neko_todo ソースリスト

美都

2025 年 2 月 23 日

目次

1 Rust ソース 3
1.1 メインモジュール　main.rs 3
1.2 アプリケーションステータス app_status.rs 6
1.3 コンフィグ設定処理 config.rs 7
1.4 アプリケーション設定情報の処理 setup.rs 14
1.5 todo モデル処理 todo.rs 17
1.5.1 新規作成 new.rs 18
1.5.2 todo データの取得　get_todo.rs 19
1.5.3 todo データの編集 edit_todo.rs 20
1.5.4 ユーザーデータの管理 use.rs 22
1.5.5 アプリケーション状態の管理 app_state.rs 23
1.5.6 Todo モジュールのテスト test.rs 25
1.6 データベースアクセスモジュール database.rs 32
1.6.1 新規作成 new.rs 34
1.6.2 セッション管理 session.rs 35
1.6.3 todo アイテム管理 todo.rs 38
1.6.4 ユーザーデータ管理 user.rs 41
1.6.5 データベースモジュールテスト test.rs 43
1.7 command モジュール tauri::command 関数群 command.rs 56
1.7.1 todo リストの表示・編集　todo.rs 57
1.7.2 ユーザー操作関係 user.rs 61
1.7.3 セッション操作関連 session.rs 62
1.7.4 アプリケーションの状態操作　app_state.rs 64
2 フロントエンド　React 関係 66
2.1 index.html 66
2.2 メイン CSS ファイル 67
2.3 main.jsx 70
2.4 アプリケーションメイン App.jsx 71
2.5 全体のベースページ BasePage.jsx 72
2.6 アプリケーションの初期化 Init.jsx 73
2.7 ユーザー登録画面 RegistUser.jsx 74
2.8 ログイン画面 Login.jsx 76
2.9 todo リストの表示 TodoList.jsx 78

2.10 todo アイテム表示 TodoItem.jsx ・・・ 80
2.11 todo リスト画面　ツールバー TodoListToolbar.jsx ・・・ 83
2.12 todo アイテムの追加 AddTodo.jsx ・・・ 85
2.13 todo アイテムの編集 EditTodo.jsx ・・・ 86
2.14 todo アイテムの複製 PasteTodo.jsx ・・・ 88
2.15 todo アイテム内容の入力フォーム InputTodo.jsx ・・・ 90
2.16 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 ・・・ 93
3 データベース構成 ・・・ 95
3.1 テーブル生成スクリプト create_table.sql ・・・ 95

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")]

mod app_status;
mod command;
mod config;
mod database;
mod setup;
mod todo;

use app_status::AppStatus;
use command::app_state::{
    get_is_incomplete, get_item_sort_order, set_is_incomplete, set_item_sort_order,
};
use command::session::is_valid_session;
use command::todo::{add_todo, edit_todo, get_todo_list, get_todo_with_id, update_done};
use command::user::{login, regist_user};
use directories::ProjectDirs;
use log::{error, info};
use setup::setup;
use tauri::Manager;

fn main() {
    setup_log();
    run()
}

/// ロギング機構のセットアップ
fn setup_log() {
    let mut log_file: std::path::PathBuf = ProjectDirs::from("jp", "laki", "nekotodo")
        .unwrap()
        .config_dir()
        .into();
    if !log_file.exists() {
        std::fs::create_dir_all(&log_file).unwrap();
    }
    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();
}

/// アプリケーション本体部分
#[cfg_attr(mobile, tauri::mobile_entry_point)]
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![
            get_todo_list,
            get_todo_with_id,
            regist_user,
            login,
            is_valid_session,
            add_todo,
            update_done,
            edit_todo,
            set_is_incomplete,
            get_is_incomplete,
            set_item_sort_order,
            get_item_sort_order,
        ])
        .build(tauri::generate_context!())
        .expect("error thile build tauri application");
```

```
88      app.run(|app, event| {
89          if let tauri::RunEvent::Exit = event {
90              info!("終了処理開始");
91              let state = app.state::<AppStatus>();
92              state.config().lock().unwrap().save().unwrap();
93          }
94      });
95  }
```

1.2 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
    }
}
```

1.3 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";

/// アプリケーション全体の状態設定
#[derive(Debug)]
pub struct NekoTodoConfig {
    db_host: String,
    db_user: String,
    db_pass: String,
    session_id: Option<Uuid>,
    dirty: bool,
    is_incomplete: bool,
    item_sort_order: ItemSortOrder,
}

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,
            is_incomplete: true,
            item_sort_order: ItemSortOrder::EndAsc,
```

```
        })
    }

    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 get_is_incomplete(&self) -> bool {
        self.is_incomplete
    }

    pub fn get_item_sort_order(&self) -> ItemSortOrder {
        self.item_sort_order
    }

    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;
    }
```

```rust

    pub fn set_is_incomplete(&mut self, is_incomplete: bool) {
        self.is_incomplete = is_incomplete;
    }

    pub fn set_item_sort_order(&mut self, item_sort_order: ItemSortOrder) {
        self.item_sort_order = item_sort_order;
    }

    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())?;
        writeln!(buffer, "{}={}", DB_PASS, self.get_db_pass())?;
        if let Some(s) = self.session_id {
            writeln!(buffer, "{}={}", SESSION, s)?;
        }
        self.dirty = false;
        Ok(())
    }

    /// コンフィグファイルのファイル名を生成する
    /// 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
    /// 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。
    fn get_config_file_path() -> Result<PathBuf> {
        use std::io;
        // 環境依存コンフィグ用ディレクトリの取得
        // 必要であれば、自分用のディレクトリを生成する。
        // ここでエラーになるのは、OSシステムに問題がある。
        let mut path: PathBuf = ProjectDirs::from("jp", "laki", "nekotodo")
            .ok_or(io::Error::new(ErrorKind::Other, "Not Found Home"))?
            .config_dir()
            .into();
        if let Err(e) = std::fs::create_dir(&path) {
            if e.kind() != ErrorKind::AlreadyExists {
                return Err(e);
            }
        }

        // 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。
        path.push(CONF_FILE_NAME);
```

```
137         if let Err(e) = std::fs::File::create_new(&path) {
138             if e.kind() != ErrorKind::AlreadyExists {
139                 return Err(e);
140             }
141         }
142         Ok(path)
143     }
144 }
145
146 impl Drop for NekoTodoConfig {
147     fn drop(&mut self) {
148         if self.dirty {
149             self.save().unwrap();
150         }
151     }
152 }
153
154 /// アイテムリストのソート順位を表す。
155 #[derive(Debug, Clone, Copy)]
156 pub enum ItemSortOrder {
157     StartAsc,
158     StartDesc,
159     EndAsc,
160     EndDesc,
161     UpdateAsc,
162     UpdateDesc,
163 }
164
165 impl std::fmt::Display for ItemSortOrder {
166     fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
167         match self {
168             Self::StartAsc => write!(f, "StartAsc"),
169             Self::StartDesc => write!(f, "StartDesc"),
170             Self::EndAsc => write!(f, "EndAsc"),
171             Self::EndDesc => write!(f, "EndDesc"),
172             Self::UpdateAsc => write!(f, "UpdateAsc"),
173             Self::UpdateDesc => write!(f, "UpdateDesc"),
174         }
175     }
176 }
177
178 impl std::str::FromStr for ItemSortOrder {
179     type Err = ItemSortOrderParseError;
180
181     fn from_str(s: &str) -> std::result::Result<Self, Self::Err> {
182         match s {
```

```
            "StartAsc" => Ok(Self::StartAsc),
            "StartDesc" => Ok(Self::StartDesc),
            "EndAsc" => Ok(Self::EndAsc),
            "EndDesc" => Ok(Self::EndDesc),
            "UpdateAsc" => Ok(Self::UpdateAsc),
            "UpdateDesc" => Ok(Self::UpdateDesc),
            _ => Err(ItemSortOrderParseError::InvalidArgument),
        }
    }
}

#[derive(thiserror::Error, Debug)]
pub enum ItemSortOrderParseError {
    #[error("Invalid Argument")]
    InvalidArgument,
}

#[cfg(test)]
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();
```

```
229         assert_eq!(conf.get_db_host(), val_db_host);
230         assert_eq!(conf.get_db_user(), val_db_user);
231         assert_eq!(conf.get_db_pass(), val_db_pass);
232         restore_curr_conf_file();
233     }
234 
235     /// テスト環境のため、元の conf ファイルを退避
236     fn save_curr_conf_file() {
237         let file = NekoTodoConfig::get_config_file_path().unwrap();
238         let mut save_file = file.clone();
239         save_file.set_extension("save");
240         if file.exists() {
241             println!(
242                 "現在の環境ファイル [{:?}] を [{:?}] に退避します。",
243                 &file, &save_file
244             );
245             std::fs::rename(file, save_file).unwrap();
246         }
247     }
248 
249     /// 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
250     fn restore_curr_conf_file() {
251         let file = NekoTodoConfig::get_config_file_path().unwrap();
252         let mut save_file = file.clone();
253         save_file.set_extension("save");
254         if save_file.exists() {
255             if file.exists() {
256                 println!("テスト用環境ファイル{:?}を削除します。", &file);
257                 std::fs::remove_file(&file).unwrap();
258             }
259             println!(
260                 "元の環境ファイル [{:?}] を [{:?}] に復元します。",
261                 &save_file, &file
262             );
263             std::fs::rename(save_file, file).unwrap();
264         }
265     }
266 
267     /// テスト環境のため、環境変数をすべて消去する。
268     fn delete_env_val() {
269         unsafe {
270             std::env::remove_var(DB_HOST);
271             std::env::remove_var(DB_USER);
272             std::env::remove_var(DB_USER);
273         }
274     }
```

```
275 }
```

1.4 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);
    };
```

```
45      let Some(ref pass) = args.pass else {
46          return Err(SetupError::Argument);
47      };
48
49      // 一度試しに接続してみる。
50      info!("次のパラメータを使用します。");
51      info!("ホスト名:{}", host);
52      info!("ユーザー名:{}", user);
53      info!("パスワード:{}", pass);
54      info!("データベースへの接続を試行します。");
55      block_on(async { Todo::new(host, user, pass).await })?;
56
57      info!("データベースへの接続に成功しました。");
58      info!("設定ファイルに接続情報を保存します。");
59      {
60          let mut conf = match NekoTodoConfig::new() {
61              Ok(c) => Ok(c),
62              Err(e) => Err(SetupError::SetupFile(e)),
63          }?;
64
65          conf.set_db_host(host);
66          conf.set_db_user(user);
67          conf.set_db_pass(pass);
68      }
69      eprintln!("アプリケーションを終了します。");
70      exit(0);
71  }
72
73  /// アプリケーション引数の定義
74  #[derive(Parser, Debug)]
75  #[command(version, about)]
76  struct Args {
77      /// データベース接続情報のセットアップを行う。
78      #[arg(long)]
79      setup: bool,
80      /// データベースのサーバー名
81      #[arg(short, long)]
82      server: Option<String>,
83      /// データベースのユーザー名
84      #[arg(short, long)]
85      user: Option<String>,
86      /// データベースのパスワード
87      #[arg(short, long)]
88      pass: Option<String>,
89  }
90
```

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

1.5 todo モデル処理 todo.rs

```rust
//! Todo アプリのビジネスロジック実装
mod app_state;
mod edit_todo;
mod get_todo;
mod new;
#[cfg(test)]
mod test;
mod user;

use crate::database::*;
use log::error;
use thiserror::Error;

/// todo アプリのビジネスロジック実装
pub struct Todo {
    database: Database,
}

#[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()
    }
}
```

1.5.1 新規作成 new.rs

```rust
//! todo 構造体新規作成
use super::*;

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 })
    }
}
```

1.5.2 todo データの取得　get_todo.rs

```rust
//! todo データの取得

use super::*;
use crate::{config::ItemSortOrder, database::*};
use chrono::Local;
use log::error;
use uuid::Uuid;

impl Todo {
    /// 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,
        sort_order: ItemSortOrder,
    ) -> Result<Vec<ItemTodo>, TodoError> {
        let ref_date = Local::now().date_naive();
        self.database
            .get_todo_item(sess, ref_date, only_imcomplete, sort_order)
            .await
            .map_err(|e| match e {
                DbError::FailDbAccess(e) => TodoError::FailDbAccess(e),
                e => unreachable!("[get_todo_list]get_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}]"),
            })
    }
}
```

1.5.3 todo データの編集 edit_todo.rs

```rust
//! todo データの編集

use super::*;
use crate::database::*;
use log::error;
use uuid::Uuid;

impl Todo {
    /// 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}]"),
            })
    }

    /// Todo の完了状態を変更する
```

```rust
    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}]"),
        })
    }
}
```

1.5.4 ユーザーデータの管理 use.rs

```rust
//! ユーザー情報の操作

use super::*;
use bcrypt::{hash, DEFAULT_COST};
use log::error;

use crate::database::*;

impl Todo {
    /// 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(())
    }
}
```

1.5.5 アプリケーション状態の管理 app_state.rs

```rust
//! アプリケーション状態の管理

use super::*;
use crate::database::*;
use bcrypt::verify;
use uuid::Uuid;

impl Todo {
    /// 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)
        }
    }
}
```

1.5.6 Todo モジュールのテスト test.rs

```rust
use super::*;
use crate::config::ItemSortOrder;
use chrono::Local;
use sqlx::MySqlPool;
use uuid::Uuid;

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);
```

```
46
47      // テスト用ユーザーの生成及び、ログイン
48      let user_name = "testdayo";
49      let user_pass = "passwordnano";
50
51      todo.add_user(user_name, user_pass).await.unwrap();
52      let sess = todo.login(user_name, user_pass).await.unwrap();
53
54      // 正しいセッションを検索する。
55      let new_sess = todo.is_valid_session(&sess).await.unwrap();
56      match new_sess {
57          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。"),
58          None => unreachable!("正しいセッションが見つからなかった。"),
59      };
60
61      // 間違ったセッションを検索する。
62      let none_sess = todo.is_valid_session(&Uuid::now_v7()).await.unwrap();
63      if none_sess.is_some() {
64          unreachable!("こんなセッションがあるわけがない。");
65      }
66  }
67
68  #[sqlx::test]
69  async fn add_todo_test(pool: MySqlPool) {
70      use chrono::Days;
71
72      let todo = Todo::test_new(pool);
73      let sess = login_for_test(&todo).await;
74
75      let item1 = ItemTodo {
76          id: 100,
77          user_name: "kore_naihazu".to_string(),
78          title: "テストアイテム 1 件目".to_string(),
79          work: Some("これは、中身を入れる。".to_string()),
80          update_date: None,
81          start_date: Some(Local::now().date_naive() - Days::new(1)),
82          end_date: Some(Local::now().date_naive() + Days::new(5)),
83          done: true,
84      };
85      let item2 = ItemTodo {
86          id: 100,
87          user_name: "kore_naihazu".to_string(),
88          title: "テストアイテム 2 件目 (work=null)".to_string(),
89          work: Some("".to_string()),
90          update_date: None,
91          start_date: Some(Local::now().date_naive() - Days::new(1)),
```

```
92          end_date: Some(Local::now().date_naive() + Days::new(5)),
93          done: true,
94      };
95      let item3 = ItemTodo {
96          id: 100,
97          user_name: "kore_naihazu".to_string(),
98          title: "テストアイテム 3 件目 (work=space)".to_string(),
99          work: Some(" \t  ".to_string()),
100         update_date: None,
101         start_date: Some(Local::now().date_naive() - Days::new(1)),
102         end_date: Some(Local::now().date_naive() + Days::new(5)),
103         done: true,
104     };
105     todo.add_todo(sess, &item1)
106         .await
107         .expect("1 件目の追加に失敗");
108     let res = todo
109         .get_todo_list(sess, true, ItemSortOrder::EndAsc)
110         .await
111         .expect("1 件目の取得に失敗");
112     assert_eq!(res.len(), 1, "一件目が取得できなかった?");
113     assert_eq!(res[0].title, item1.title, "一件目の title が違う");
114     assert_eq!(res[0].work, item1.work, "一件目の work が違う");
115     assert_eq!(res[0].user_name, "testdayo", "一件目の user_name が違う");
116     assert_eq!(
117         res[0].update_date,
118         Some(Local::now().date_naive()),
119         "一件目の update_date が違う"
120     );
121     assert_eq!(res[0].start_date, item1.start_date, "一件目の開始日が違う");
122     assert_eq!(res[0].end_date, item1.end_date, "一件目の終了日が違う");
123     assert!(!res[0].done, "一件目の完了マークが違う");
124
125     todo.add_todo(sess, &item2)
126         .await
127         .expect("二件目の追加に失敗");
128     let res = todo
129         .get_todo_list(sess, true, ItemSortOrder::EndAsc)
130         .await
131         .expect("二件目の取得に失敗");
132     assert_eq!(res.len(), 2, "二件あるはずなんだけど");
133     assert!(
134         res.iter()
135             .find(|&x| match x.title.find("work=null") {
136                 Some(n) => n > 0,
137                 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, ItemSortOrder::EndAsc)
        .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, ItemSortOrder::EndAsc)
        .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。");
```

```
184     let items = todo
185         .get_todo_list(sess, true, ItemSortOrder::EndAsc)
186         .await
187         .unwrap();
188     assert_eq!(
189         items.len(),
190         2,
191         "一件完了済みにしたので、このリストは 2 件しかない。"
192     );
193     let items = todo
194         .get_todo_list(sess, false, ItemSortOrder::EndAsc)
195         .await
196         .unwrap();
197     assert_eq!(items.len(), 3, "完了済みを含むので、3 件になる。");
198     let item = items
199         .iter()
200         .find(|&i| i.id == id)
201         .expect("さっきあった id だから必ずある。");
202     assert!(item.done, "さっき完了済みに変更した。");
203 
204     let max_id = items.iter().max_by_key(|&x| x.id).unwrap().id;
205     let res = todo.change_done(max_id + 1, sess, false).await;
206     match res {
207         Ok(_) => unreachable!("この id の todo があるはずがない。"),
208         Err(TodoError::NotFoundTodo) => {}
209         Err(e) => unreachable!("このエラーもありえない。[{e}]"),
210     };
211 
212     // 間違ったセッションのテスト
213     let res = todo.change_done(id, Uuid::now_v7(), true).await;
214     match res {
215         Ok(_) => unreachable!("このセッションでは、更新を許してはいけない。"),
216         Err(TodoError::NotFoundTodo) => { /* 正常 */ }
217         Err(e) => unreachable!("このエラーもおかしい。[{e}]"),
218     }
219 }
220 
221 #[sqlx::test]
222 async fn edit_todo_test(pool: MySqlPool) {
223     let todo = Todo::test_new(pool);
224     let sess = login_for_test(&todo).await;
225     create_todo_for_test(&todo, sess).await;
226 
227     let items = todo
228         .get_todo_list(sess, false, ItemSortOrder::EndAsc)
229         .await
```

```
230             .unwrap();
231         let mut item = items
232             .iter()
233             .find(|&i| i.title.contains("1 件目"))
234             .unwrap()
235             .clone();
236         item.title = "更新した一件目".to_string();
237         if let Err(e) = todo.edit_todo(&item, sess).await {
238             unreachable!("更新処理に失敗した。[{e}]");
239         }
240         let Some(item_new) = todo
241             .get_todo_list(sess, false, ItemSortOrder::EndAsc)
242             .await
243             .unwrap()
244             .iter()
245             .find(|&i| i.title.contains("更新した一件目"))
246             .cloned()
247         else {
248             unreachable!("更新したレコードが見つからないよ?");
249         };
250         assert_eq!(item.id, item_new.id, "更新したレコードの id が化けてる");
251 
252         // ニセセッションで試す
253         match todo.edit_todo(&item, Uuid::now_v7()).await {
254             Ok(_) => unreachable!("偽のセッションで更新成功してはならない。"),
255             Err(TodoError::NotFoundTodo) => { /* 正常 */ }
256             Err(e) => unreachable!("偽セッションのときのエラー:{e}"),
257         }
258     }
259 
260     async fn login_for_test(todo: &Todo) -> Uuid {
261         let user_name = "testdayo";
262         let user_pass = "passrordnona";
263         todo.add_user(user_name, user_pass).await.unwrap();
264         todo.login(user_name, user_pass).await.unwrap()
265     }
266 
267     async fn create_todo_for_test(todo: &Todo, sess: Uuid) {
268         use chrono::Days;
269         let items = [
270             ItemTodo {
271                 id: 100,
272                 user_name: "kore_naihazu".to_string(),
273                 title: "テストアイテム 1 件目".to_string(),
274                 work: Some("これは、中身を入れる。".to_string()),
275                 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();
    }
}
```

1.6 データベースアクセスモジュール database.rs

```rust
//! データベースの操作を司る
mod new;
mod session;
#[cfg(test)]
mod test;
mod todo;
mod user;

use chrono::NaiveDate;
use log::error;
use serde::{Deserialize, Serialize};
use sqlx::{
    mysql::{MySqlPool, MySqlPoolOptions},
    prelude::*,
};
use thiserror::Error;

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

#[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>,
    pub start_date: Option<NaiveDate>,
    pub end_date: Option<NaiveDate>,
    pub done: bool,
}

#[derive(Error, Debug)]
pub enum DbError {
    #[error("データベースへの接続に失敗。")]
```

```
45      FailConnect(sqlx::Error),
46      #[error("データベース操作失敗 (一般)")]
47      FailDbAccess(sqlx::Error),
48      #[error("User 挿入失敗 (name 重複)")]
49      DuplicateUserName(sqlx::Error),
50      #[error("ユーザーが見つかりません。")]
51      NotFoundUser,
52      #[error("指定されたセッション id が見つかりません。")]
53      NotFoundSession,
54      #[error("指定された id の todo が見つかりません。")]
55      NotFoundTodo,
56  }
```

1.6.1 新規作成 new.rs

```rust
//! database 構造体新規作成

use super::*;

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 })
    }
}
```

1.6.2 セッション管理 session.rs

```rust
//! セッション情報の操作
use super::*;
use sqlx::{prelude::*, query};
use uuid::Uuid;

impl Database {
    /// 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)
    }

    /// 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)
    }

    /// 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)
```

```
92              .await
93              .map_err(DbError::FailDbAccess)?
94              .get("cnt");
95          if sess_cnt == 1 {
96              Ok(true)
97          } else {
98              Ok(false)
99          }
100     }
101 }
```

1.6.3 todo アイテム管理 todo.rs

```rust
//! todo アイテム操作
use super::*;
use crate::config::ItemSortOrder;
use chrono::{Local, NaiveDate};
use sqlx::{query, query_as};
use uuid::Uuid;

impl Database {
    /// 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,
        sort_order: ItemSortOrder,
    ) -> 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 sql3 = match sort_order {
            ItemSortOrder::EndAsc => " order by end_date, update_date",
            ItemSortOrder::EndDesc => " order by end_date desc,  update_date",
            ItemSortOrder::StartAsc => " order by start_date, update_date",
            ItemSortOrder::StartDesc => " order by start_date desc, update_date",
            ItemSortOrder::UpdateAsc => " order by update_date, end_date",
            ItemSortOrder::UpdateDesc => " order by update_date desc, end_date",
        };
        let sql = if only_incomplete {
            format!("{} {} {};", sql1, sql2, sql3)
        } else {
            format!("{} {};", sql1, sql3)
        };
        let items = query_as::<_, ItemTodo>(&sql)
            .bind(sess.to_string())
            .bind(ref_date)
            .fetch_all(&self.pool)
            .await
            .map_err(DbError::FailDbAccess)?;

        Ok(items)
    }

    /// 指定 id の Todo 項目を取得する。
    /// 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。
    pub async fn get_todo_item_with_id(&self, id: u32, sess: Uuid) -> Result<ItemTodo, DbError> {
        let sql = r#"
            select t.id, t.user_name, t.title, t.work, t.update_date, t.start_date, t.end_date,
            ↪  t.done
            from todo t join sessions s on s.user_name = t.user_name
            where s.id=? and t.id=?
            "#;
        query_as::<_, ItemTodo>(sql)
            .bind(sess.to_string())
            .bind(id)
            .fetch_one(&self.pool)
            .await
            .map_err(|e| match e {
                sqlx::Error::RowNotFound => DbError::NotFoundTodo,
                e => DbError::FailDbAccess(e),
            })
```

```
    }

    /// Todo の完了状態を更新する。
    pub async fn change_done(&self, id: u32, done: bool) -> Result<(), DbError> {
        let sql = "update todo set done = ? where id = ?";
        let res = query(sql)
            .bind(done)
            .bind(id)
            .execute(&self.pool)
            .await
            .map_err(DbError::FailDbAccess)?;
        if res.rows_affected() > 0 {
            Ok(())
        } else {
            Err(DbError::NotFoundTodo)
        }
    }

    /// Todo の項目編集
    pub async fn edit_todo(&self, item: &ItemTodo) -> Result<(), DbError> {
        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());

        let sql = r#"
            update todo
            set title=?, work=?, update_date=curdate(), start_date=?, end_date=?
            where id=?;
            "#;
        let res = query(sql)
            .bind(&item.title)
            .bind(&item.work)
            .bind(start_date)
            .bind(end_date)
            .bind(item.id)
            .execute(&self.pool)
            .await
            .map_err(DbError::FailDbAccess)?;
        if res.rows_affected() > 0 {
            Ok(())
        } else {
            Err(DbError::NotFoundTodo)
        }
    }
}
```

1.6.4 ユーザーデータ管理 user.rs

```rust
//! ユーザーデータの操作
use super::*;
use sqlx::{query, query_as};
use uuid::Uuid;

impl Database {
    /// 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 ID をキーにしてユーザー情報を取得
    pub async fn get_user_from_sess(&self, sess: Uuid) -> Result<User, DbError> {
        let sql = r#"
            select u.name, u.password
            from users u join sessions s on u.name=s.user_name
```

```
            where s.id = ?;
            "#;

        query_as(sql)
            .bind(sess.to_string())
            .fetch_one(&self.pool)
            .await
            .map_err(|e| match e {
                sqlx::Error::RowNotFound => DbError::NotFoundSession,
                e => DbError::FailDbAccess(e),
            })
    }
}
```

1.6.5 データベースモジュールテスト test.rs

```rust
//! database モジュールテスト

use crate::config::ItemSortOrder;
use chrono::{Days, Local};
use sqlx::query;
use uuid::Uuid;

use super::*;

/// テスト用の Database 生成。テスト用 Pool をインジェクション
impl Database {
    pub(crate) fn new_test(pool: MySqlPool) -> Self {
        Self { pool }
    }
}

/// ユーザー生成のテスト
#[sqlx::test]
async fn test_add_user_and_get_user(pool: MySqlPool) {
    let db = Database::new_test(pool);
    db.add_user("hyara", "password").await.unwrap();
    let user = db.get_user("hyara").await.unwrap();
    assert_eq!(user.name, "hyara");
    assert_eq!(user.password, "password");
    let error_user = db.get_user("naiyo").await;
    match error_user {
        Ok(_) => unreachable!("結果が帰ってくるはずがない。"),
        Err(DbError::NotFoundUser) => { /* 正常 */ }
        Err(e) => unreachable!("このエラーはおかしい。{e}"),
    }
}

/// セッション生成関係の一連のテスト。
#[sqlx::test]
async fn test_make_new_session(pool: MySqlPool) {
    println!("まずはテスト用のユーザーの生成");
    let db = Database::new_test(pool);
    let user_name = "nekodayo";
    let password = "password";
    db.add_user(user_name, password).await.unwrap();

    println!("次に、普通にセッションを作ってみる。");
    let sess1 = db.make_new_session(user_name).await.unwrap();
    println!("セッション生成成功 id=[{}]", sess1);

```

```
    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。");
    let sess2 = db.make_new_session("detarame").await;
    match sess2 {
        Ok(_) => unreachable!("このユーザーは存在しなかったはず。"),
        Err(DbError::NotFoundUser) => { /* 正常 */ }
        Err(e) => unreachable!("このエラーもおかしい。[{}]", e),
    }

    println!("普通に、セッションを更新してみる。");
    let sess3 = db.update_session(&sess1).await.unwrap();
    assert_ne!(sess1, sess3);

    println!("ないはずのセッションを更新しようとしてみる。");
    let sess4 = Uuid::now_v7();
    let sess5 = db.update_session(&sess4).await;
    match sess5 {
        Ok(_) => unreachable!("このセッションはないはずなのに。"),
        Err(DbError::NotFoundSession) => { /* 正常 */ }
        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}"),
    }
}

/// セッションが有効かどうかを確認するテスト
#[sqlx::test]
async fn test_is_session_valid(pool: MySqlPool) {
    let db = Database::new_test(pool);

    println!("テスト用ユーザーの作成");
    let name = "nekodayo";
    let pass = "nekodamon";
    db.add_user(name, pass).await.unwrap();

    println!("新規セッションを生成する。");
    let sess = db.make_new_session(name).await.unwrap();
    println!("生成したセッション ID は、[{}] です。", &sess);

    println!("今作ったセッション ID の妥当性を問い合わせてみる。");
    assert!(db.is_session_valid(&sess).await.unwrap());

    println!("偽セッション ID をいれて、問い合わせてみる。");
    assert!(!db.is_session_valid(&Uuid::now_v7()).await.unwrap());
}

/// todo の書き込みと、単純な読み出しのテスト
#[sqlx::test]
async fn test_add_todo(pool: MySqlPool) {
```

```
92      let db = Database::new_test(pool);
93      let sess = login_for_test(&db).await;
94      let name = db.get_user_from_sess(sess).await.unwrap().name;
95
96      println!("テストデータをインサート");
97      let mut item = ItemTodo {
98          id: 0,
99          user_name: name.to_string(),
100         title: "インサートできるかな?".to_string(),
101         work: Some("中身入り".to_string()),
102         update_date: None,
103         start_date: Some(Local::now().date_naive()),
104         end_date: Some(Local::now().date_naive() + Days::new(3)),
105         done: true,
106     };
107     db.add_todo_item(&item).await.unwrap();
108
109     println!("テストデータを読み出す。一件しかないはず");
110     let last_day = Local::now().date_naive() + Days::new(1);
111     let res = db
112         .get_todo_item(sess, last_day, true, ItemSortOrder::EndAsc)
113         .await
114         .unwrap();
115     assert_eq!(res.len(), 1, "あれ?一件のはずだよ");
116     item.id = res[0].id;
117     item.update_date = Some(Local::now().date_naive());
118     item.done = false;
119 }
120
121 /// todoの書き込みと読み出し。
122 /// workが未入力の場合。
123 #[sqlx::test]
124 async fn test_add_todo_without_work(pool: MySqlPool) {
125     let db = Database::new_test(pool);
126     let sess = login_for_test(&db).await;
127     let name = db.get_user_from_sess(sess).await.unwrap().name;
128
129     println!("テストデータをインサート");
130     let mut item = ItemTodo {
131         id: 0,
132         user_name: name.to_string(),
133         title: "インサートできるかな?".to_string(),
134         work: None,
135         update_date: None,
136         start_date: Some(Local::now().date_naive()),
137         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, ItemSortOrder::EndAsc)
        .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, ItemSortOrder::EndAsc)
        .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, ItemSortOrder::EndAsc)
```

```
184             .await
185             .unwrap();
186         assert_eq!(res.len(), 1, "未完了だけだけど、一件あるはず。");
187 
188         println!("今作った job を完了済みにする。");
189         let sql = "update todo set done=true where id=?;";
190         query(sql).bind(res[0].id).execute(&pool).await.unwrap();
191         let res = db
192             .get_todo_item(sess, last_day, false, ItemSortOrder::EndAsc)
193             .await
194             .unwrap();
195         assert_eq!(res.len(), 1, "全部読み出しだけど一件あるはず。");
196         let res = db
197             .get_todo_item(sess, last_day, true, ItemSortOrder::EndAsc)
198             .await
199             .unwrap();
200         assert_eq!(res.len(), 0, "未完了だけだけだから、なにもないはず。");
201     }
202 
203     /// todo の書き込みと読み出し
204     /// 基準日の挙動テスト
205     #[sqlx::test]
206     async fn test_get_todo_ref_date(pool: MySqlPool) {
207         let db = Database::new_test(pool.clone());
208         let sess = login_for_test(&db).await;
209         let name = db.get_user_from_sess(sess).await.unwrap().name;
210 
211         println!("テストデータをインサート");
212         let item = ItemTodo {
213             id: 0,
214             user_name: name.to_string(),
215             title: "インサートできるかな?".to_string(),
216             work: None,
217             update_date: None,
218             start_date: Some(Local::now().date_naive()),
219             end_date: Some(Local::now().date_naive() + Days::new(3)),
220             done: false,
221         };
222         db.add_todo_item(&item).await.unwrap();
223 
224         let ref_date = Local::now().date_naive();
225         let res = db
226             .get_todo_item(sess, ref_date, true, ItemSortOrder::EndAsc)
227             .await
228             .unwrap();
229         assert_eq!(res.len(), 1, "基準日と開始日が同じだからみつかる。");
```

```
    let res = db
        .get_todo_item(sess, ref_date + Days::new(1), true, ItemSortOrder::EndAsc)
        .await
        .unwrap();
    assert_eq!(res.len(), 1, "開始日の翌日が基準日だからみつかる。");
    let res = db
        .get_todo_item(sess, ref_date - Days::new(1), true, ItemSortOrder::EndAsc)
        .await
        .unwrap();
    assert_eq!(res.len(), 0, "基準日が開始日の前日だからみつからない。");
    let res = db
        .get_todo_item(sess, ref_date + Days::new(4), true, ItemSortOrder::EndAsc)
        .await
        .unwrap();
    assert_eq!(res.len(), 1, "基準日が期限を過ぎているけどみつかるの。");
}

#[sqlx::test]
async fn test_get_user_from_sess(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;

    let user = db.get_user_from_sess(sess).await.unwrap();
    assert_eq!(user.name, name, "これはみつかるはず");
    let dummy_sess = Uuid::now_v7();
    let user = db.get_user_from_sess(dummy_sess).await;
    match user {
        Ok(_) => unreachable!("見つかるわけないでしょう。"),
        Err(DbError::NotFoundSession) => { /* 正常 */ }
        Err(e) => unreachable!("トラブルです。{e}"),
    };
}

#[sqlx::test]
async fn test_change_done(pool: MySqlPool) {
    let db = Database::new_test(pool);
    let sess = login_for_test(&db).await;
    let ref_date = Local::now().date_naive();
    create_todo_for_test(&db, sess).await;

    let items = db
        .get_todo_item(sess, ref_date, true, ItemSortOrder::EndAsc)
        .await
        .unwrap();
```

```
    let item = items.iter().find(|&i| i.title.contains("二件目")).unwrap();
    db.change_done(item.id, true).await.unwrap();

    let items = db
        .get_todo_item(sess, ref_date, true, ItemSortOrder::EndAsc)
        .await
        .unwrap();
    let item = items.iter().find(|&i| i.title.contains("二件目"));
    assert!(item.is_none(), "状態を完了にしたので見つからないはず。");

    let items = db
        .get_todo_item(sess, ref_date, false, ItemSortOrder::EndAsc)
        .await
        .unwrap();
    let item = items.iter().find(|&i| i.title.contains("二件目"));
    match item {
        Some(i) => assert!(i.done, "完了済みになっているはずですね?"),
        None => unreachable!("状態を変えたら、レコードなくなった???"),
    }
    assert_eq!(
        items.len(),
        3,
        "全件見ているのでレコードは３件あるはずですが?"
    );
}

#[sqlx::test]
async fn test_get_todo_with_id(pool: MySqlPool) {
    let db = Database::new_test(pool);
    let sess = login_for_test(&db).await;
    create_todo_for_test(&db, sess).await;

    let items = db
        .get_todo_item(
            sess,
            Local::now().date_naive(),
            false,
            ItemSortOrder::EndAsc,
        )
        .await
        .unwrap();
    let id = items
        .iter()
        .find(|&i| i.title.contains("一件目"))
        .expect("これはあるはず")
        .id;
```

```
322     let non_exist_id = items.iter().max_by_key(|&i| i.id).unwrap().id + 1;
323
324     // 正常な読み出し
325     let res = db
326         .get_todo_item_with_id(id, sess)
327         .await
328         .expect("これは正常に読み出せるはず。エラーはだめ");
329     res.work
330         .expect("このレーコードは work を持つはずです。")
331         .find("働いてます。")
332         .expect("work の内容がおかしい。");
333
334     // 間違った id
335     let res = db.get_todo_item_with_id(non_exist_id, sess).await;
336     match res {
337         Ok(_) => unreachable!("そんな ID は存在しなかったはずなのに。"),
338         Err(DbError::NotFoundTodo) => { /* 正常 */ }
339         Err(e) => unreachable!("データベースエラーだよ。({e})"),
340     }
341
342     // 間違ったセッション
343     let res = db.get_todo_item_with_id(id, Uuid::now_v7()).await;
344     match res {
345         Ok(_) => unreachable!("そんなセッションはないはず。"),
346         Err(DbError::NotFoundTodo) => { /* 正常 */ }
347         Err(e) => unreachable!("データベースエラー発生。({e})"),
348     }
349 }
350
351 #[sqlx::test]
352 async fn test_edit(pool: MySqlPool) {
353     let db = Database::new_test(pool);
354     let sess = login_for_test(&db).await;
355     create_todo_for_test(&db, sess).await;
356
357     // 書き込みテスト用レコードの取得
358     let today = Local::now().date_naive();
359     let items = db
360         .get_todo_item(sess, today, false, ItemSortOrder::EndAsc)
361         .await
362         .unwrap();
363     let mut item = items
364         .iter()
365         .find(|&i| i.title.contains("一件目"))
366         .expect("ないはずがない。")
367         .clone();
```

```
368     item.title = "更新しました。".to_string();
369     item.work = Some("書き換え後".to_string());
370     item.start_date = Some(today - Days::new(5));
371     item.end_date = Some(today + Days::new(10));
372     db.edit_todo(&item).await.expect("更新がエラーを起こした。");
373     // 書き込み後の照合
374     let items_new = db
375         .get_todo_item(sess, today, false, ItemSortOrder::EndAsc)
376         .await
377         .unwrap();
378     let item_new = items_new
379         .iter()
380         .find(|&i| i.title.contains("更新しました。"))
381         .expect("更新されたレコードが存在しない。");
382     assert_eq!(
383         item_new.work,
384         Some("書き換え後".to_string()),
385         "更新後の work がおかしい"
386     );
387     assert_eq!(
388         item_new.start_date,
389         Some(today - Days::new(5)),
390         "更新後の start_date がおかしい"
391     );
392     assert_eq!(
393         item_new.end_date,
394         Some(today + Days::new(10)),
395         "更新後の end_date がおかしい"
396     );
397
398     // 存在しないレコードの更新
399     let id_max_plus_one = items.iter().max_by_key(|&i| i.id).unwrap().id + 1;
400     item.id = id_max_plus_one;
401     let res = db.edit_todo(&item).await;
402     match res {
403         Ok(_) => unreachable!("更新できちゃだめっ"),
404         Err(DbError::NotFoundTodo) => {}
405         Err(e) => unreachable!("db_err: {e}"),
406     }
407 }
408
409 #[sqlx::test]
410 async fn test_sort_end_date(pool: MySqlPool) {
411     let db = Database::new_test(pool);
412     let sess = login_for_test(&db).await;
413     create_todo_for_test(&db, sess).await;
```

```
414
415     let today = Local::now().date_naive();
416     let recs = db
417         .get_todo_item(sess, today, false, ItemSortOrder::EndAsc)
418         .await
419         .expect("取得時にエラーを起こした。");
420     eprintln!("取得データ（昇順）");
421     eprintln!("0 => {:?}", recs[0]);
422     eprintln!("1 => {:?}", recs[1]);
423     eprintln!("2 => {:?}", recs[2]);
424     assert!(
425         recs[0].end_date <= recs[1].end_date,
426         "終了日が昇順になってない。"
427     );
428     assert!(
429         recs[1].end_date <= recs[2].end_date,
430         "終了日が昇順になってない（2）。"
431     );
432
433     let recs = db
434         .get_todo_item(sess, today, false, ItemSortOrder::EndDesc)
435         .await
436         .expect("取得時にエラーを起こした（2）");
437     eprintln!("取得データ（降順）");
438     eprintln!("0 => {:?}", recs[0]);
439     eprintln!("1 => {:?}", recs[1]);
440     eprintln!("2 => {:?}", recs[2]);
441     assert!(
442         recs[0].end_date >= recs[1].end_date,
443         "終了日が降順になってない（1）"
444     );
445     assert!(
446         recs[1].end_date >= recs[2].end_date,
447         "終了日が降順になってない（2）"
448     );
449 }
450
451 #[sqlx::test]
452 async fn test_sort_start_date(pool: MySqlPool) {
453     let db = Database::new_test(pool);
454     let sess = login_for_test(&db).await;
455     create_todo_for_test(&db, sess).await;
456
457     let today = Local::now().date_naive();
458     let recs = db
459         .get_todo_item(sess, today, false, ItemSortOrder::StartAsc)
```

```
460             .await
461             .expect("取得時にエラーを起こした。");
462         assert!(
463             recs[0].start_date <= recs[1].start_date,
464             "開始日が昇順になってない。"
465         );
466         assert!(
467             recs[1].start_date <= recs[2].start_date,
468             "開始日が昇順になってない (2)。"
469         );
470 
471         let recs = db
472             .get_todo_item(sess, today, false, ItemSortOrder::StartDesc)
473             .await
474             .expect("取得時にエラーを起こした (2)");
475         assert!(
476             recs[0].start_date >= recs[1].start_date,
477             "開始日が降順になってない (1)"
478         );
479         assert!(
480             recs[1].start_date >= recs[2].start_date,
481             "開始日が降順になってない (2)"
482         );
483     }
484 
485     #[sqlx::test]
486     async fn test_sort_update_date(pool: MySqlPool) {
487         let db = Database::new_test(pool);
488         let sess = login_for_test(&db).await;
489         create_todo_for_test(&db, sess).await;
490         let today = Local::now().date_naive();
491 
492         // Database のインターフェースで update_date を更新するすべはないので直接編集
493         let keys = db
494             .get_todo_item(sess, today, false, ItemSortOrder::EndAsc)
495             .await
496             .unwrap()
497             .iter()
498             .map(|r| r.id)
499             .collect::<Vec<_>>();
500         let sql = "update todo set update_date = ? where id = ?";
501         let days = [
502             today + Days::new(2),
503             today + Days::new(1),
504             today + Days::new(3),
505         ];
```

```
506     for i in 0..3 {
507         query(sql)
508             .bind(days[i])
509             .bind(keys[i])
510             .execute(&db.pool)
511             .await
512             .unwrap();
513     }
514 
515     let recs = db
516         .get_todo_item(sess, today, false, ItemSortOrder::UpdateAsc)
517         .await
518         .expect("取得時にエラーを起こした。");
519     assert!(
520         recs[0].update_date <= recs[1].update_date,
521         "更新日が昇順になってない。"
522     );
523     assert!(
524         recs[1].update_date <= recs[2].update_date,
525         "更新日が昇順になってない (2)。"
526     );
527 
528     let recs = db
529         .get_todo_item(sess, today, false, ItemSortOrder::UpdateDesc)
530         .await
531         .expect("取得時にエラーを起こした (2)");
532     assert!(
533         recs[0].update_date >= recs[1].update_date,
534         "更新日が降順になってない (1)"
535     );
536     assert!(
537         recs[1].update_date >= recs[2].update_date,
538         "更新日が降順になってない (2)"
539     );
540 }
541 
542 async fn login_for_test(db: &Database) -> Uuid {
543     println!("テスト用ユーザー及びセッションの生成");
544     let name = "test";
545     let pass = "test";
546     db.add_user(name, pass).await.unwrap();
547     db.make_new_session(name).await.unwrap()
548 }
549 
550 async fn create_todo_for_test(db: &Database, sess: Uuid) {
551     let name = db.get_user_from_sess(sess).await.unwrap().name;
```

```
552
553     println!("テストデータをインサート");
554     let item = ItemTodo {
555         id: 0,
556         user_name: name.to_string(),
557         title: "一件目（work 有り）".to_string(),
558         work: Some("働いてます。".to_string()),
559         update_date: None,
560         start_date: Some(Local::now().date_naive() - Days::new(4)),
561         end_date: Some(Local::now().date_naive() + Days::new(2)),
562         done: false,
563     };
564     db.add_todo_item(&item).await.unwrap();
565
566     let item = ItemTodo {
567         id: 0,
568         user_name: name.to_string(),
569         title: "二件目（work 無し）".to_string(),
570         work: None,
571         update_date: None,
572         start_date: Some(Local::now().date_naive() - Days::new(5)),
573         end_date: Some(Local::now().date_naive() + Days::new(1)),
574         done: false,
575     };
576     db.add_todo_item(&item).await.unwrap();
577
578     let item = ItemTodo {
579         id: 0,
580         user_name: name.to_string(),
581         title: "三件目（work 無し）".to_string(),
582         work: None,
583         update_date: None,
584         start_date: Some(Local::now().date_naive()),
585         end_date: Some(Local::now().date_naive() + Days::new(3)),
586         done: false,
587     };
588     db.add_todo_item(&item).await.unwrap();
589 }
```

1.7 command モジュール tauri::command 関数群 command.rs

```rust
//! フロントエンドとのインターフェース  tauri::command
pub mod app_state;
pub mod session;
pub mod todo;
pub mod user;
```

1.7.1 todo リストの表示・編集　todo.rs

```rust
//! todo リスト操作インターフェース

use super::session::{get_cur_session_with_update, get_curr_session};
use crate::app_status::AppStatus;
use crate::database::ItemTodo;
use log::{debug, info};
use serde::Deserialize;
use tauri::State;

/// todo のリストを取得する。
#[tauri::command]
pub 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()),
    };

    let is_incomplete;
    let sort_order;
    {
        let conf = app_status.config().lock().unwrap();
        is_incomplete = conf.get_is_incomplete();
        sort_order = conf.get_item_sort_order();
    }

    let ret = app_status
        .todo()
        .get_todo_list(sess, is_incomplete, sort_order)
        .await
        .map_err(|e| e.to_string())?;
    info!("todo リスト、{}件、取得完了", ret.len());
    Ok(ret)
}

/// todo アイテムを取得する
#[tauri::command]
pub 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());
    };

    let ret = app_status
```

```
        .todo()
        .get_todo_with_id(id, sess)
        .await
        .map_err(|e| e.to_string())?;
    info!("todo 一件の取得完了  id=>{}", id);
    Ok(ret)
}

/// todo を追加する。
#[tauri::command]
pub 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(|e| e.to_string())?;
    info!("todo の追加完了");
    Ok(())
}

/// todo の完了状態を変更する。
#[tauri::command]
pub 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(|e| e.to_string())?;
    info!(
        "todo の状態を変更。id=>{}, state=>{}",
```

```
92          id,
93          if done { "完了" } else { "未完了" }
94      );
95      Ok(())
96  }
97
98  /// todo の編集を行う。
99  #[tauri::command]
100 pub async fn edit_todo(
101     app_status: State<'_, AppStatus>,
102     id: u32,
103     item: FormTodo,
104 ) -> Result<(), String> {
105     let sess = match get_cur_session_with_update(&app_status).await {
106         Ok(Some(u)) => u,
107         Ok(None) => return Err("NotLogin".to_string()),
108         Err(e) => return Err(e),
109     };
110
111     debug!("input => id: {},  item: {:?}", id, &item);
112     let mut item: ItemTodo = item.into();
113     item.id = id;
114     app_status
115         .todo()
116         .edit_todo(&item, sess)
117         .await
118         .map_err(|e| e.to_string())?;
119     info!("アイテム編集完了 id=>{}", id);
120     Ok(())
121 }
122
123 /// Todo 項目追加画面データ取得用
124 #[derive(Deserialize, Debug, Clone)]
125 pub struct FormTodo {
126     title: String,
127     work: Option<String>,
128     start: Option<String>,
129     end: Option<String>,
130 }
131
132 impl From<FormTodo> for ItemTodo {
133     fn from(val: FormTodo) -> Self {
134         let start = val.start.map(|d| d.replace("/", "-").parse().unwrap());
135         let end = val.end.map(|d| d.replace("/", "-").parse().unwrap());
136         ItemTodo {
137             id: 0,
```

```
138                 user_name: "".to_string(),
139                 title: val.title,
140                 work: val.work,
141                 update_date: None,
142                 start_date: start,
143                 end_date: end,
144                 done: false,
145             }
146         }
147     }
```

1.7.2 ユーザー操作関係 user.rs

```rust
//! ユーザー操作インターフェース

use crate::app_status::AppStatus;
use log::info;
use tauri::{command, State};

// ユーザー登録
#[command]
pub async fn regist_user(
    app_status: State<'_, AppStatus>,
    name: String,
    password: String,
) -> Result<(), String> {
    app_status
        .todo()
        .add_user(&name, &password)
        .await
        .map_err(|e| e.to_string())?;
    info!("ユーザー登録完了:user->{}", &name);
    Ok(())
}

/// ログイン
#[command]
pub 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))?;
    info!("ログイン完了:user->{}", &name);
    Ok(session.to_string())
}
```

1.7.3 セッション操作関連 session.rs

```rust
//! セッション関係の関数及びインターフェース

use crate::app_status::AppStatus;
use log::info;
use tauri::{command, State};
use uuid::Uuid;

/// 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]
pub 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),
    }
    sess
}

/// 現在、有効なセッションを返す。
/// 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。
pub async fn get_cur_session_with_update(app_status: &AppStatus) -> Result<Option<Uuid>, String> {
    let cur_session = get_curr_session(app_status);
    let Some(cur_session) = cur_session else {
        return Ok(None);
    };

    match app_status.todo().is_valid_session(&cur_session).await {
        Ok(Some(s)) => {
            // 更新されたセッションを再登録
            let mut cnf = app_status.config().lock().unwrap();
            cnf.set_session_id(&s);
            //cnf.save().map_err(|e| format!("FailSession:{e}"))?;
            Ok(Some(s))
        }
        Ok(None) => Ok(None),
        Err(e) => Err(format!("FailSession:{e}")),
    }
}

/// 現在のセッションを取得する。
pub fn get_curr_session(app_status: &AppStatus) -> Option<Uuid> {
    let conf = app_status.config().lock().unwrap();
    conf.get_session_id()
```

```
46  }
```

1.7.4 アプリケーションの状態操作 app_state.rs

```rust
//! アプリケーションの全体ステータスの取得・設定用インターフェース

use crate::app_status::AppStatus;
use crate::config::ItemSortOrder;
use log::info;
use tauri::{command, State};

/// 完了済みのみを表示するかどうかを設定する。
#[tauri::command]
pub fn set_is_incomplete(app_status: State<'_, AppStatus>, is_incomplete: bool) {
    let mut conf = app_status.config().lock().unwrap();
    conf.set_is_incomplete(is_incomplete);
    info!("未完了のみ表示モードを{}にセット", is_incomplete);
}

/// 完了済みのみ表示モードの現在の値を取得する。
#[tauri::command]
pub fn get_is_incomplete(app_status: State<'_, AppStatus>) -> bool {
    app_status.config().lock().unwrap().get_is_incomplete()
}

/// 現在のアイテムリストのソート方法を返す
#[command]
pub fn get_item_sort_order(app_status: State<'_, AppStatus>) -> String {
    app_status
        .config()
        .lock()
        .unwrap()
        .get_item_sort_order()
        .to_string()
}

/// アイテムリストのソート方法を設定する
#[command]
pub fn set_item_sort_order(
    app_status: State<'_, AppStatus>,
    sort_order: String,
) -> Result<(), String> {
    let sort_order = sort_order
        .parse::<ItemSortOrder>()
        .map_err(|e| e.to_string())?;
    app_status
        .config()
        .lock()
        .unwrap()
```

```
46          .set_item_sort_order(sort_order);
47      info!("ソートオーダー更新 => {}", sort_order.to_string());
48      Ok(())
49  }
```

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

```
.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";
import PasteTodo from "./PasteTodo.jsx";

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 path="/pastetodo/:id" element={<PasteTodo/>}/>
            </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

```
/* 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

```
/* 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>
                <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}
                >
```

```
45                      <Input {...register("pass", {required: "入力は必須です。"},)}/>
46                  </FormControl>
47                  <Button type="submit" w="30%" ml="auto" mr="auto"> ログイン </Button>
48                  <Text>{sendMessage}</Text>
49              </VStack>
50          </>
51      );
52  }
53
54  export default Login;
55
```

2.9 todo リストの表示 TodoList.jsx

```jsx
import { useQuery, } from "@tanstack/react-query";
import { Container, Grid, GridItem, } from "@yamada-ui/react";
import { invoke } from "@tauri-apps/api/core";
import "./App.css";
import TodoItem from "./TodoItem.jsx";
import TodoItemToolbar from "./TodoListToolbar.jsx";

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,
    });

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

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

    console.log(todos);
    return (
        <>
            <Container gap="0" bg="backgound">
                <TodoItemToolbar/>

                <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>
            </Container>
        </>
    );
}
```

```
45
46
47  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, Container } from "@yamada-ui/react";
import { GrWorkshop } from "react-icons/gr";
import { BsAlarm } from "react-icons/bs";
import { BsEmojiGrin } from "react-icons/bs";
import { BiPencil } from "react-icons/bi";
import { FaRegCopy } from "react-icons/fa6";

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 onPasteClick = () => {
        navi("/pastetodo/"+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);

```

```
45      // 完了ボタンのアイコン選択
46      let done_icon;
47      if (item.done) {
48          done_icon = <BsEmojiGrin/>;
49      } else if (!!end_date && geDate(new Date(), end_date)) {
50          done_icon = <BsAlarm/>;
51      } else {
52          done_icon = <GrWorkshop/>
53      }
54
55      // 上部バーの背景色
56      const oneday = 24 * 60 * 60 * 1000;
57      const today = new Date();
58      const delivery = new Date(item.end_date);
59      const daysToDelivery = (delivery - today) / oneday;
60      let line_color;
61      if (daysToDelivery < 0) {
62          line_color = "danger";
63      } else if (daysToDelivery < 2) {
64          line_color = "warning";
65      } else {
66          line_color = "success";
67      }
68
69      return (
70          <Container p="1%" gap="0">
71              <SimpleGrid w="full" columns={{base: 2, md: 1}} gap="md" bg={line_color}>
72                  <GridItem>
73                      <HStack>
74                          <IconButton size="xs" icon={done_icon} onClick={onDoneClick}
75                              bg={line_color}/>
76                          <IconButton size="xs" icon={<BiPencil/>} onClick={onEditClick}
77                              bg={line_color}/>
78                          <IconButton size="xs" icon={<FaRegCopy/>} onClick={onPasteClick}
79                              bg={line_color}/>
80                      </HStack>
81                  </GridItem>
82
83                  <GridItem>
84                      <Text fontSize="xs" align="right">
85                          {update_date?.toLocaleDateString()}
86                      </Text>
87                  </GridItem>
88              </SimpleGrid>
89              <Text align="center" fontSize="lg" as="b">
90                  {item.title}
```

```
            </Text>
            <Text fontSize="sm">
                {item.work}
            </Text>
            <Text fontSize="sm">
                {start_date?.toLocaleDateString()} ~ {end_date?.toLocaleDateString()}
            </Text>
        </Container>
    );
}

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) {
            return day1 >= day2;
        } else {
            return month1 > month2;
        }
    } else {
        return year1 > year2;
    }
}

```

2.11 todo リスト画面　ツールバー TodoListToolbar.jsx

```jsx
import { useNavigate } from "react-router-dom";
import { useMutation, useQuery, useQueryClient } from "@tanstack/react-query";
import { HStack, IconButton, Select, Switch, Option } from "@yamada-ui/react";
import { invoke } from "@tauri-apps/api/core";
import { AiOutlineFileAdd } from "react-icons/ai";
import "./App.css";


export default function TodoListToolbar() {

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

    return (
        <>
            <HStack>
                <IconButton icon={<AiOutlineFileAdd/>} onClick={handleAddTodo}/>
                <SwitchIncomplete/>
                <SelectItemSortOrder/>
            </HStack>
        </>
    );
}

function SwitchIncomplete() {
    const queryClient = useQueryClient();
    const {data: IsIncomplete, isPending} = useQuery({
        queryKey: ['is_incomplete'],
        queryFn: () => invoke('get_is_incomplete') ,
    });

    const {mutate} = useMutation({
        mutationFn: (checked) => invoke('set_is_incomplete', {isIncomplete: checked}) ,
        onSuccess: () => {
            queryClient.invalidateQueries({queryKey: ['is_incomplete']});
            queryClient.invalidateQueries({queryKey: ['todo_list']});
        }
    });

    const onIsIncompleteChange = (e) => mutate(e.target.checked) ;

    if (isPending) {
        return (<p> Loading... </p>);
    }
```

```
45
46      return (
47          <Switch checked={IsIncomplete} onChange={onIsIncompleteChange}>
48              未完了のみ
49          </Switch>
50      );
51  }
52
53  function SelectItemSortOrder() {
54      const {data, isPending} = useQuery({
55          queryKey: ['item_sort_order'],
56          queryFn: () => invoke('get_item_sort_order') ,
57      });
58
59      const queryClient = useQueryClient();
60
61      const {mutate} = useMutation({
62          mutationFn: (sortOrder) => invoke('set_item_sort_order', {sortOrder: sortOrder}) ,
63          onSuccess: () => {
64              queryClient.invalidateQueries({queryKey: ['item_sort_order']});
65              queryClient.invalidateQueries({queryKey: ['todo_list']});
66          },
67          onError: (err) => console.log(err),
68      });
69
70      const onChange = (value) =>  mutate(value) ;
71
72      if (isPending) {
73          return (<p> loading </p>);
74      }
75
76      return (
77          <Select w="9em" value={data} onChange={onChange}>
78              <Option value="StartAsc">開始 (昇順)</Option>
79              <Option value="StartDesc">開始 (降順)</Option>
80              <Option value="EndAsc">終了 (昇順)</Option>
81              <Option value="EndDesc">終了 (降順)</Option>
82              <Option value="UpdateAsc">更新日 (昇順)</Option>
83              <Option value="UpdateDesc">更新日 (降順)</Option>
84          </Select>
85      );
86  }
```

2.12 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;
```

2.13 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,"/"),
    }

    return (
```

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

2.14 todo アイテムの複製 PasteTodo.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 PasteTodo() {
    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 = {
            item: {
                title: data.title,
                work: data.work,
                start: str2date(data.start)?.toLocaleDateString(),
                end: str2date(data.end)?.toLocaleDateString(),
            }
        };
        await invoke("add_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,"/"),
    }

    return (
        <>
```

```
45              <InputTodo send_data={handleSendData} init_val={initForm}/>
46          </>
47      );
48  }
```

2.15 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('/'),
        onError: (error) => setErrorMessage(error),
    });

    const onCancelClick = () => { navi('/'); };

    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"/>
```

```
45                      <InputDate name="end" label="終了"/>
46                      <HStack>
47                          <Button type="submit" w="30%" ml="auto" mr="5%">送信</Button>
48                          <Button w="30%" onClick={onCancelClick} mr="auto">キャンセル</Button>
49                      </HStack>
50                      <Text> {isPending ? "送信中です。" : null} </Text>
51                      <Text> {errorMessage} </Text>
52                  </VStack>
53              </FormProvider>
54          </>
55      );
56  }
57
58
59  function InputDate({name, label}) {
60      const { register, watch, formState: {errors} } = useFormContext();
61
62      const val = watch(name);
63      const [ date, setDate,  ] = useState(null);
64      useEffect(() => {
65          setDate(str2date(val));
66      }, [val]);
67
68      return (
69          <>
70              <FormControl
71                  invalid = {!!errors[name]}
72                  label={label}
73                  errorMessage={errors[name]?.message}>
74                  <HStack>
75                      <Input
76                          w="50%"
77                          placeholder="[[YYYY/]MM/]DD or +dd"
78                          {...register(name, {
79                              validate: (data) => {
80                              if (data==null) { return }
81                              if (data.length===0) { return }
82                              if (str2date(data)==null) { return "日付の形式が不正です。" }
83                              }
84                          })}
85                      />
86                      <Text> {date?.toLocaleDateString()} </Text>
87                  </HStack>
88              </FormControl>
89          </>
90      );
```

```
91  }
92
```

2.16 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;
        }
```

```
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)
    );

```