neko_todo ソースリスト

美都

2025 年 2 月 20 日

目次

1 Rust ソース

1.1 メインモジュール main.rs

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

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

use app_status::AppStatus;
use config::ItemSortOrder;
use database::ItemTodo;
use directories::ProjectDirs;
use log::{debug, error, info};
use serde::Deserialize;
use setup::setup;
use tauri::{command, Manager, State};
use uuid::Uuid;

fn main() {
    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();

    run()
}

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

    let app = tauri::Builder::default()
        .plugin(tauri_plugin_shell::init())
        .manage(app_status)
        .invoke_handler(tauri::generate_handler![
            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");
    app.run(|app, event| {
        if let tauri::RunEvent::Exit = event {
            info!("終了処理開始");
            let state = app.state::<AppStatus>();
            state.config().lock().unwrap().save().unwrap();
        }
    });
}
```

```rust

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

    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();
    }

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

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

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

/// todo を追加する。
#[tauri::command]
async fn add_todo(app_status: State<'_, AppStatus>, item: FormTodo) -> Result<(), String> {
    let sess = match get_cur_session_with_update(&app_status).await {
        Ok(Some(u)) => u,
        Ok(None) => return Err("NotLogin".to_string()),
        Err(e) => return Err(e),
    };
```

```
134
135     debug!("input = {:?}", &item);
136     app_status
137         .todo()
138         .add_todo(sess, &item.into())
139         .await
140         .map_err(Into::into)
141 }
142
143 /// todo の完了状態を変更する。
144 #[tauri::command]
145 async fn update_done(app_status: State<'_, AppStatus>, id: u32, done: bool) -> Result<(), String> {
146     let sess = match get_cur_session_with_update(&app_status).await {
147         Ok(Some(s)) => s,
148         Ok(None) => return Err("NotLogin".to_string()),
149         Err(e) => return Err(e),
150     };
151     app_status
152         .todo()
153         .change_done(id, sess, done)
154         .await
155         .map_err(Into::into)
156 }
157
158 /// todo の編集を行う。
159 #[tauri::command]
160 async fn edit_todo(
161     app_status: State<'_, AppStatus>,
162     id: u32,
163     item: FormTodo,
164 ) -> Result<(), String> {
165     let sess = match get_cur_session_with_update(&app_status).await {
166         Ok(Some(u)) => u,
167         Ok(None) => return Err("NotLogin".to_string()),
168         Err(e) => return Err(e),
169     };
170
171     debug!("input => id: {},  item: {:?}", id, &item);
172     let mut item: ItemTodo = item.into();
173     item.id = id;
174     app_status
175         .todo()
176         .edit_todo(&item, sess)
177         .await
178         .map_err(Into::into)
179 }
```

```rust

/// 完了済みのみを表示するかどうかを設定する。
#[tauri::command]
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]
fn get_is_incomplete(app_status: State<'_, AppStatus>) -> bool {
    app_status.config().lock().unwrap().get_is_incomplete()
}

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

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

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

/// 現在、有効なセッションが存在するかどうか確認。(ユーザ I/F 用)
#[command]
```

```rust
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
}

/// 現在のアイテムリストのソート方法を返す
#[command]
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]
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()
        .set_item_sort_order(sort_order);
    info!("ソートオーダー更新 => {}", sort_order.to_string());
    Ok(())
}

/// 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。
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。
fn get_curr_session(app_status: &AppStatus) -> Option<Uuid> {
    let conf = app_status.config().lock().unwrap();
    conf.get_session_id()
}

/// Todo 項目追加画面データ取得用
#[derive(Deserialize, Debug, Clone)]
struct FormTodo {
    title: String,
    work: Option<String>,
    start: Option<String>,
    end: Option<String>,
}

impl From<FormTodo> for ItemTodo {
    fn from(val: FormTodo) -> Self {
        let start = val.start.map(|d| d.replace("/", "-").parse().unwrap());
        let end = val.end.map(|d| d.replace("/", "-").parse().unwrap());
        ItemTodo {
            id: 0,
            user_name: "".to_string(),
            title: val.title,
            work: val.work,
            update_date: None,
            start_date: start,
            end_date: end,
            done: false,
        }
    }
}
```

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             std::env::remove_var(DB_HOST);
270             std::env::remove_var(DB_USER);
271             std::env::remove_var(DB_USER);
272         }
273 }
```

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

use crate::{config::ItemSortOrder, database::*};

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

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

    /// todo の一覧を取得する。(仮実装。インターフェース未確定)
    pub async fn get_todo_list(
        &self,
        sess: Uuid,
        only_imcomplete: bool,
        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}]"),
            })
    }

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

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

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

```
91      }
92
93      /// Todo の完了状態を変更する
94      pub async fn change_done(&self, id: u32, sess: Uuid, done: bool) -> Result<(), TodoError> {
95          self.get_todo_with_id(id, sess).await?;
96          self.database
97              .change_done(id, done)
98              .await
99              .map_err(|e| match e {
100                 DbError::FailDbAccess(e) => {
101                     error!("[Todo::change_done]change_done:[{e}]");
102                     TodoError::FailDbAccess(e)
103                 }
104                 DbError::NotFoundTodo => TodoError::NotFoundTodo,
105                 e => unreachable!("[change_done]change_done[{e}]"),
106             })
107     }
108
109     /// Todo の編集を行う。
110     pub async fn edit_todo(&self, item: &ItemTodo, sess: Uuid) -> Result<(), TodoError> {
111         let mut item = item.clone();
112         if let Some(ref s) = item.work {
113             if s.trim().is_empty() {
114                 item.work = None;
115             }
116         }
117         self.get_todo_with_id(item.id, sess).await?;
118         self.database.edit_todo(&item).await.map_err(|e| match e {
119             DbError::FailDbAccess(e) => {
120                 error!("[Todo::edit_todo]edit_todo:[{e}]");
121                 TodoError::FailDbAccess(e)
122             }
123             DbError::NotFoundTodo => TodoError::NotFoundTodo,
124             e => unreachable!("[edit_todo]edit_todo[{e}]"),
125         })
126     }
127
128     /// ユーザーの追加を行う。
129     pub async fn add_user(&self, name: &str, password: &str) -> Result<(), TodoError> {
130         let hashed_pass = hash(password, DEFAULT_COST)?;
131         if let Err(e) = self.database.add_user(name, &hashed_pass).await {
132             match e {
133                 DbError::DuplicateUserName(e) => return Err(TodoError::DuplicateUser(e)),
134                 DbError::FailDbAccess(e) => {
135                     error!("[Todo::add_user]Database::add_user:[{e}]");
136                     return Err(TodoError::FailDbAccess(e));
```

```
                }
                _ => {}
            }
        }
        Ok(())
    }

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

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

```
183             })?;
184         if is_valid {
185             match self.database.update_session(sess).await {
186                 Ok(s) => Ok(Some(s)),
187                 Err(DbError::NotFoundSession) => Ok(None),
188                 Err(DbError::FailDbAccess(e)) => Err(TodoError::FailDbAccess(e)),
189                 Err(e) => {
190                     unreachable!("[Todo::is_valid_session]update_session:[{e}]")
191                 }
192             }
193         } else {
194             Ok(None)
195         }
196     }
197 }
198
199 #[derive(Error, Debug)]
200 pub enum TodoError {
201     #[error("FailInitDatabase")]
202     DbInit(sqlx::Error),
203     #[error("DuplicateUserName")]
204     DuplicateUser(sqlx::Error),
205     #[error("InvalidPassword:{0}")]
206     HashUserPassword(#[from] bcrypt::BcryptError),
207     #[error("NotFoundUser")]
208     NotFoundUser,
209     #[error("WrongPassword")]
210     WrongPassword,
211     #[error("NotFoundSession")]
212     NotFoundSession,
213     #[error("NotFoundTodo")]
214     NotFoundTodo,
215     #[error("DatabaseError:{0}")]
216     FailDbAccess(sqlx::Error),
217 }
218
219 impl From<TodoError> for String {
220     fn from(value: TodoError) -> Self {
221         value.to_string()
222     }
223 }
224
225 #[cfg(test)]
226 mod test {
227     use super::*;
228     use sqlx::MySqlPool;
```

```
229
230     impl Todo {
231         fn test_new(pool: MySqlPool) -> Self {
232             Self {
233                 database: Database::new_test(pool),
234             }
235         }
236     }
237
238     #[sqlx::test]
239     async fn new_user_and_login(pool: MySqlPool) {
240         let todo = Todo::test_new(pool);
241         // ユーザー生成
242         let user_name = "testdayo";
243         let user_pass = "passnano";
244         todo.add_user(user_name, user_pass).await.unwrap();
245
246         // 正しいユーザーでログイン
247         let _sess = todo.login(user_name, user_pass).await.unwrap();
248
249         // 間違ったユーザー名でログイン
250         let res = todo.login("detarame", user_pass).await;
251         match res {
252             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?"),
253             Err(TodoError::NotFoundUser) => {}
254             Err(e) => unreachable!("おなしなエラーが帰ってきた。{e}"),
255         }
256
257         // 間違ったパスワードでログイン
258         let res = todo.login(user_name, "detarame").await;
259         match res {
260             Ok(_) => unreachable!("間違ったパスワードでログインできちゃだめ"),
261             Err(TodoError::WrongPassword) => {}
262             Err(e) => unreachable!("こんなえらーだめです。{e}"),
263         }
264     }
265
266     #[sqlx::test]
267     async fn is_valid_session_test(pool: MySqlPool) {
268         let todo = Todo::test_new(pool);
269
270         // テスト用ユーザーの生成及び、ログイン
271         let user_name = "testdayo";
272         let user_pass = "passwordnano";
273
274         todo.add_user(user_name, user_pass).await.unwrap();
```

```
        let sess = todo.login(user_name, user_pass).await.unwrap();

        // 正しいセッションを検索する。
        let new_sess = todo.is_valid_session(&sess).await.unwrap();
        match new_sess {
            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。"),
            None => unreachable!("正しいセッションが見つからなかった。"),
        };

        // 間違ったセッションを検索する。
        let none_sess = todo.is_valid_session(&Uuid::now_v7()).await.unwrap();
        if none_sess.is_some() {
            unreachable!("こんなセッションがあるわけがない。");
        }
    }

    #[sqlx::test]
    async fn add_todo_test(pool: MySqlPool) {
        use chrono::Days;

        let todo = Todo::test_new(pool);
        let sess = login_for_test(&todo).await;

        let item1 = ItemTodo {
            id: 100,
            user_name: "kore_naihazu".to_string(),
            title: "テストアイテム１件目".to_string(),
            work: Some("これは、中身を入れる。".to_string()),
            update_date: None,
            start_date: Some(Local::now().date_naive() - Days::new(1)),
            end_date: Some(Local::now().date_naive() + Days::new(5)),
            done: true,
        };
        let item2 = 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: true,
        };
        let item3 = ItemTodo {
            id: 100,
            user_name: "kore_naihazu".to_string(),
```

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

    todo.add_todo(sess, &item2)
        .await
        .expect("二件目の追加に失敗");
    let res = todo
        .get_todo_list(sess, true, ItemSortOrder::EndAsc)
        .await
        .expect("二件目の取得に失敗");
    assert_eq!(res.len(), 2, "二件あるはずなんだけど");
    assert!(
        res.iter()
            .find(|&x| match x.title.find("work=null") {
                Some(n) => n > 0,
                None => false,
            })
            .expect("二件目に追加したデータがない")
            .work
            .is_none(),
        "二件目の work は None のはず"
    );
```

```
367         todo.add_todo(sess, &item3)
368             .await
369             .expect("三件目の追加に失敗");
370         let res = todo
371             .get_todo_list(sess, true, ItemSortOrder::EndAsc)
372             .await
373             .expect("三件目の取得に失敗");
374         assert_eq!(res.len(), 3, "三件あるはずですよ。");
375         assert!(
376             res.iter()
377                 .find(|&x| match x.title.find("work=space") {
378                     Some(n) => n > 0,
379                     None => false,
380                 })
381                 .expect("三件目のデータがないよ?")
382                 .work
383                 .is_none(),
384             "三件目のデータは None に変換してくれてるはず。"
385         );
386     }
387
388     #[sqlx::test]
389     async fn change_done_test(pool: MySqlPool) {
390         let todo = Todo::test_new(pool);
391         let sess = login_for_test(&todo).await;
392         create_todo_for_test(&todo, sess).await;
393
394         let items = todo
395             .get_todo_list(sess, true, ItemSortOrder::EndAsc)
396             .await
397             .unwrap();
398         let item = items
399             .iter()
400             .find(|&i| i.title.contains("1 件目"))
401             .expect("「1 件目」を含むアイテムは必ずあるはず");
402         assert!(!item.done, "まだ、未完了のはずです。");
403         let id = item.id;
404         todo.change_done(id, sess, true)
405             .await
406             .expect("状態更新に失敗。あってはならない。");
407         let items = todo
408             .get_todo_list(sess, true, ItemSortOrder::EndAsc)
409             .await
410             .unwrap();
411         assert_eq!(
412             items.len(),
```

```
            2,
            "一件完了済みにしたので、このリストは 2 件しかない。"
        );
        let items = todo
            .get_todo_list(sess, false, ItemSortOrder::EndAsc)
            .await
            .unwrap();
        assert_eq!(items.len(), 3, "完了済みを含むので、3 件になる。");
        let item = items
            .iter()
            .find(|&i| i.id == id)
            .expect("さっきあった id だから必ずある。");
        assert!(item.done, "さっき完了済みに変更した。");

        let max_id = items.iter().max_by_key(|&x| x.id).unwrap().id;
        let res = todo.change_done(max_id + 1, sess, false).await;
        match res {
            Ok(_) => unreachable!("この id の todo があるはずがない。"),
            Err(TodoError::NotFoundTodo) => {}
            Err(e) => unreachable!("このエラーもありえない。[{e}]"),
        };

        // 間違ったセッションのテスト
        let res = todo.change_done(id, Uuid::now_v7(), true).await;
        match res {
            Ok(_) => unreachable!("このセッションでは、更新を許してはいけない。"),
            Err(TodoError::NotFoundTodo) => { /* 正常 */ }
            Err(e) => unreachable!("このエラーもおかしい。[{e}]"),
        }
    }

    #[sqlx::test]
    async fn edit_todo_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, false, ItemSortOrder::EndAsc)
            .await
            .unwrap();
        let mut item = items
            .iter()
            .find(|&i| i.title.contains("1 件目"))
            .unwrap()
            .clone();
```

```
459             item.title = "更新した一件目".to_string();
460             if let Err(e) = todo.edit_todo(&item, sess).await {
461                 unreachable!("更新処理に失敗した。[{e}]");
462             }
463             let Some(item_new) = todo
464                 .get_todo_list(sess, false, ItemSortOrder::EndAsc)
465                 .await
466                 .unwrap()
467                 .iter()
468                 .find(|&i| i.title.contains("更新した一件目"))
469                 .cloned()
470             else {
471                 unreachable!("更新したレコードが見つからないよ?");
472             };
473             assert_eq!(item.id, item_new.id, "更新したレコードの id が化けてる");
474 
475             // ニセセッションで試す
476             match todo.edit_todo(&item, Uuid::now_v7()).await {
477                 Ok(_) => unreachable!("偽のセッションで更新成功してはならない。"),
478                 Err(TodoError::NotFoundTodo) => { /* 正常 */ }
479                 Err(e) => unreachable!("偽セッションのときのエラー:{e}"),
480             }
481         }
482 
483         async fn login_for_test(todo: &Todo) -> Uuid {
484             let user_name = "testdayo";
485             let user_pass = "passrordnona";
486             todo.add_user(user_name, user_pass).await.unwrap();
487             todo.login(user_name, user_pass).await.unwrap()
488         }
489 
490         async fn create_todo_for_test(todo: &Todo, sess: Uuid) {
491             use chrono::Days;
492             let items = [
493                 ItemTodo {
494                     id: 100,
495                     user_name: "kore_naihazu".to_string(),
496                     title: "テストアイテム 1 件目".to_string(),
497                     work: Some("これは、中身を入れる。".to_string()),
498                     update_date: None,
499                     start_date: Some(Local::now().date_naive() - Days::new(1)),
500                     end_date: Some(Local::now().date_naive() + Days::new(5)),
501                     done: false,
502                 },
503                 ItemTodo {
504                     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 database.rs

```rust
//! データベースの操作を司る

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

use crate::config::ItemSortOrder;

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

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

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

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

    /// Todo の一覧を取得する。
    /// 基準日 (ref_date) 以降のアイテムを選別する。
    /// セッション ID を必要とする。
    /// 検索オプションのとり方は未確定。インターフェース変更の可能性大。
    pub async fn get_todo_item(
        &self,
        sess: Uuid,
        ref_date: NaiveDate,
        only_incomplete: bool,
        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)
```

```
91              .bind(sess.to_string())
92              .bind(ref_date)
93              .fetch_all(&self.pool)
94              .await
95              .map_err(DbError::FailDbAccess)?;
96
97          Ok(items)
98      }
99
100     /// 指定 id の Todo 項目を取得する。
101     /// 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。
102     pub async fn get_todo_item_with_id(&self, id: u32, sess: Uuid) -> Result<ItemTodo, DbError> {
103         let sql = r#"
104             select t.id, t.user_name, t.title, t.work, t.update_date, t.start_date, t.end_date,
                ↪  t.done
105             from todo t join sessions s on s.user_name = t.user_name
106             where s.id=? and t.id=?
107             "#;
108         query_as::<_, ItemTodo>(sql)
109             .bind(sess.to_string())
110             .bind(id)
111             .fetch_one(&self.pool)
112             .await
113             .map_err(|e| match e {
114                 sqlx::Error::RowNotFound => DbError::NotFoundTodo,
115                 e => DbError::FailDbAccess(e),
116             })
117     }
118
119     /// Todo の完了状態を更新する。
120     pub async fn change_done(&self, id: u32, done: bool) -> Result<(), DbError> {
121         let sql = "update todo set done = ? where id = ?";
122         let res = query(sql)
123             .bind(done)
124             .bind(id)
125             .execute(&self.pool)
126             .await
127             .map_err(DbError::FailDbAccess)?;
128         if res.rows_affected() > 0 {
129             Ok(())
130         } else {
131             Err(DbError::NotFoundTodo)
132         }
133     }
134
135     /// Todo の項目編集
```

```
136     pub async fn edit_todo(&self, item: &ItemTodo) -> Result<(), DbError> {
137         let start_date = item.start_date.unwrap_or(Local::now().date_naive());
138         let end_date = item
139             .end_date
140             .unwrap_or(NaiveDate::from_ymd_opt(9999, 12, 31).unwrap());
141 
142         let sql = r#"
143             update todo
144             set title=?, work=?, update_date=curdate(), start_date=?, end_date=?
145             where id=?;
146             "#;
147         let res = query(sql)
148             .bind(&item.title)
149             .bind(&item.work)
150             .bind(start_date)
151             .bind(end_date)
152             .bind(item.id)
153             .execute(&self.pool)
154             .await
155             .map_err(DbError::FailDbAccess)?;
156         if res.rows_affected() > 0 {
157             Ok(())
158         } else {
159             Err(DbError::NotFoundTodo)
160         }
161     }
162 
163     /// ユーザーの追加
164     pub async fn add_user(&self, name: &str, pass: &str) -> Result<(), DbError> {
165         let sql = "insert into users(name, password) values (?, ?);";
166         query(sql)
167             .bind(name)
168             .bind(pass)
169             .execute(&self.pool)
170             .await
171             .map_err(|e| match e {
172                 sqlx::Error::Database(ref db_err) => {
173                     if db_err.kind() == sqlx::error::ErrorKind::UniqueViolation {
174                         DbError::DuplicateUserName(e)
175                     } else {
176                         DbError::FailDbAccess(e)
177                     }
178                 }
179                 _ => DbError::FailDbAccess(e),
180             })?;
181         Ok(())
```

```
182     }
183
184     /// ユーザー名をキーとして、ユーザー情報を取得
185     pub async fn get_user(&self, name: &str) -> Result<User, DbError> {
186         let sql = "select name, password from users where name = ?;";
187         query_as(sql)
188             .bind(name)
189             .fetch_one(&self.pool)
190             .await
191             .map_err(|e| match e {
192                 sqlx::Error::RowNotFound => DbError::NotFoundUser,
193                 e => DbError::FailDbAccess(e),
194             })
195     }
196
197     /// セッション ID をキーにしてユーザー情報を取得
198     pub async fn get_user_from_sess(&self, sess: Uuid) -> Result<User, DbError> {
199         let sql = r#"
200             select u.name, u.password
201             from users u join sessions s on u.name=s.user_name
202             where s.id = ?;
203             "#;
204
205         query_as(sql)
206             .bind(sess.to_string())
207             .fetch_one(&self.pool)
208             .await
209             .map_err(|e| match e {
210                 sqlx::Error::RowNotFound => DbError::NotFoundSession,
211                 e => DbError::FailDbAccess(e),
212             })
213     }
214
215     /// セッション情報を新規作成する。
216     ///   生成した uuid を返す。
217     pub async fn make_new_session(&self, user_name: &str) -> Result<Uuid, DbError> {
218         let sql = "insert into sessions(id, user_name) values (?,?);";
219         // キー情報の作成
220         let id = Uuid::now_v7();
221
222         query(sql)
223             .bind(id.to_string())
224             .bind(user_name)
225             .execute(&self.pool)
226             .await
227             .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 (?, ?);";
```

```
274             let id = Uuid::now_v7();
275             query(sql_create_sess)
276                 .bind(id.to_string())
277                 .bind(user)
278                 .execute(&mut *tr)
279                 .await
280                 .map_err(DbError::FailDbAccess)?;
281 
282             tr.commit().await.map_err(DbError::FailDbAccess)?;
283             Ok(id)
284         }
285 
286         /// 指定されたセッション ID が有効であるか確認する。
287         /// データベースエラーが発生した場合は、Err(DbError::FailDbAccess) を返す。
288         pub async fn is_session_valid(&self, sess: &Uuid) -> Result<bool, DbError> {
289             // 期限切れのセッションを削除する。
290             let sql_old_del = "delete from sessions where expired < now();";
291             query(sql_old_del)
292                 .execute(&self.pool)
293                 .await
294                 .map_err(DbError::FailDbAccess)?;
295             // 指定セッション ID の有無を確認する。
296             let sql_find_sess = "select count(*) as cnt from sessions where id = ?;";
297             let sess_cnt: i64 = query(sql_find_sess)
298                 .bind(sess.to_string())
299                 .fetch_one(&self.pool)
300                 .await
301                 .map_err(DbError::FailDbAccess)?
302                 .get("cnt");
303             if sess_cnt == 1 {
304                 Ok(true)
305             } else {
306                 Ok(false)
307             }
308         }
309 }
310 
311 #[derive(FromRow, Debug, PartialEq)]
312 pub struct User {
313     pub name: String,
314     pub password: String,
315 }
316 
317 #[derive(FromRow, Serialize, Deserialize, Debug, PartialEq, Clone)]
318 pub struct ItemTodo {
319     pub id: u32,
```

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

```
366         let error_user = db.get_user("naiyo").await;
367         match error_user {
368             Ok(_) => unreachable!("結果が帰ってくるはずがない。"),
369             Err(DbError::NotFoundUser) => { /* 正常 */ }
370             Err(e) => unreachable!("このエラーはおかしい。{e}"),
371         }
372     }
373
374     /// セッション生成関係の一連のテスト。
375     #[sqlx::test]
376     async fn test_make_new_session(pool: MySqlPool) {
377         println!("まずはテスト用のユーザーの生成");
378         let db = Database::new_test(pool);
379         let user_name = "nekodayo";
380         let password = "password";
381         db.add_user(user_name, password).await.unwrap();
382
383         println!("次に、普通にセッションを作ってみる。");
384         let sess1 = db.make_new_session(user_name).await.unwrap();
385         println!("セッション生成成功 id=[{}]", sess1);
386
387         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。");
388         let sess2 = db.make_new_session("detarame").await;
389         match sess2 {
390             Ok(_) => unreachable!("このユーザーは存在しなかったはず。"),
391             Err(DbError::NotFoundUser) => { /* 正常 */ }
392             Err(e) => unreachable!("このエラーもおかしい。[{}]", e),
393         }
394
395         println!("普通に、セッションを更新してみる。");
396         let sess3 = db.update_session(&sess1).await.unwrap();
397         assert_ne!(sess1, sess3);
398
399         println!("ないはずのセッションを更新しようとしてみる。");
400         let sess4 = Uuid::now_v7();
401         let sess5 = db.update_session(&sess4).await;
402         match sess5 {
403             Ok(_) => unreachable!("このセッションはないはずなのに。"),
404             Err(DbError::NotFoundSession) => { /* 正常 */ }
405             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}"),
406         }
407     }
408
409     /// セッションが有効かどうかを確認するテスト
410     #[sqlx::test]
411     async fn test_is_session_valid(pool: MySqlPool) {
```

```
412         let db = Database::new_test(pool);
413
414         println!("テスト用ユーザーの作成");
415         let name = "nekodayo";
416         let pass = "nekodamon";
417         db.add_user(name, pass).await.unwrap();
418
419         println!("新規セッションを生成する。");
420         let sess = db.make_new_session(name).await.unwrap();
421         println!("生成したセッション ID は、[{}] です。", &sess);
422
423         println!("今作ったセッション ID の妥当性を問い合わせてみる。");
424         assert!(db.is_session_valid(&sess).await.unwrap());
425
426         println!("偽セッション ID をいれて、問い合わせてみる。");
427         assert!(!db.is_session_valid(&Uuid::now_v7()).await.unwrap());
428     }
429
430     /// todo の書き込みと、単純な読み出しのテスト
431     #[sqlx::test]
432     async fn test_add_todo(pool: MySqlPool) {
433         let db = Database::new_test(pool);
434         let sess = login_for_test(&db).await;
435         let name = db.get_user_from_sess(sess).await.unwrap().name;
436
437         println!("テストデータをインサート");
438         let mut item = ItemTodo {
439             id: 0,
440             user_name: name.to_string(),
441             title: "インサートできるかな?".to_string(),
442             work: Some("中身入り".to_string()),
443             update_date: None,
444             start_date: Some(Local::now().date_naive()),
445             end_date: Some(Local::now().date_naive() + Days::new(3)),
446             done: true,
447         };
448         db.add_todo_item(&item).await.unwrap();
449
450         println!("テストデータを読み出す。一件しかないはず");
451         let last_day = Local::now().date_naive() + Days::new(1);
452         let res = db
453             .get_todo_item(sess, last_day, true, ItemSortOrder::EndAsc)
454             .await
455             .unwrap();
456         assert_eq!(res.len(), 1, "あれ?一件のはずだよ");
457         item.id = res[0].id;
```

```
458         item.update_date = Some(Local::now().date_naive());
459         item.done = false;
460     }
461 
462     /// todo の書き込みと読み出し。
463     /// work が未入力の場合。
464     #[sqlx::test]
465     async fn test_add_todo_without_work(pool: MySqlPool) {
466         let db = Database::new_test(pool);
467         let sess = login_for_test(&db).await;
468         let name = db.get_user_from_sess(sess).await.unwrap().name;
469 
470         println!("テストデータをインサート");
471         let mut item = ItemTodo {
472             id: 0,
473             user_name: name.to_string(),
474             title: "インサートできるかな?".to_string(),
475             work: None,
476             update_date: None,
477             start_date: Some(Local::now().date_naive()),
478             end_date: Some(Local::now().date_naive() + Days::new(3)),
479             done: true,
480         };
481         db.add_todo_item(&item).await.unwrap();
482 
483         println!("テストデータを読み出す。一件しかないはず");
484         let last_day = Local::now().date_naive() + Days::new(1);
485         let res = db
486             .get_todo_item(sess, last_day, true, ItemSortOrder::EndAsc)
487             .await
488             .unwrap();
489         assert_eq!(res.len(), 1, "あれ?一件のはずだよ");
490         item.id = res[0].id;
491         item.update_date = Some(Local::now().date_naive());
492         item.done = false;
493     }
494 
495     /// todo の書き込みと読み出し
496     /// done=true と false の挙動テスト
497     #[sqlx::test]
498     async fn test_get_todo_done_param(pool: MySqlPool) {
499         let db = Database::new_test(pool.clone());
500         let sess = login_for_test(&db).await;
501         let name = db.get_user_from_sess(sess).await.unwrap().name;
502 
503         println!("テストデータをインサート");
```

```
504         let item = ItemTodo {
505             id: 0,
506             user_name: name.to_string(),
507             title: "インサートできるかな?".to_string(),
508             work: None,
509             update_date: None,
510             start_date: Some(Local::now().date_naive()),
511             end_date: Some(Local::now().date_naive() + Days::new(3)),
512             done: true,
513         };
514         db.add_todo_item(&item).await.unwrap();
515 
516         println!("テストデータを読み出す。一件しかないはず");
517         let last_day = Local::now().date_naive() + Days::new(1);
518         let res = db
519             .get_todo_item(sess, last_day, false, ItemSortOrder::EndAsc)
520             .await
521             .unwrap();
522         assert_eq!(res.len(), 1, "全部読み出しだけど一件あるはず。");
523         let res = db
524             .get_todo_item(sess, last_day, true, ItemSortOrder::EndAsc)
525             .await
526             .unwrap();
527         assert_eq!(res.len(), 1, "未完了だけだけど、一件あるはず。");
528 
529         println!("今作った job を完了済みにする。");
530         let sql = "update todo set done=true where id=?;";
531         query(sql).bind(res[0].id).execute(&pool).await.unwrap();
532         let res = db
533             .get_todo_item(sess, last_day, false, ItemSortOrder::EndAsc)
534             .await
535             .unwrap();
536         assert_eq!(res.len(), 1, "全部読み出しだけど一件あるはず。");
537         let res = db
538             .get_todo_item(sess, last_day, true, ItemSortOrder::EndAsc)
539             .await
540             .unwrap();
541         assert_eq!(res.len(), 0, "未完了だけだけだから、なにもないはず。");
542     }
543 
544     /// todo の書き込みと読み出し
545     /// 基準日の挙動テスト
546     #[sqlx::test]
547     async fn test_get_todo_ref_date(pool: MySqlPool) {
548         let db = Database::new_test(pool.clone());
549         let sess = login_for_test(&db).await;
```

```
550         let name = db.get_user_from_sess(sess).await.unwrap().name;
551
552         println!("テストデータをインサート");
553         let item = ItemTodo {
554             id: 0,
555             user_name: name.to_string(),
556             title: "インサートできるかな?".to_string(),
557             work: None,
558             update_date: None,
559             start_date: Some(Local::now().date_naive()),
560             end_date: Some(Local::now().date_naive() + Days::new(3)),
561             done: false,
562         };
563         db.add_todo_item(&item).await.unwrap();
564
565         let ref_date = Local::now().date_naive();
566         let res = db
567             .get_todo_item(sess, ref_date, true, ItemSortOrder::EndAsc)
568             .await
569             .unwrap();
570         assert_eq!(res.len(), 1, "基準日と開始日が同じだからみつかる。");
571         let res = db
572             .get_todo_item(sess, ref_date + Days::new(1), true, ItemSortOrder::EndAsc)
573             .await
574             .unwrap();
575         assert_eq!(res.len(), 1, "開始日の翌日が基準日だからみつかる。");
576         let res = db
577             .get_todo_item(sess, ref_date - Days::new(1), true, ItemSortOrder::EndAsc)
578             .await
579             .unwrap();
580         assert_eq!(res.len(), 0, "基準日が開始日の前日だからみつからない。");
581         let res = db
582             .get_todo_item(sess, ref_date + Days::new(4), true, ItemSortOrder::EndAsc)
583             .await
584             .unwrap();
585         assert_eq!(res.len(), 1, "基準日が期限を過ぎているけどみつかるの。");
586     }
587
588     #[sqlx::test]
589     async fn test_get_user_from_sess(pool: MySqlPool) {
590         let db = Database::new_test(pool.clone());
591
592         let sess = login_for_test(&db).await;
593         let name = db.get_user_from_sess(sess).await.unwrap().name;
594
595         let user = db.get_user_from_sess(sess).await.unwrap();
```

```
596         assert_eq!(user.name, name, "これはみつかるはず");
597         let dummy_sess = Uuid::now_v7();
598         let user = db.get_user_from_sess(dummy_sess).await;
599         match user {
600             Ok(_) => unreachable!("見つかるわけないでしょう。"),
601             Err(DbError::NotFoundSession) => { /* 正常 */ }
602             Err(e) => unreachable!("トラブルです。{e}"),
603         };
604     }
605
606     #[sqlx::test]
607     async fn test_change_done(pool: MySqlPool) {
608         let db = Database::new_test(pool);
609         let sess = login_for_test(&db).await;
610         let ref_date = Local::now().date_naive();
611         create_todo_for_test(&db, sess).await;
612
613         let items = db
614             .get_todo_item(sess, ref_date, true, ItemSortOrder::EndAsc)
615             .await
616             .unwrap();
617         let item = items.iter().find(|&i| i.title.contains("二件目")).unwrap();
618         db.change_done(item.id, true).await.unwrap();
619
620         let items = db
621             .get_todo_item(sess, ref_date, true, ItemSortOrder::EndAsc)
622             .await
623             .unwrap();
624         let item = items.iter().find(|&i| i.title.contains("二件目"));
625         assert!(item.is_none(), "状態を完了にしたので見つからないはず。");
626
627         let items = db
628             .get_todo_item(sess, ref_date, false, ItemSortOrder::EndAsc)
629             .await
630             .unwrap();
631         let item = items.iter().find(|&i| i.title.contains("二件目"));
632         match item {
633             Some(i) => assert!(i.done, "完了済みになっているはずですね?"),
634             None => unreachable!("状態を変えたら、レコードなくなった???"),
635         }
636         assert_eq!(
637             items.len(),
638             3,
639             "全件見ているのでレコードは３件あるはずですが?"
640         );
641     }
```

```rust
642
643     #[sqlx::test]
644     async fn test_get_todo_with_id(pool: MySqlPool) {
645         let db = Database::new_test(pool);
646         let sess = login_for_test(&db).await;
647         create_todo_for_test(&db, sess).await;
648
649         let items = db
650             .get_todo_item(
651                 sess,
652                 Local::now().date_naive(),
653                 false,
654                 ItemSortOrder::EndAsc,
655             )
656             .await
657             .unwrap();
658         let id = items
659             .iter()
660             .find(|&i| i.title.contains("一件目"))
661             .expect("これはあるはず")
662             .id;
663         let non_exist_id = items.iter().max_by_key(|&i| i.id).unwrap().id + 1;
664
665         // 正常な読み出し
666         let res = db
667             .get_todo_item_with_id(id, sess)
668             .await
669             .expect("これは正常に読み出せるはず。エラーはだめ");
670         res.work
671             .expect("このレーコードは work を持つはずです。")
672             .find("働いてます。")
673             .expect("work の内容がおかしい。");
674
675         // 間違った id
676         let res = db.get_todo_item_with_id(non_exist_id, sess).await;
677         match res {
678             Ok(_) => unreachable!("そんな ID は存在しなかったはずなのに。"),
679             Err(DbError::NotFoundTodo) => { /* 正常 */ }
680             Err(e) => unreachable!("データベースエラーだよ。({e})"),
681         }
682
683         // 間違ったセッション
684         let res = db.get_todo_item_with_id(id, Uuid::now_v7()).await;
685         match res {
686             Ok(_) => unreachable!("そんなセッションはないはず。"),
687             Err(DbError::NotFoundTodo) => { /* 正常 */ }
```

```
688             Err(e) => unreachable!("データベースエラー発生。({e})"),
689         }
690     }
691
692     #[sqlx::test]
693     async fn test_edit(pool: MySqlPool) {
694         let db = Database::new_test(pool);
695         let sess = login_for_test(&db).await;
696         create_todo_for_test(&db, sess).await;
697
698         // 書き込みテスト用レコードの取得
699         let today = Local::now().date_naive();
700         let items = db
701             .get_todo_item(sess, today, false, ItemSortOrder::EndAsc)
702             .await
703             .unwrap();
704         let mut item = items
705             .iter()
706             .find(|&i| i.title.contains("一件目"))
707             .expect("ないはずがない。")
708             .clone();
709         item.title = "更新しました。".to_string();
710         item.work = Some("書き換え後".to_string());
711         item.start_date = Some(today - Days::new(5));
712         item.end_date = Some(today + Days::new(10));
713         db.edit_todo(&item).await.expect("更新がエラーを起こした。");
714         // 書き込み後の照合
715         let items_new = db
716             .get_todo_item(sess, today, false, ItemSortOrder::EndAsc)
717             .await
718             .unwrap();
719         let item_new = items_new
720             .iter()
721             .find(|&i| i.title.contains("更新しました。"))
722             .expect("更新されたレコードが存在しない。");
723         assert_eq!(
724             item_new.work,
725             Some("書き換え後".to_string()),
726             "更新後の work がおかしい"
727         );
728         assert_eq!(
729             item_new.start_date,
730             Some(today - Days::new(5)),
731             "更新後の start_date がおかしい"
732         );
733         assert_eq!(
```

```
734             item_new.end_date,
735             Some(today + Days::new(10)),
736             "更新後の end_date がおかしい"
737         );
738 
739         // 存在しないレコードの更新
740         let id_max_plus_one = items.iter().max_by_key(|&i| i.id).unwrap().id + 1;
741         item.id = id_max_plus_one;
742         let res = db.edit_todo(&item).await;
743         match res {
744             Ok(_) => unreachable!("更新できちゃだめっ"),
745             Err(DbError::NotFoundTodo) => {}
746             Err(e) => unreachable!("db_err: {e}"),
747         }
748     }
749 
750     #[sqlx::test]
751     async fn test_sort_end_date(pool: MySqlPool) {
752         let db = Database::new_test(pool);
753         let sess = login_for_test(&db).await;
754         create_todo_for_test(&db, sess).await;
755 
756         let today = Local::now().date_naive();
757         let recs = db
758             .get_todo_item(sess, today, false, ItemSortOrder::EndAsc)
759             .await
760             .expect("取得時にエラーを起こした。");
761         eprintln!("取得データ（昇順）");
762         eprintln!("0 => {:?}", recs[0]);
763         eprintln!("1 => {:?}", recs[1]);
764         eprintln!("2 => {:?}", recs[2]);
765         assert!(
766             recs[0].end_date <= recs[1].end_date,
767             "終了日が昇順になってない。"
768         );
769         assert!(
770             recs[1].end_date <= recs[2].end_date,
771             "終了日が昇順になってない（2）。"
772         );
773 
774         let recs = db
775             .get_todo_item(sess, today, false, ItemSortOrder::EndDesc)
776             .await
777             .expect("取得時にエラーを起こした（2）");
778         eprintln!("取得データ（降順）");
779         eprintln!("0 => {:?}", recs[0]);
```

```
780         eprintln!("1 => {:?}", recs[1]);
781         eprintln!("2 => {:?}", recs[2]);
782         assert!(
783             recs[0].end_date >= recs[1].end_date,
784             "終了日が降順になってない (1)"
785         );
786         assert!(
787             recs[1].end_date >= recs[2].end_date,
788             "終了日が降順になってない (2)"
789         );
790     }
791
792     #[sqlx::test]
793     async fn test_sort_start_date(pool: MySqlPool) {
794         let db = Database::new_test(pool);
795         let sess = login_for_test(&db).await;
796         create_todo_for_test(&db, sess).await;
797
798         let today = Local::now().date_naive();
799         let recs = db
800             .get_todo_item(sess, today, false, ItemSortOrder::StartAsc)
801             .await
802             .expect("取得時にエラーを起こした。");
803         assert!(
804             recs[0].start_date <= recs[1].start_date,
805             "開始日が昇順になってない。"
806         );
807         assert!(
808             recs[1].start_date <= recs[2].start_date,
809             "開始日が昇順になってない (2)。"
810         );
811
812         let recs = db
813             .get_todo_item(sess, today, false, ItemSortOrder::StartDesc)
814             .await
815             .expect("取得時にエラーを起こした (2)");
816         assert!(
817             recs[0].start_date >= recs[1].start_date,
818             "開始日が降順になってない (1)"
819         );
820         assert!(
821             recs[1].start_date >= recs[2].start_date,
822             "開始日が降順になってない (2)"
823         );
824     }
825
```

```
826     #[sqlx::test]
827     async fn test_sort_update_date(pool: MySqlPool) {
828         let db = Database::new_test(pool);
829         let sess = login_for_test(&db).await;
830         create_todo_for_test(&db, sess).await;
831         let today = Local::now().date_naive();
832 
833         // 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
834         let keys = db
835             .get_todo_item(sess, today, false, ItemSortOrder::EndAsc)
836             .await
837             .unwrap()
838             .iter()
839             .map(|r| r.id)
840             .collect::<Vec<_>>();
841         let sql = "update todo set update_date = ? where id = ?";
842         let days = vec![
843             today + Days::new(2),
844             today + Days::new(1),
845             today + Days::new(3),
846         ];
847         for i in 0..3 {
848             query(sql)
849                 .bind(days[i])
850                 .bind(keys[i])
851                 .execute(&db.pool)
852                 .await
853                 .unwrap();
854         }
855 
856         let recs = db
857             .get_todo_item(sess, today, false, ItemSortOrder::UpdateAsc)
858             .await
859             .expect("取得時にエラーを起こした。");
860         assert!(
861             recs[0].update_date <= recs[1].update_date,
862             "更新日が昇順になってない。"
863         );
864         assert!(
865             recs[1].update_date <= recs[2].update_date,
866             "更新日が昇順になってない (2)。"
867         );
868 
869         let recs = db
870             .get_todo_item(sess, today, false, ItemSortOrder::UpdateDesc)
871             .await
```

```
872             .expect("取得時にエラーを起こした (2)");
873         assert!(
874             recs[0].update_date >= recs[1].update_date,
875             "更新日が降順になってない (1)"
876         );
877         assert!(
878             recs[1].update_date >= recs[2].update_date,
879             "更新日が降順になってない (2)"
880         );
881     }
882 
883     async fn login_for_test(db: &Database) -> Uuid {
884         println!("テスト用ユーザー及びセッションの生成");
885         let name = "test";
886         let pass = "test";
887         db.add_user(name, pass).await.unwrap();
888         db.make_new_session(name).await.unwrap()
889     }
890 
891     async fn create_todo_for_test(db: &Database, sess: Uuid) {
892         let name = db.get_user_from_sess(sess).await.unwrap().name;
893 
894         println!("テストデータをインサート");
895         let item = ItemTodo {
896             id: 0,
897             user_name: name.to_string(),
898             title: "一件目 (work 有り)".to_string(),
899             work: Some("働いてます。".to_string()),
900             update_date: None,
901             start_date: Some(Local::now().date_naive() - Days::new(4)),
902             end_date: Some(Local::now().date_naive() + Days::new(2)),
903             done: false,
904         };
905         db.add_todo_item(&item).await.unwrap();
906 
907         let item = ItemTodo {
908             id: 0,
909             user_name: name.to_string(),
910             title: "二件目 (work 無し)".to_string(),
911             work: None,
912             update_date: None,
913             start_date: Some(Local::now().date_naive() - Days::new(5)),
914             end_date: Some(Local::now().date_naive() + Days::new(1)),
915             done: false,
916         };
917         db.add_todo_item(&item).await.unwrap();
```

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

2　フロントエンド　React 関係

2.1　index.html

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

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

2.2 メイン CSS ファイル

```
.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

```
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"/>
```

```
                    <InputDate name="end" label="終了"/>
                    <HStack>
                        <Button type="submit" w="30%" ml="auto" mr="5%">送信</Button>
                        <Button w="30%" onClick={onCancelClick} mr="auto">キャンセル</Button>
                    </HStack>
                    <Text> {isPending ? "送信中です。" : null} </Text>
                    <Text> {errorMessage} </Text>
                </VStack>
            </FormProvider>
        </>
    );
}


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

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

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

```
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)
    );

```