoError::FailDbAccess(e)
            }
            DbError::NotFoundTodo => TodoError::NotFoundTodo,
            e => unreachable!("[edit_todo]edit_todo[{e}]"),
        })
    }

    /// 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::FailDbAccess(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::FailDbAccess(e),
            e => unreachable!("[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::FailDbAccess(e),
                e => {
                    unreachable!("[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::FailDbAccess(e),
```

```
                e => {
                    unreachable!("[Todo::is_valid_session]is_session_valid:[{e}]")
                }
            })?;
        if is_valid {
            match self.database.update_session(sess).await {
                Ok(s) => Ok(Some(s)),
                Err(DbError::NotFoundSession) => Ok(None),
                Err(DbError::FailDbAccess(e)) => Err(TodoError::FailDbAccess(e)),
                Err(e) => {
                    unreachable!("[Todo::is_valid_session]update_session:[{e}]")
                }
            }
        } else {
            Ok(None)
        }
    }
}

#[derive(Error, Debug)]
pub enum TodoError {
    #[error("FailInitDatabase")]
    DbInit(sqlx::Error),
    #[error("DuplicateUserName")]
    DuplicateUser(sqlx::Error),
    #[error("InvalidPassword:{0}")]
    HashUserPassword(#[from] bcrypt::BcryptError),
    #[error("NotFoundUser")]
    NotFoundUser,
    #[error("WrongPassword")]
    WrongPassword,
    #[error("NotFoundSession")]
    NotFoundSession,
    #[error("NotFoundTodo")]
    NotFoundTodo,
    #[error("DatabaseError:{0}")]
    FailDbAccess(sqlx::Error),
}

impl From<TodoError> for String {
    fn from(value: TodoError) -> Self {
        value.to_string()
    }
}

```

```rust
#[cfg(test)]
mod test {
    use super::*;
    use sqlx::MySqlPool;

    impl Todo {
        fn test_new(pool: MySqlPool) -> Self {
            Self {
                database: Database::new_test(pool),
            }
        }
    }

    #[sqlx::test]
    async fn new_user_and_login(pool: MySqlPool) {
        let todo = Todo::test_new(pool);
        // ユーザー生成
        let user_name = "testdayo";
        let user_pass = "passnano";
        todo.add_user(user_name, user_pass).await.unwrap();

        // 正しいユーザーでログイン
        let _sess = todo.login(user_name, user_pass).await.unwrap();

        // 間違ったユーザー名でログイン
        let res = todo.login("detarame", user_pass).await;
        match res {
            Ok(_) => unreachable!("こんなユーザーいないのに、なんでログインできたの?"),
            Err(TodoError::NotFoundUser) => {}
            Err(e) => unreachable!("おなしなエラーが帰ってきた。{e}"),
        }

        // 間違ったパスワードでログイン
        let res = todo.login(user_name, "detarame").await;
        match res {
            Ok(_) => unreachable!("間違ったパスワードでログインできちゃだめ"),
            Err(TodoError::WrongPassword) => {}
            Err(e) => unreachable!("こんなえらーだめです。{e}"),
        }
    }

    #[sqlx::test]
    async fn is_valid_session_test(pool: MySqlPool) {
        let todo = Todo::test_new(pool);

```

```
269         // テスト用ユーザーの生成及び、ログイン
270         let user_name = "testdayo";
271         let user_pass = "passwordnano";
272 
273         todo.add_user(user_name, user_pass).await.unwrap();
274         let sess = todo.login(user_name, user_pass).await.unwrap();
275 
276         // 正しいセッションを検索する。
277         let new_sess = todo.is_valid_session(&sess).await.unwrap();
278         match new_sess {
279             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。"),
280             None => unreachable!("正しいセッションが見つからなかった。"),
281         };
282 
283         // 間違ったセッションを検索する。
284         let none_sess = todo.is_valid_session(&Uuid::now_v7()).await.unwrap();
285         if none_sess.is_some() {
286             unreachable!("こんなセッションがあるわけがない。");
287         }
288     }
289 
290     #[sqlx::test]
291     async fn add_todo_test(pool: MySqlPool) {
292         use chrono::Days;
293 
294         let todo = Todo::test_new(pool);
295         let sess = login_for_test(&todo).await;
296 
297         let item1 = ItemTodo {
298             id: 100,
299             user_name: "kore_naihazu".to_string(),
300             title: "テストアイテム 1 件目".to_string(),
301             work: Some("これは、中身を入れる。".to_string()),
302             update_date: None,
303             start_date: Some(Local::now().date_naive() - Days::new(1)),
304             end_date: Some(Local::now().date_naive() + Days::new(5)),
305             done: true,
306         };
307         let item2 = ItemTodo {
308             id: 100,
309             user_name: "kore_naihazu".to_string(),
310             title: "テストアイテム 2 件目 (work=null)".to_string(),
311             work: Some("".to_string()),
312             update_date: None,
313             start_date: Some(Local::now().date_naive() - Days::new(1)),
```

```
            end_date: Some(Local::now().date_naive() + Days::new(5)),
            done: true,
        };
        let item3 = ItemTodo {
            id: 100,
            user_name: "kore_naihazu".to_string(),
            title: "テストアイテム 3 件目 (work=space)".to_string(),
            work: Some(" \t  ".to_string()),
            update_date: None,
            start_date: Some(Local::now().date_naive() - Days::new(1)),
            end_date: Some(Local::now().date_naive() + Days::new(5)),
            done: true,
        };
        todo.add_todo(sess, &item1)
            .await
            .expect("1 件目の追加に失敗");
        let res = todo
            .get_todo_list(sess, true)
            .await
            .expect("1 件目の取得に失敗");
        assert_eq!(res.len(), 1, "一件目が取得できなかった?");
        assert_eq!(res[0].title, item1.title, "一件目の title が違う");
        assert_eq!(res[0].work, item1.work, "一件目の work が違う");
        assert_eq!(res[0].user_name, "testdayo", "一件目の user_name が違う");
        assert_eq!(
            res[0].update_date,
            Some(Local::now().date_naive()),
            "一件目の update_date が違う"
        );
        assert_eq!(res[0].start_date, item1.start_date, "一件目の開始日が違う");
        assert_eq!(res[0].end_date, item1.end_date, "一件目の終了日が違う");
        assert!(!res[0].done, "一件目の完了マークが違う");

        todo.add_todo(sess, &item2)
            .await
            .expect("二件目の追加に失敗");
        let res = todo
            .get_todo_list(sess, true)
            .await
            .expect("二件目の取得に失敗");
        assert_eq!(res.len(), 2, "二件あるはずなんだけど");
        assert!(
            res.iter()
                .find(|&x| match x.title.find("work=null") {
                    Some(n) => n > 0,
```

```
                    None => false,
                })
                .expect("二件目に追加したデータがない")
                .work
                .is_none(),
            "二件目の work は None のはず"
        );
        todo.add_todo(sess, &item3)
            .await
            .expect("三件目の追加に失敗");
        let res = todo
            .get_todo_list(sess, true)
            .await
            .expect("三件目の取得に失敗");
        assert_eq!(res.len(), 3, "三件あるはずですよ。");
        assert!(
            res.iter()
                .find(|&x| match x.title.find("work=space") {
                    Some(n) => n > 0,
                    None => false,
                })
                .expect("三件目のデータがないよ?")
                .work
                .is_none(),
            "三件目のデータは None に変換してくれてるはず。"
        );
    }

    #[sqlx::test]
    async fn change_done_test(pool: MySqlPool) {
        let todo = Todo::test_new(pool);
        let sess = login_for_test(&todo).await;
        create_todo_for_test(&todo, sess).await;

        let items = todo.get_todo_list(sess, true).await.unwrap();
        let item = items
            .iter()
            .find(|&i| i.title.contains("1 件目"))
            .expect("「1 件目」を含むアイテムは必ずあるはず");
        assert!(!item.done, "まだ、未完了のはずです。");
        let id = item.id;
        todo.change_done(id, sess, true)
            .await
            .expect("状態更新に失敗。あってはならない。");
        let items = todo.get_todo_list(sess, true).await.unwrap();
```

```
404         assert_eq!(
405             items.len(),
406             2,
407             "一件完了済みにしたので、このリストは 2 件しかない。"
408         );
409         let items = todo.get_todo_list(sess, false).await.unwrap();
410         assert_eq!(items.len(), 3, "完了済みを含むので、3 件になる。");
411         let item = items
412             .iter()
413             .find(|&i| i.id == id)
414             .expect("さっきあった id だから必ずある。");
415         assert!(item.done, "さっき完了済みに変更した。");
416 
417         let max_id = items.iter().max_by_key(|&x| x.id).unwrap().id;
418         let res = todo.change_done(max_id + 1, sess, false).await;
419         match res {
420             Ok(_) => unreachable!("この id の todo があるはずがない。"),
421             Err(TodoError::NotFoundTodo) => {}
422             Err(e) => unreachable!("このエラーもありえない。[{e}]"),
423         };
424 
425         // 間違ったセッションのテスト
426         let res = todo.change_done(id, Uuid::now_v7(), true).await;
427         match res {
428             Ok(_) => unreachable!("このセッションでは、更新を許してはいけない。"),
429             Err(TodoError::NotFoundTodo) => { /* 正常 */ }
430             Err(e) => unreachable!("このエラーもおかしい。[{e}]"),
431         }
432     }
433 
434     #[sqlx::test]
435     async fn edit_todo_test(pool: MySqlPool) {
436         let todo = Todo::test_new(pool);
437         let sess = login_for_test(&todo).await;
438         create_todo_for_test(&todo, sess).await;
439 
440         let items = todo.get_todo_list(sess, false).await.unwrap();
441         let mut item = items
442             .iter()
443             .find(|&i| i.title.contains("1 件目"))
444             .unwrap()
445             .clone();
446         item.title = "更新した一件目".to_string();
447         if let Err(e) = todo.edit_todo(&item, sess).await {
448             unreachable!("更新処理に失敗した。[{e}]");
```

```
449         }
450         let Some(item_new) = todo
451             .get_todo_list(sess, false)
452             .await
453             .unwrap()
454             .iter()
455             .find(|&i| i.title.contains("更新した一件目"))
456             .cloned()
457         else {
458             unreachable!("更新したレコードが見つからないよ?");
459         };
460         assert_eq!(item.id, item_new.id, "更新したレコードの id が化けてる");
461 
462         // ニセセッションで試す
463         match todo.edit_todo(&item, Uuid::now_v7()).await {
464             Ok(_) => unreachable!("偽のセッションで更新成功してはならない。"),
465             Err(TodoError::NotFoundTodo) => { /* 正常 */ }
466             Err(e) => unreachable!("偽セッションのときのエラー:{e}"),
467         }
468     }
469 
470     async fn login_for_test(todo: &Todo) -> Uuid {
471         let user_name = "testdayo";
472         let user_pass = "passrordnona";
473         todo.add_user(user_name, user_pass).await.unwrap();
474         todo.login(user_name, user_pass).await.unwrap()
475     }
476 
477     async fn create_todo_for_test(todo: &Todo, sess: Uuid) {
478         use chrono::Days;
479         let items = [
480             ItemTodo {
481                 id: 100,
482                 user_name: "kore_naihazu".to_string(),
483                 title: "テストアイテム 1 件目".to_string(),
484                 work: Some("これは、中身を入れる。".to_string()),
485                 update_date: None,
486                 start_date: Some(Local::now().date_naive() - Days::new(1)),
487                 end_date: Some(Local::now().date_naive() + Days::new(5)),
488                 done: false,
489             },
490             ItemTodo {
491                 id: 100,
492                 user_name: "kore_naihazu".to_string(),
493                 title: "テストアイテム 2 件目 (work=null)".to_string(),
```

```
                work: Some("".to_string()),
                update_date: None,
                start_date: Some(Local::now().date_naive() - Days::new(1)),
                end_date: Some(Local::now().date_naive() + Days::new(5)),
                done: false,
            },
            ItemTodo {
                id: 100,
                user_name: "kore_naihazu".to_string(),
                title: "テストアイテム３件目(work=space)".to_string(),
                work: Some(" \t  ".to_string()),
                update_date: None,
                start_date: Some(Local::now().date_naive() - Days::new(1)),
                end_date: Some(Local::now().date_naive() + Days::new(5)),
                done: false,
            },
        ];
        for item in items {
            todo.add_todo(sess, &item).await.unwrap();
        }
    }
}
```

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 chrono::{Local, NaiveDate};
use log::error;
use serde::{Deserialize, Serialize};
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 })
    }

    /// Todo 項目を追加する。
    /// item 引数のうち、id, update_date, done は、無視される
    /// 各々、自動値・今日の日付・false がはいる。
    /// start_date, end_date のデフォルト値は、今日・NaiveDate::MAX である。
    pub async fn add_todo_item(&self, item: &ItemTodo) -> Result<(), DbError> {
        let sql = r#"
            insert into todo(user_name, title, work, update_date, start_date, end_date, done)
            values (?, ?, ?, curdate(), ?, ?, false);
        "#;
        let start_date = item.start_date.unwrap_or(Local::now().date_naive());
        let end_date = item
```

```
            .end_date
            .unwrap_or(NaiveDate::from_ymd_opt(9999, 12, 31).unwrap());
        query(sql)
            .bind(&item.user_name)
            .bind(&item.title)
            .bind(&item.work)
            .bind(start_date)
            .bind(end_date)
            .execute(&self.pool)
            .await
            .map_err(DbError::FailDbAccess)?;
        Ok(())
    }

    /// Todo の一覧を取得する。
    /// 基準日 (ref_date) 以降のアイテムを選別する。
    /// セッション ID を必要とする。
    /// 検索オプションのとり方は未確定。インターフェース変更の可能性大。
    pub async fn get_todo_item(
        &self,
        sess: Uuid,
        ref_date: NaiveDate,
        only_incomplete: bool,
    ) -> Result<Vec<ItemTodo>, DbError> {
        let sql1 = r#"
            select t.id, t.user_name, title, work, update_date, start_date, end_date, done
            from todo t join sessions s on s.user_name = t.user_name
            where s.id=? and t.start_date <= ?
            "#;
        let sql2 = " and done = false";
        let sql = if only_incomplete {
            format!("{} {};", sql1, sql2)
        } else {
            format!("{} ;", sql1)
        };
        let items = query_as::<_, ItemTodo>(&sql)
            .bind(sess.to_string())
            .bind(ref_date)
            .fetch_all(&self.pool)
            .await
            .map_err(DbError::FailDbAccess)?;

        Ok(items)
    }

```

```
89      /// 指定 id の Todo 項目を取得する。
90      /// 有効なセッションが指定されていなければ、未発見とする。
91      pub async fn get_todo_item_with_id(&self, id: u32, sess: Uuid) -> Result<ItemTodo, DbError> {
92          let sql = r#"
93              select t.id, t.user_name, t.title, t.work, t.update_date, t.start_date, t.end_date,
                ↪  t.done
94              from todo t join sessions s on s.user_name = t.user_name
95              where s.id=? and t.id=?
96              "#;
97          query_as::<_, ItemTodo>(sql)
98              .bind(sess.to_string())
99              .bind(id)
100             .fetch_one(&self.pool)
101             .await
102             .map_err(|e| match e {
103                 sqlx::Error::RowNotFound => DbError::NotFoundTodo,
104                 e => DbError::FailDbAccess(e),
105             })
106     }
107
108     /// Todo の完了状態を更新する。
109     pub async fn change_done(&self, id: u32, done: bool) -> Result<(), DbError> {
110         let sql = "update todo set done = ? where id = ?";
111         let res = query(sql)
112             .bind(done)
113             .bind(id)
114             .execute(&self.pool)
115             .await
116             .map_err(DbError::FailDbAccess)?;
117         if res.rows_affected() > 0 {
118             Ok(())
119         } else {
120             Err(DbError::NotFoundTodo)
121         }
122     }
123
124     /// Todo の項目編集
125     pub async fn edit_todo(&self, item: &ItemTodo) -> Result<(), DbError> {
126         let start_date = item.start_date.unwrap_or(Local::now().date_naive());
127         let end_date = item
128             .end_date
129             .unwrap_or(NaiveDate::from_ymd_opt(9999, 12, 31).unwrap());
130
131         let sql = r#"
132             update todo
```

```
133             set title=?, work=?, update_date=curdate(), start_date=?, end_date=?
134             where id=?;
135             "#;
136         let res = query(sql)
137             .bind(&item.title)
138             .bind(&item.work)
139             .bind(start_date)
140             .bind(end_date)
141             .bind(item.id)
142             .execute(&self.pool)
143             .await
144             .map_err(DbError::FailDbAccess)?;
145         if res.rows_affected() > 0 {
146             Ok(())
147         } else {
148             Err(DbError::NotFoundTodo)
149         }
150     }
151 
152     /// ユーザーの追加
153     pub async fn add_user(&self, name: &str, pass: &str) -> Result<(), DbError> {
154         let sql = "insert into users(name, password) values (?, ?);";
155         query(sql)
156             .bind(name)
157             .bind(pass)
158             .execute(&self.pool)
159             .await
160             .map_err(|e| match e {
161                 sqlx::Error::Database(ref db_err) => {
162                     if db_err.kind() == sqlx::error::ErrorKind::UniqueViolation {
163                         DbError::DuplicateUserName(e)
164                     } else {
165                         DbError::FailDbAccess(e)
166                     }
167                 }
168                 _ => DbError::FailDbAccess(e),
169             })?;
170         Ok(())
171     }
172 
173     /// ユーザー名をキーとして、ユーザー情報を取得
174     pub async fn get_user(&self, name: &str) -> Result<User, DbError> {
175         let sql = "select name, password from users where name = ?;";
176         query_as(sql)
177             .bind(name)
```

```
178             .fetch_one(&self.pool)
179             .await
180             .map_err(|e| match e {
181                 sqlx::Error::RowNotFound => DbError::NotFoundUser,
182                 e => DbError::FailDbAccess(e),
183             })
184     }
185 
186     /// セッション ID をキーにしてユーザー情報を取得
187     pub async fn get_user_from_sess(&self, sess: Uuid) -> Result<User, DbError> {
188         let sql = r#"
189             select u.name, u.password
190             from users u join sessions s on u.name=s.user_name
191             where s.id = ?;
192             "#;
193 
194         query_as(sql)
195             .bind(sess.to_string())
196             .fetch_one(&self.pool)
197             .await
198             .map_err(|e| match e {
199                 sqlx::Error::RowNotFound => DbError::NotFoundSession,
200                 e => DbError::FailDbAccess(e),
201             })
202     }
203 
204     /// セッション情報を新規作成する。
205     ///   生成した uuid を返す。
206     pub async fn make_new_session(&self, user_name: &str) -> Result<Uuid, DbError> {
207         let sql = "insert into sessions(id, user_name) values (?,?);";
208         // キー情報の作成
209         let id = Uuid::now_v7();
210 
211         query(sql)
212             .bind(id.to_string())
213             .bind(user_name)
214             .execute(&self.pool)
215             .await
216             .map_err(|err| match err {
217                 sqlx::Error::Database(ref e) => {
218                     if e.is_foreign_key_violation() {
219                         // 外部キーエラー。存在しないユーザーを指定した。
220                         return DbError::NotFoundUser;
221                     }
222                     DbError::FailDbAccess(err)
```

```
223                 }
224                 _ => DbError::FailDbAccess(err),
225             })?;
226 
227         Ok(id)
228     }
229 
230     /// 指定されたセッションを新規セッションに更新する。
231     /// 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。
232     pub async fn update_session(&self, id: &uuid::Uuid) -> Result<Uuid, DbError> {
233         let mut tr = self.pool.begin().await.map_err(DbError::FailDbAccess)?;
234         // 期限切れのセッション削除
235         let sql_old_del = "delete from sessions where expired < now();";
236         query(sql_old_del)
237             .execute(&mut *tr)
238             .await
239             .map_err(DbError::FailDbAccess)?;
240 
241         // ユーザー ID の特定
242         let sql_query_user = "select user_name from sessions where id=?;";
243         let user: String = query(sql_query_user)
244             .bind(id.to_string())
245             .fetch_one(&mut *tr)
246             .await
247             .map_err(|e| match e {
248                 sqlx::Error::RowNotFound => DbError::NotFoundSession,
249                 e => DbError::FailDbAccess(e),
250             })?
251             .get("user_name");
252 
253         // 旧セッションの削除
254         let sql_del_curr_sess = "delete from sessions where id = ?;";
255         query(sql_del_curr_sess)
256             .bind(id.to_string())
257             .execute(&mut *tr)
258             .await
259             .map_err(DbError::FailDbAccess)?;
260 
261         // 新セッションの生成
262         let sql_create_sess = "insert into sessions(id, user_name) values (?, ?);";
263         let id = Uuid::now_v7();
264         query(sql_create_sess)
265             .bind(id.to_string())
266             .bind(user)
267             .execute(&mut *tr)
```

```
            .await
            .map_err(DbError::FailDbAccess)?;

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

    /// 指定されたセッション ID が有効であるか確認する。
    /// データベースエラーが発生した場合は、Err(DbError::FailDbAccess) を返す。
    pub async fn is_session_valid(&self, sess: &Uuid) -> Result<bool, DbError> {
        // 期限切れのセッションを削除する。
        let sql_old_del = "delete from sessions where expired < now();";
        query(sql_old_del)
            .execute(&self.pool)
            .await
            .map_err(DbError::FailDbAccess)?;
        // 指定セッション ID の有無を確認する。
        let sql_find_sess = "select count(*) as cnt from sessions where id = ?;";
        let sess_cnt: i64 = query(sql_find_sess)
            .bind(sess.to_string())
            .fetch_one(&self.pool)
            .await
            .map_err(DbError::FailDbAccess)?
            .get("cnt");
        if sess_cnt == 1 {
            Ok(true)
        } else {
            Ok(false)
        }
    }
}

#[derive(FromRow, Debug, PartialEq)]
pub struct User {
    pub name: String,
    pub password: String,
}

#[derive(FromRow, Serialize, Deserialize, Debug, PartialEq, Clone)]
pub struct ItemTodo {
    pub id: u32,
    pub user_name: String,
    pub title: String,
    pub work: Option<String>,
    pub update_date: Option<NaiveDate>,
```

```
313     pub start_date: Option<NaiveDate>,
314     pub end_date: Option<NaiveDate>,
315     pub done: bool,
316 }
317 
318 #[derive(Error, Debug)]
319 pub enum DbError {
320     #[error("データベースへの接続に失敗。")]
321     FailConnect(sqlx::Error),
322     #[error("データベース操作失敗（一般）")]
323     FailDbAccess(sqlx::Error),
324     #[error("User 挿入失敗（name 重複）")]
325     DuplicateUserName(sqlx::Error),
326     #[error("ユーザーが見つかりません。")]
327     NotFoundUser,
328     #[error("指定されたセッション id が見つかりません。")]
329     NotFoundSession,
330     #[error("指定された id の todo が見つかりません。")]
331     NotFoundTodo,
332 }
333 
334 #[cfg(test)]
335 mod test {
336     use chrono::Days;
337 
338     use super::*;
339 
340     /// テスト用の Database 生成。テスト用 Pool をインジェクション
341     impl Database {
342         pub(crate) fn new_test(pool: MySqlPool) -> Self {
343             Self { pool }
344         }
345     }
346 
347     /// ユーザー生成のテスト
348     #[sqlx::test]
349     async fn test_add_user_and_get_user(pool: MySqlPool) {
350         let db = Database::new_test(pool);
351         db.add_user("hyara", "password").await.unwrap();
352         let user = db.get_user("hyara").await.unwrap();
353         assert_eq!(user.name, "hyara");
354         assert_eq!(user.password, "password");
355         let error_user = db.get_user("naiyo").await;
356         match error_user {
357             Ok(_) => unreachable!("結果が帰ってくるはずがない。"),
```

```
358             Err(DbError::NotFoundUser) => { /* 正常 */ }
359             Err(e) => unreachable!("このエラーはおかしい。{e}"),
360         }
361     }
362
363     /// セッション生成関係の一連のテスト。
364     #[sqlx::test]
365     async fn test_make_new_session(pool: MySqlPool) {
366         println!("まずはテスト用のユーザーの生成");
367         let db = Database::new_test(pool);
368         let user_name = "nekodayo";
369         let password = "password";
370         db.add_user(user_name, password).await.unwrap();
371
372         println!("次に、普通にセッションを作ってみる。");
373         let sess1 = db.make_new_session(user_name).await.unwrap();
374         println!("セッション生成成功 id=[{}]", sess1);
375
376         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。");
377         let sess2 = db.make_new_session("detarame").await;
378         match sess2 {
379             Ok(_) => unreachable!("このユーザーは存在しなかったはず。"),
380             Err(DbError::NotFoundUser) => { /* 正常 */ }
381             Err(e) => unreachable!("このエラーもおかしい。[{}]", e),
382         }
383
384         println!("普通に、セッションを更新してみる。");
385         let sess3 = db.update_session(&sess1).await.unwrap();
386         assert_ne!(sess1, sess3);
387
388         println!("ないはずのセッションを更新しようとしてみる。");
389         let sess4 = Uuid::now_v7();
390         let sess5 = db.update_session(&sess4).await;
391         match sess5 {
392             Ok(_) => unreachable!("このセッションはないはずなのに。"),
393             Err(DbError::NotFoundSession) => { /* 正常 */ }
394             Err(e) => unreachable!("セッション更新２回め。失敗するにしてもこれはない{e}"),
395         }
396     }
397
398     /// セッションが有効かどうかを確認するテスト
399     #[sqlx::test]
400     async fn test_is_session_valid(pool: MySqlPool) {
401         let db = Database::new_test(pool);
402
```

```
403         println!("テスト用ユーザーの作成");
404         let name = "nekodayo";
405         let pass = "nekodamon";
406         db.add_user(name, pass).await.unwrap();
407 
408         println!("新規セッションを生成する。");
409         let sess = db.make_new_session(name).await.unwrap();
410         println!("生成したセッション ID は、[{}] です。", &sess);
411 
412         println!("今作ったセッション ID の妥当性を問い合わせてみる。");
413         assert!(db.is_session_valid(&sess).await.unwrap());
414 
415         println!("偽セッション ID をいれて、問い合わせてみる。");
416         assert!(!db.is_session_valid(&Uuid::now_v7()).await.unwrap());
417     }
418 
419     /// todo の書き込みと、単純な読み出しのテスト
420     #[sqlx::test]
421     async fn test_add_todo(pool: MySqlPool) {
422         let db = Database::new_test(pool);
423         let sess = login_for_test(&db).await;
424         let name = db.get_user_from_sess(sess).await.unwrap().name;
425 
426         println!("テストデータをインサート");
427         let mut item = ItemTodo {
428             id: 0,
429             user_name: name.to_string(),
430             title: "インサートできるかな?".to_string(),
431             work: Some("中身入り".to_string()),
432             update_date: None,
433             start_date: Some(Local::now().date_naive()),
434             end_date: Some(Local::now().date_naive() + Days::new(3)),
435             done: true,
436         };
437         db.add_todo_item(&item).await.unwrap();
438 
439         println!("テストデータを読み出す。一件しかないはず");
440         let last_day = Local::now().date_naive() + Days::new(1);
441         let res = db.get_todo_item(sess, last_day, true).await.unwrap();
442         assert_eq!(res.len(), 1, "あれ?一件のはずだよ");
443         item.id = res[0].id;
444         item.update_date = Some(Local::now().date_naive());
445         item.done = false;
446     }
447 
```

```rust
    /// todo の書き込みと読み出し。
    /// work が未入力の場合。
    #[sqlx::test]
    async fn test_add_todo_without_work(pool: MySqlPool) {
        let db = Database::new_test(pool);
        let sess = login_for_test(&db).await;
        let name = db.get_user_from_sess(sess).await.unwrap().name;

        println!("テストデータをインサート");
        let mut item = ItemTodo {
            id: 0,
            user_name: name.to_string(),
            title: "インサートできるかな?".to_string(),
            work: None,
            update_date: None,
            start_date: Some(Local::now().date_naive()),
            end_date: Some(Local::now().date_naive() + Days::new(3)),
            done: true,
        };
        db.add_todo_item(&item).await.unwrap();

        println!("テストデータを読み出す。一件しかないはず");
        let last_day = Local::now().date_naive() + Days::new(1);
        let res = db.get_todo_item(sess, last_day, true).await.unwrap();
        assert_eq!(res.len(), 1, "あれ?一件のはずだよ");
        item.id = res[0].id;
        item.update_date = Some(Local::now().date_naive());
        item.done = false;
    }

    /// todo の書き込みと読み出し
    /// done=true と false の挙動テスト
    #[sqlx::test]
    async fn test_get_todo_done_param(pool: MySqlPool) {
        let db = Database::new_test(pool.clone());
        let sess = login_for_test(&db).await;
        let name = db.get_user_from_sess(sess).await.unwrap().name;

        println!("テストデータをインサート");
        let item = ItemTodo {
            id: 0,
            user_name: name.to_string(),
            title: "インサートできるかな?".to_string(),
            work: None,
            update_date: None,
```

```
            start_date: Some(Local::now().date_naive()),
            end_date: Some(Local::now().date_naive() + Days::new(3)),
            done: true,
        };
        db.add_todo_item(&item).await.unwrap();

        println!("テストデータを読み出す。一件しかないはず");
        let last_day = Local::now().date_naive() + Days::new(1);
        let res = db.get_todo_item(sess, last_day, false).await.unwrap();
        assert_eq!(res.len(), 1, "全部読み出しだけど一件あるはず。");
        let res = db.get_todo_item(sess, last_day, true).await.unwrap();
        assert_eq!(res.len(), 1, "未完了だけだけど、一件あるはず。");

        println!("今作った job を完了済みにする。");
        let sql = "update todo set done=true where id=?;";
        query(sql).bind(res[0].id).execute(&pool).await.unwrap();
        let res = db.get_todo_item(sess, last_day, false).await.unwrap();
        assert_eq!(res.len(), 1, "全部読み出しだけど一件あるはず。");
        let res = db.get_todo_item(sess, last_day, true).await.unwrap();
        assert_eq!(res.len(), 0, "未完了だけだけだから、なにもないはず。");
    }

    /// todo の書き込みと読み出し
    /// 基準日の挙動テスト
    #[sqlx::test]
    async fn test_get_todo_ref_date(pool: MySqlPool) {
        let db = Database::new_test(pool.clone());
        let sess = login_for_test(&db).await;
        let name = db.get_user_from_sess(sess).await.unwrap().name;

        println!("テストデータをインサート");
        let item = ItemTodo {
            id: 0,
            user_name: name.to_string(),
            title: "インサートできるかな?".to_string(),
            work: None,
            update_date: None,
            start_date: Some(Local::now().date_naive()),
            end_date: Some(Local::now().date_naive() + Days::new(3)),
            done: false,
        };
        db.add_todo_item(&item).await.unwrap();

        let ref_date = Local::now().date_naive();
        let res = db.get_todo_item(sess, ref_date, true).await.unwrap();
```

```
538         assert_eq!(res.len(), 1, "基準日と開始日が同じだからみつかる。");
539         let res = db
540             .get_todo_item(sess, ref_date + Days::new(1), true)
541             .await
542             .unwrap();
543         assert_eq!(res.len(), 1, "開始日の翌日が基準日だからみつかる。");
544         let res = db
545             .get_todo_item(sess, ref_date - Days::new(1), true)
546             .await
547             .unwrap();
548         assert_eq!(res.len(), 0, "基準日が開始日の前日だからみつからない。");
549         let res = db
550             .get_todo_item(sess, ref_date + Days::new(4), true)
551             .await
552             .unwrap();
553         assert_eq!(res.len(), 1, "基準日が期限を過ぎているけどみつかるの。");
554     }
555
556     #[sqlx::test]
557     async fn test_get_user_from_sess(pool: MySqlPool) {
558         let db = Database::new_test(pool.clone());
559
560         let sess = login_for_test(&db).await;
561         let name = db.get_user_from_sess(sess).await.unwrap().name;
562
563         let user = db.get_user_from_sess(sess).await.unwrap();
564         assert_eq!(user.name, name, "これはみつかるはず");
565         let dummy_sess = Uuid::now_v7();
566         let user = db.get_user_from_sess(dummy_sess).await;
567         match user {
568             Ok(_) => unreachable!("見つかるわけないでしょう。"),
569             Err(DbError::NotFoundSession) => { /* 正常 */ }
570             Err(e) => unreachable!("トラブルです。{e}"),
571         };
572     }
573
574     #[sqlx::test]
575     async fn test_change_done(pool: MySqlPool) {
576         let db = Database::new_test(pool);
577         let sess = login_for_test(&db).await;
578         let ref_date = Local::now().date_naive();
579         create_todo_for_test(&db, sess).await;
580
581         let items = db.get_todo_item(sess, ref_date, true).await.unwrap();
582         let item = items.iter().find(|&i| i.title.contains("二件目")).unwrap();
```

```
583         db.change_done(item.id, true).await.unwrap();
584
585         let items = db.get_todo_item(sess, ref_date, true).await.unwrap();
586         let item = items.iter().find(|&i| i.title.contains("二件目"));
587         assert!(item.is_none(), "状態を完了にしたので見つからないはず。");
588
589         let items = db.get_todo_item(sess, ref_date, false).await.unwrap();
590         let item = items.iter().find(|&i| i.title.contains("二件目"));
591         match item {
592             Some(i) => assert!(i.done, "完了済みになっているはずですね?"),
593             None => unreachable!("状態を変えたら、レコードなくなった???"),
594         }
595         assert_eq!(
596             items.len(),
597             3,
598             "全件見ているのでレコードは 3 件あるはずですが?"
599         );
600     }
601
602     #[sqlx::test]
603     async fn test_get_todo_with_id(pool: MySqlPool) {
604         let db = Database::new_test(pool);
605         let sess = login_for_test(&db).await;
606         create_todo_for_test(&db, sess).await;
607
608         let items = db
609             .get_todo_item(sess, Local::now().date_naive(), false)
610             .await
611             .unwrap();
612         let id = items
613             .iter()
614             .find(|&i| i.title.contains("一件目"))
615             .expect("これはあるはず")
616             .id;
617         let non_exist_id = items.iter().max_by_key(|&i| i.id).unwrap().id + 1;
618
619         // 正常な読み出し
620         let res = db
621             .get_todo_item_with_id(id, sess)
622             .await
623             .expect("これは正常に読み出せるはず。エラーはだめ");
624         res.work
625             .expect("このレーコードは work を持つはずです。")
626             .find("働いてます。")
627             .expect("work の内容がおかしい。");
```

```
628
629         // 間違った id
630         let res = db.get_todo_item_with_id(non_exist_id, sess).await;
631         match res {
632             Ok(_) => unreachable!("そんな ID は存在しなかったはずなのに。"),
633             Err(DbError::NotFoundTodo) => { /* 正常 */ }
634             Err(e) => unreachable!("データベースエラーだよ。({e})"),
635         }
636
637         // 間違ったセッション
638         let res = db.get_todo_item_with_id(id, Uuid::now_v7()).await;
639         match res {
640             Ok(_) => unreachable!("そんなセッションはないはず。"),
641             Err(DbError::NotFoundTodo) => { /* 正常 */ }
642             Err(e) => unreachable!("データベースエラー発生。({e})"),
643         }
644     }
645
646     #[sqlx::test]
647     async fn test_edit(pool: MySqlPool) {
648         let db = Database::new_test(pool);
649         let sess = login_for_test(&db).await;
650         create_todo_for_test(&db, sess).await;
651
652         // 書き込みテスト用レコードの取得
653         let today = Local::now().date_naive();
654         let items = db.get_todo_item(sess, today, false).await.unwrap();
655         let mut item = items
656             .iter()
657             .find(|&i| i.title.contains("一件目"))
658             .expect("ないはずがない。")
659             .clone();
660         item.title = "更新しました。".to_string();
661         item.work = Some("書き換え後".to_string());
662         item.start_date = Some(today - Days::new(5));
663         item.end_date = Some(today + Days::new(10));
664         db.edit_todo(&item).await.expect("更新がエラーを起こした。");
665         // 書き込み後の照合
666         let items_new = db.get_todo_item(sess, today, false).await.unwrap();
667         let item_new = items_new
668             .iter()
669             .find(|&i| i.title.contains("更新しました。"))
670             .expect("更新されたレコードが存在しない。");
671         assert_eq!(
672             item_new.work,
```

```
673             Some("書き換え後".to_string()),
674             "更新後の work がおかしい"
675         );
676         assert_eq!(
677             item_new.start_date,
678             Some(today - Days::new(5)),
679             "更新後の start_date がおかしい"
680         );
681         assert_eq!(
682             item_new.end_date,
683             Some(today + Days::new(10)),
684             "更新後の end_date がおかしい"
685         );
686 
687         // 存在しないレコードの更新
688         let id_max_plus_one = items.iter().max_by_key(|&i| i.id).unwrap().id + 1;
689         item.id = id_max_plus_one;
690         let res = db.edit_todo(&item).await;
691         match res {
692             Ok(_) => unreachable!("更新できちゃだめっ"),
693             Err(DbError::NotFoundTodo) => {}
694             Err(e) => unreachable!("db_err: {e}"),
695         }
696     }
697     async fn login_for_test(db: &Database) -> Uuid {
698         println!("テスト用ユーザー及びセッションの生成");
699         let name = "test";
700         let pass = "test";
701         db.add_user(name, pass).await.unwrap();
702         db.make_new_session(name).await.unwrap()
703     }
704 
705     async fn create_todo_for_test(db: &Database, sess: Uuid) {
706         let name = db.get_user_from_sess(sess).await.unwrap().name;
707 
708         println!("テストデータをインサート");
709         let item = ItemTodo {
710             id: 0,
711             user_name: name.to_string(),
712             title: "一件目（work 有り）".to_string(),
713             work: Some("働いてます。".to_string()),
714             update_date: None,
715             start_date: Some(Local::now().date_naive()),
716             end_date: Some(Local::now().date_naive() + Days::new(3)),
717             done: false,
```

```
        };
        db.add_todo_item(&item).await.unwrap();

        let item = ItemTodo {
            id: 0,
            user_name: name.to_string(),
            title: "二件目（work無し）".to_string(),
            work: None,
            update_date: None,
            start_date: Some(Local::now().date_naive()),
            end_date: Some(Local::now().date_naive() + Days::new(3)),
            done: false,
        };
        db.add_todo_item(&item).await.unwrap();

        let item = ItemTodo {
            id: 0,
            user_name: name.to_string(),
            title: "三件目（work無し）".to_string(),
            work: None,
            update_date: None,
            start_date: Some(Local::now().date_naive()),
            end_date: Some(Local::now().date_naive() + Days::new(3)),
            done: false,
        };
        db.add_todo_item(&item).await.unwrap();
    }
}
```

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 } from "@yamada-ui/react";
import App from "./App";
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
import { theme } from "./theme";

const query_client = new QueryClient();

ReactDOM.createRoot(document.getElementById("root")).render(
    <React.StrictMode>
        <UIProvider theme={theme}>
            <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 AddTodo from "./AddTodo.jsx";
import Login from "./Login.jsx";
import RegistUser from "./RegistUser.jsx";
import Init from "./Init.jsx";
import EditTodo from "./EditTodo";

export const routes = createBrowserRouter(
    createRoutesFromElements(
        <>
            <Route element={ <BasePage/> }>
                <Route path="/" element={<Init/>}/>
                <Route path="/login" element={<Login/>}/>
                <Route path="/regist_user" element={<RegistUser/>}/>
                <Route path="/todo" element={<TodoList/>}/>
                <Route path="/addtodo" element={<AddTodo/>}/>
                <Route path="/edittodo/:id" element={<EditTodo/>}/>
            </Route>
        </>
    ));

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

2.5 全体のベースページ BasePage.jsx

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


function BasePage() {

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


export default BasePage;
```

2.6 アプリケーションの初期化 Init.jsx

```jsx
/* アプリケーションの初期化 */
/* 有効なセッションがあれば、ログイン済みに */
/* でなければ、ログイン画面へ遷移 */

import { Container, Heading} from "@yamada-ui/react";
import { invoke } from "@tauri-apps/api/core";
import { useNavigate } from "react-router-dom";
import {useQuery} from "@tanstack/react-query";
import { useEffect } from "react";

function Init() {

    const navi = useNavigate();

    const { data, isFetching, isSuccess, isError, error } = useQuery({
        queryKey: ['check_login'],
        queryFn: async () => invoke('is_valid_session')
        });

    useEffect( () => {
        if (isSuccess && !isFetching) {
            if (data === true) {
                navi('/todo');
            } else {
                navi('/login');
            }
        }
    },[isSuccess, isFetching])

    return (
        <>
            <Container centerContent>
                <Heading> ただいま、初期化中です。</Heading>
                <p> しばらくお待ちください。</p>
                <p> 現在、ログイン状態の検査中です。</p>
                <p> { isError && "error 発生:"+error }</p>
            </Container>
        </>
    );

}

export default Init;
```

2.7 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
```

2.8 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, useNavigate } from "react-router-dom";
import { useState } from "react";
import {useQueryClient} from "@tanstack/react-query";

function Login() {
    const { register, handleSubmit, formState: {errors} } = useForm();
    const [ sendMessage, setSendMessage ] = useState('');
    const navi = useNavigate();
    const queryClient = useQueryClient();

    const onSubmit = async (data) => {
        try {
            setSendMessage('処理中です。');
            await invoke('login', { name: data.name, password: data.pass });
            queryClient.invalidateQueries("check_login");
            navi('/');
        } 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>
            </VStack>
        </>
    );
}

export default Login;

```

2.9 todo リストの表示 TodoList.jsx

```
import { useNavigate } from "react-router-dom";
import { useQuery } from "@tanstack/react-query";
import { Grid, GridItem, HStack, IconButton} from "@yamada-ui/react";
import { invoke } from "@tauri-apps/api/core";
import { AiOutlineFileAdd } from "react-icons/ai";
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: ['todo_list'],
        queryFn: get_todo_list,
    });

    const navi = useNavigate();
    const handleAddTodo = () => navi('/addtodo');

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

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

    console.log(todos);
    return (
        <>
            <HStack>
                <IconButton icon={<AiOutlineFileAdd/>} onClick={handleAddTodo}/>
            </HStack>

            <h1>現在の予定</h1>
            <Grid templateColumns="repeat(4, 1fr)" gap="md">
                {todos?.map( todo_item => {
                    return (
                        <GridItem key={todo_item.id} w="full" rounded="md" bg="primary">
                            <TodoItem item={todo_item}/>
                        </GridItem>
                    )}
```

```
                )}
            </Grid>
        </>
    );
}


export default TodoList;
```

2.10 todo アイテム表示 todoitem.jsx

```
// todo リストの各アイテム
import {useNavigate} from "react-router-dom";
import {useMutation, useQueryClient} from "@tanstack/react-query";
import {invoke} from "@tauri-apps/api/core";
import { SimpleGrid, GridItem, IconButton, Text, HStack } from "@yamada-ui/react";
import { BsWrenchAdjustable } from "react-icons/bs";
import { BsAlarm } from "react-icons/bs";
import { BsEmojiGrin } from "react-icons/bs";
import { BiPencil } from "react-icons/bi";

export default function TodoItem({item}) {
    const navi = useNavigate();
    const queyrClient = useQueryClient();
    const {mutate} = useMutation({
        mutationFn: () => {
            return invoke("update_done", {id: item.id, done: !item.done})
        },
        onSuccess: () => {
            queyrClient.invalidateQueries({ queryKey: ["todo_list"]});
        }
    });

    const onEditClick = () => {
        navi("/edittodo/"+item.id);
    }

    const onDoneClick = () => {
        console.log(item.id + " : " + item.title);
        mutate();
    }

    // 日付の表示内容生成
    let end_date = new Date(item.end_date);
    if (item.end_date === "9999-12-31") {
        end_date = null;
    }
    const start_date = new Date(item.start_date);
    const update_date = new Date(item.update_date);

    // 完了ボタンのアイコン選択
    let done_icon;
    if (item.done) {
        done_icon = <BsEmojiGrin/>;
```

```
    } else if (!!end_date && geDate(new Date(), end_date)) {
        done_icon = <BsAlarm/>;
    } else {
        done_icon = <BsWrenchAdjustable/>
    }

    return (
        <>
            <SimpleGrid w="full" columns={{base: 2, md: 1}} gap="md">
                <GridItem>
                    <HStack>
                        <IconButton size="xs" icon={done_icon} onClick={onDoneClick}/>
                        <IconButton size="xs" icon={<BiPencil/>} onClick={onEditClick}/>
                    </HStack>
                </GridItem>

                <GridItem>
                    <Text fontSize="xs" align="right">
                        {update_date?.toLocaleDateString()}
                    </Text>
                </GridItem>
            </SimpleGrid>
            <Text align="center" fontSize="lg" as="b">
                {item.title}
            </Text>
            <Text fontSize="sm">
                {item.work}
            </Text>
            <Text fontSize="sm">
                {start_date?.toLocaleDateString()} ~ {end_date?.toLocaleDateString()}
            </Text>
        </>
    );
}

function geDate(val1, val2) {
    const year1 = val1.getFullYear();
    const month1 = val1.getMonth();
    const day1 = val1.getDate();
    const year2 = val2.getFullYear();
    const month2 = val2.getMonth();
    const day2 = val2.getDate();

    if (year1 === year2) {
        if (month1 === month2) {
```

```
89              return day1 >= day2;
90          } else {
91              return month1 > month2;
92          }
93      } else {
94          return year1 > year2;
95      }
96  }
97
```

2.11 todo アイテムの追加 AddTodo.jsx

```
import { invoke } from "@tauri-apps/api/core";
import { str2date } from "./str2date.jsx";
import { InputTodo } from "./InputTodo.jsx";

function AddTodo() {

    const send_data = async (data) => {
        const res = {item : {
            title : data.title,
            work : data.work,
            start : str2date(data.start)?.toLocaleDateString(),
            end : str2date(data.end)?.toLocaleDateString(),
        }};
        await invoke('add_todo', res);
    };

    const init_val = {
        title : "",
        work : "",
        start : "",
        end : "",
    };

    return (
        <>
            <InputTodo send_data={send_data} init_val={init_val}/>
        </>
    );
}

export default AddTodo;
```

2.12 todo アイテムの編集 EditTodo.jsx

```
import {useQuery} from "@tanstack/react-query";
import {useParams} from "react-router-dom";
import {invoke} from "@tauri-apps/api/core";

import {InputTodo} from "./InputTodo.jsx";
import { str2date } from "./str2date.jsx";

export default function EditTodo() {
    const { id } = useParams();

    const { data: todo, isLoading, isError, error} = useQuery({
        queryKey: ['todo_item_'+id],
        queryFn: async () => invoke('get_todo_with_id', {id: Number(id)}),
    });

    const handleSendData = async (data) => {
        const res = {
            id: Number(id),
            item: {
                title: data.title,
                work: data.work,
                start: str2date(data.start)?.toLocaleDateString(),
                end: str2date(data.end)?.toLocaleDateString(),
            }
        };
        await invoke("edit_todo", res);
    };

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

    if (isError) {
        return ( <p> Error: {error} </p> );
    }

    const initForm = {
        title: todo.title,
        work: todo.work,
        start: todo.start_date?.replace(/-/g,"/"),
        end: todo.end_date==="9999-12-31" ? "" : todo.end_date.replace(/-/g,"/"),
    }

```

```
44      return (
45          <>
46              <p> 工事中 id: {id} </p>
47              <InputTodo send_data={handleSendData} init_val={initForm}/>
48          </>
49      );
50  }
```

2.13 todo アイテム内容の入力フォーム InputTodo.jsx

```
import { FormProvider, useForm, useFormContext } from "react-hook-form";
import { Button, FormControl, HStack, Input, Text, Textarea, VStack } from "@yamada-ui/react";
import {useEffect, useState} from "react";
import { useNavigate } from "react-router-dom";
import { useMutation } from "@tanstack/react-query";
import { str2date } from "./str2date.jsx";

export function InputTodo({send_data, init_val}) {
    const form = useForm({
        defaultValues: {
            title: init_val.title,
            work: init_val.work,
            start: init_val.start,
            end: init_val.end
        },
    });
    const { register, handleSubmit, formState: {errors} } = form;
    const [ errorMessage, setErrorMessage ] = useState("");
    const navi = useNavigate();

    const {mutate, isPending} = useMutation( {
        mutationFn: (data) => send_data(data),
        onSuccess: () => navi('/todo'),
        onError: (error) => setErrorMessage(error),
    });

    return (
        <>
            <FormProvider {...form}>
                <VStack as="form" onSubmit={handleSubmit((data)=>mutate(data))}>
                    <FormControl
                        invalid={!!errors.title}
                        label="タイトル"
                        errorMessage={errors?.title?.message}
                    >
                        <Input placeholder="やること"
                            {...register("title", {required:"入力は必須です。"})}/>
                    </FormControl>
                    <FormControl label="詳細">
                        <Textarea {...register("work")} />
                    </FormControl>
                    <InputDate name="start" label="開始"/>
                    <InputDate name="end" label="終了"/>
```

```
                    <Button type="submit" w="30%" ml="auto" mr="auto">送信</Button>
                    <Text> {isPending ? "送信中です。" : null} </Text>
                    <Text> {errorMessage} </Text>
                </VStack>
            </FormProvider>
        </>
    );
}


function InputDate({name, label}) {
    const { register, watch, formState: {errors} } = useFormContext();

    const val = watch(name);
    const [ date, setDate,  ] = useState(null);
    useEffect(() => {
        setDate(str2date(val));
    }, [val]);

    return (
        <>
            <FormControl
                invalid = {!!errors[name]}
                label={label}
                errorMessage={errors[name]?.message}>
                <HStack>
                    <Input
                        w="50%"
                        placeholder="[[YYYY/]MM/]DD or +dd"
                        {...register(name, {
                            validate: (data) => {
                            if (data==null) { return }
                            if (data.length===0) { return }
                            if (str2date(data)==null) { return "日付の形式が不正です。" }
                            }
                        })}
                    />
                    <Text> {date?.toLocaleDateString()} </Text>
                </HStack>
            </FormControl>
        </>
    );
}

```

2.14 todo アイテム処理のための日付処理ユーティリティー str2date.jsx

```
// 日付処理ユーティリティ

export function str2date(str) {
    if (str == null) { return null; }
    if (str.length === 0) { return null; }
    const date_item = str.split('/');
    for (const s of date_item) {
        if (Number.isNaN(Number(s))) { return null; }
    }

    const cur_date = new Date();
    const cur_year = cur_date.getFullYear();

    let ret_date = cur_date;
    try {
        switch (date_item.length) {
            case 0:
                return null;
            case 1:
                if (date_item[0][0] == '+') {
                    ret_date.setDate(ret_date.getDate() + Number(date_item[0]));
                } else {
                    ret_date.setDate(Number(date_item[0]));
                    if (ret_date < new Date()) {
                        ret_date.setMonth(ret_date.getMonth() + 1);
                    }
                }

                break;
            case 2:
                ret_date = new Date(cur_year, Number(date_item[0])-1, Number(date_item[1]))
                if (ret_date < new Date()) {
                    ret_date.setFullYear(ret_date.getFullYear() + 1);
                }
                break;
            case 3:
                const year = Number(date_item[0]);
                const month = Number(date_item[1]);
                const date = Number(date_item[2]);
                ret_date = new Date(year, month-1, date);
                break;
            default:
                return null;
```

```
44          }
45      } catch(e) {
46          return null;
47      }
48      if (Number.isNaN(ret_date.getTime())) { return null; }
49
50      return ret_date;
51  }
```

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 not null,
    end_date date not null,
    done bool not null
    ) ;

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)
    );

```