neko_todo ソースリスト

美都

2025 年 1 月 18 日

目次

1　Rust ソース

1.1　メインモジュール　main.rs

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

use env_logger::builder;

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

    neko_todo_lib::run()
}
```

1.2 ライブラリメインモジュール lib.rs

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

use app_status::AppStatus;
use database::ItemTodo;
use log::error;
use serde::Deserialize;
use setup::setup;
use tauri::{command, State};
use todo::TodoError;
use uuid::Uuid;

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

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

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

```
        .run(tauri::generate_context!())
        .expect("error while running tauri application");
}

/// 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> {
    println!("todo 取得");
    let sess = match get_curr_session(&app_status) {
        Some(u) => u,
        None => return Err("NotLogin".to_string()),
    };

    app_status
        .todo()
        .get_todo_list(sess, true)
        .await
        .map_err(|e| e.to_string())
}

/// 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),
    };

    eprintln!("input = {:?}", &item);
    app_status
        .todo()
        .add_todo(sess, &item.into())
        .await
        .map_err(|e| match e {
            todo::TodoError::NotFoundSession => "NotFoundSession".to_string(),
            e => format!("OtherError:[{e}]"),
        })
}

/// 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,
        }
    }
}

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

/// ユーザー登録
#[tauri::command]
async fn regist_user(
    app_status: State<'_, AppStatus>,
    name: String,
    password: String,
```

```
) -> Result<(), String> {
    match app_status.todo().add_user(&name, &password).await {
        Ok(_) => Ok(()),
        Err(e) => match e {
            TodoError::DuplicateUser(_) => Err("DuplicateUser".to_string()),
            TodoError::HashUserPassword(e) => Err(format!("InvalidPassword.[{}]", e)),
            TodoError::FailDbAccess(e) => Err(e.to_string()),
            _ => unimplemented!("[lib.rs regist_user] add_user 返り値異常"),
        },
    }
}

/// 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
        .map_err(|e| match e {
            TodoError::NotFoundUser => "NotFoundUser.".to_string(),
            TodoError::HashUserPassword(e) => format!("InvalidPassword.[{}]", e),
            TodoError::WrongPassword => "WrongPassword".to_string(),
            TodoError::FailDbAccess(e) => format!("OtherError:{}", e),
            _ => unimplemented!("[lib.rs::login] login から予期しないエラー"),
        })?;

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

/// 現在、有効なセッションが存在するかどうか確認。(ユーザ I/F 用)
#[command]
async fn is_valid_session(app_status: State<'_, AppStatus>) -> Result<bool, String> {
    get_cur_session_with_update(&app_status)
        .await
        .map(|i| i.is_some())
}

```

```rust
/// 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()
}
```

1.3 アプリケーションステータス app_status.rs

```rust
//! アプリケーション全体のステータスを保持する。

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

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

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

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

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

1.4 コンフィグ設定処理 config.rs

```rust
//! アプリケーション設定の取得関係

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

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

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

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

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

```

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

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

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

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

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

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

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

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

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

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

```
134
135     /// 環境設定の挙動テスト
136     #[test]
137     #[ignore]
138     fn test_env_val() {
139         let val_db_host = "test_host";
140         let val_db_user = "test_user";
141         let val_db_pass = "test_pass";
142         save_curr_conf_file();
143         {
144             let mut conf = NekoTodoConfig::new().unwrap();
145             // 初期状態では空文字列が返るはず
146             assert_eq!(conf.get_db_host(), "");
147             assert_eq!(conf.get_db_user(), "");
148             assert_eq!(conf.get_db_pass(), "");
149             // test_host をセットしてセットされているか確認。
150             conf.set_db_host(val_db_host);
151             conf.set_db_user(val_db_user);
152             conf.set_db_pass(val_db_pass);
153             assert_eq!(conf.get_db_host(), val_db_host);
154             assert_eq!(conf.get_db_user(), val_db_user);
155             assert_eq!(conf.get_db_pass(), val_db_pass);
156         } // この時点で一旦環境ファイルを保存してみる。
157           // 環境ファイルをもう一度ロードして、環境を確認
158         delete_env_val();
159         let conf = NekoTodoConfig::new().unwrap();
160         assert_eq!(conf.get_db_host(), val_db_host);
161         assert_eq!(conf.get_db_user(), val_db_user);
162         assert_eq!(conf.get_db_pass(), val_db_pass);
163         restore_curr_conf_file();
164     }
165
166     /// テスト環境のため、元の conf ファイルを退避
167     fn save_curr_conf_file() {
168         let file = NekoTodoConfig::get_config_file_path().unwrap();
169         let mut save_file = file.clone();
170         save_file.set_extension("save");
171         if file.exists() {
172             println!(
173                 "現在の環境ファイル [{:?}] を [{:?}] に退避します。",
174                 &file, &save_file
175             );
176             std::fs::rename(file, save_file).unwrap();
177         }
178     }
```

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

1.5　アプリケーション設定情報の処理 setup.rs

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

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

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

    let conf = NekoTodoConfig::new()?;

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

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

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

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

```
44      };
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      eprintln!("次のパラメータを使用します。");
51      eprintln!("ホスト名:{}", host);
52      eprintln!("ユーザー名:{}", user);
53      eprintln!("パスワード:{}", pass);
54      eprintln!("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      eprintln!("データベースへの接続に成功しました。");
58      eprintln!("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
91  #[derive(Error, Debug)]
92  pub enum SetupError {
93      #[error("設定ファイルへのアクセスに失敗")]
94      SetupFile(#[from] dotenvy::Error),
95      #[error("--setup 時には、server,user,pass の設定が必須です")]
96      Argument,
97      #[error("データベースへの接続に失敗")]
98      ConnectDatabase(#[from] TodoError),
99  }
```

1.6 todo モデル処理 todo.rs

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

use crate::database::*;

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

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

    /// todo の一覧を取得する。(仮実装。インターフェース未確定)
    pub async fn get_todo_list(
        &self,
        sess: Uuid,
        only_imcomplete: bool,
    ) -> Result<Vec<ItemTodo>, TodoError> {
        let ref_date = Local::now().date_naive();
        self.database
            .get_todo_item(sess, ref_date, only_imcomplete)
            .await
            .map_err(|e| match e {
                DbError::FailDbAccess(e) => TodoError::FailDbAccess(e),
                e => unimplemented!("[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 => unimplemented!("[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 => unimplemented!("[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 => unimplemented!("[Todo::get_todo_with_id]get_todo_item_with_id[{e}]"),
```

```
89              })
90      }
91
92      /// Todo の完了状態を変更する
93      pub async fn change_done(&self, id: u32, sess: Uuid, done: bool) -> Result<(), TodoError> {
94          self.get_todo_with_id(id, sess).await?;
95          self.database
96              .change_done(id, done)
97              .await
98              .map_err(|e| match e {
99                  DbError::FailDbAccess(e) => {
100                     error!("[Todo::change_done]change_done:[{e}]");
101                     TodoError::FailDbAccess(e)
102                 }
103                 DbError::NotFoundTodo => TodoError::NotFoundTodo,
104                 e => unimplemented!("[change_done]change_done[{e}]"),
105             })
106     }
107
108     /// ユーザーの追加を行う。
109     pub async fn add_user(&self, name: &str, password: &str) -> Result<(), TodoError> {
110         let hashed_pass = hash(password, DEFAULT_COST)?;
111         if let Err(e) = self.database.add_user(name, &hashed_pass).await {
112             match e {
113                 DbError::DuplicateUserName(e) => return Err(TodoError::DuplicateUser(e)),
114                 DbError::FailDbAccess(e) => {
115                     error!("[Todo::add_user]Database::add_user:[{e}]");
116                     return Err(TodoError::FailDbAccess(e));
117                 }
118                 _ => {}
119             }
120         }
121         Ok(())
122     }
123
124     /// ログイン処理を行う。
125     pub async fn login(&self, name: &str, password: &str) -> Result<Uuid, TodoError> {
126         // 認証
127         let user = self.database.get_user(name).await.map_err(|e| match e {
128             DbError::NotFoundUser => TodoError::NotFoundUser,
129             DbError::FailDbAccess(e) => TodoError::FailDbAccess(e),
130             e => unimplemented!("[ToDo::login] Database::get_user:[{e}]"),
131         })?;
132         if !verify(password, &user.password)? {
133             return Err(TodoError::WrongPassword);
```

```
134         }
135         // セッションの生成
136         let session = self
137             .database
138             .make_new_session(&user.name)
139             .await
140             .map_err(|e| match e {
141                 DbError::NotFoundUser => TodoError::NotFoundUser,
142                 DbError::FailDbAccess(e) => TodoError::FailDbAccess(e),
143                 e => {
144                     unimplemented!("[Todo::login] Database::make_new_session:[{e}]")
145                 }
146             })?;
147         Ok(session)
148     }
149 
150     /// 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。
151     /// もし指定されたセッション ID が無効な場合は、None を返す。
152     /// 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。
153     pub async fn is_valid_session(&self, sess: &Uuid) -> Result<Option<Uuid>, TodoError> {
154         let is_valid = self
155             .database
156             .is_session_valid(sess)
157             .await
158             .map_err(|e| match e {
159                 DbError::FailDbAccess(e) => TodoError::FailDbAccess(e),
160                 e => {
161                     unimplemented!("[Todo::is_valid_session]is_session_valid:[{e}]")
162                 }
163             })?;
164         if is_valid {
165             match self.database.update_session(sess).await {
166                 Ok(s) => Ok(Some(s)),
167                 Err(DbError::NotFoundSession) => Ok(None),
168                 Err(DbError::FailDbAccess(e)) => Err(TodoError::FailDbAccess(e)),
169                 Err(e) => {
170                     unimplemented!("[Todo::is_valid_session]update_session:[{e}]")
171                 }
172             }
173         } else {
174             Ok(None)
175         }
176     }
177 }
178 
```

```rust
#[derive(Error, Debug)]
pub enum TodoError {
    #[error("データベース初期化失敗")]
    DbInit(sqlx::Error),
    #[error("すでに、このユーザー名は使用されています。")]
    DuplicateUser(sqlx::Error),
    #[error("ユーザーパスワードのハッシュに失敗。")]
    HashUserPassword(#[from] bcrypt::BcryptError),
    #[error("ユーザーが見つかりません。")]
    NotFoundUser,
    #[error("パスワードが違います。")]
    WrongPassword,
    #[error("セッションが見つかりません。")]
    NotFoundSession,
    #[error("指定 id の todo が見つかりません。")]
    NotFoundTodo,
    #[error("データベースアクセスに失敗")]
    FailDbAccess(sqlx::Error),
}

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

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

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

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

        // 間違ったユーザー名でログイン
```

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

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

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

        // テスト用ユーザーの生成及び、ログイン
        let user_name = "testdayo";
        let user_pass = "passwordnano";

        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 => assert!(false, "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();
        match none_sess {
            Some(_) => assert!(false, "こんなセッションがあるわけがない。"),
            None => {}
        }
    }

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

```
269
270         let todo = Todo::test_new(pool);
271         let sess = login_for_test(&todo).await;
272
273         let item1 = ItemTodo {
274             id: 100,
275             user_name: "kore_naihazu".to_string(),
276             title: "テストアイテム 1 件目".to_string(),
277             work: Some("これは、中身を入れる。".to_string()),
278             update_date: None,
279             start_date: Some(Local::now().date_naive() - Days::new(1)),
280             end_date: Some(Local::now().date_naive() + Days::new(5)),
281             done: true,
282         };
283         let item2 = ItemTodo {
284             id: 100,
285             user_name: "kore_naihazu".to_string(),
286             title: "テストアイテム 2 件目 (work=null)".to_string(),
287             work: Some("".to_string()),
288             update_date: None,
289             start_date: Some(Local::now().date_naive() - Days::new(1)),
290             end_date: Some(Local::now().date_naive() + Days::new(5)),
291             done: true,
292         };
293         let item3 = ItemTodo {
294             id: 100,
295             user_name: "kore_naihazu".to_string(),
296             title: "テストアイテム 3 件目 (work=space)".to_string(),
297             work: Some(" \t  ".to_string()),
298             update_date: None,
299             start_date: Some(Local::now().date_naive() - Days::new(1)),
300             end_date: Some(Local::now().date_naive() + Days::new(5)),
301             done: true,
302         };
303         todo.add_todo(sess, &item1)
304             .await
305             .expect("1 件目の追加に失敗");
306         let res = todo
307             .get_todo_list(sess, true)
308             .await
309             .expect("1 件目の取得に失敗");
310         assert_eq!(res.len(), 1, "一件目が取得できなかった?");
311         assert_eq!(res[0].title, item1.title, "一件目の title が違う");
312         assert_eq!(res[0].work, item1.work, "一件目の work が違う");
313         assert_eq!(res[0].user_name, "testdayo", "一件目の user_name が違う");
```

```
314         assert_eq!(
315             res[0].update_date,
316             Some(Local::now().date_naive()),
317             "一件目の update_date が違う"
318         );
319         assert_eq!(res[0].start_date, item1.start_date, "一件目の開始日が違う");
320         assert_eq!(res[0].end_date, item1.end_date, "一件目の終了日が違う");
321         assert_eq!(res[0].done, false, "一件目の完了マークが違う");
322 
323         todo.add_todo(sess, &item2)
324             .await
325             .expect("二件目の追加に失敗");
326         let res = todo
327             .get_todo_list(sess, true)
328             .await
329             .expect("二件目の取得に失敗");
330         assert_eq!(res.len(), 2, "二件あるはずなんだけど");
331         assert!(
332             res.iter()
333                 .find(|&x| match x.title.find("work=null") {
334                     Some(n) => n > 0,
335                     None => false,
336                 })
337                 .expect("二件目に追加したデータがない")
338                 .work
339                 .is_none(),
340             "二件目の work は None のはず"
341         );
342         todo.add_todo(sess, &item3)
343             .await
344             .expect("三件目の追加に失敗");
345         let res = todo
346             .get_todo_list(sess, true)
347             .await
348             .expect("三件目の取得に失敗");
349         assert_eq!(res.len(), 3, "三件あるはずですよ。");
350         assert!(
351             res.iter()
352                 .find(|&x| match x.title.find("work=space") {
353                     Some(n) => n > 0,
354                     None => false,
355                 })
356                 .expect("三件目のデータがないよ?")
357                 .work
358                 .is_none(),
```

```
            "三件目のデータは None に変換してくれてるはず。"
        );
    }

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

        let items = todo.get_todo_list(sess, true).await.unwrap();
        let item = items
            .iter()
            .find(|&i| i.title.find("1 件目").is_some())
            .expect("「1 件目」を含むアイテムは必ずあるはず");
        assert!(!item.done, "まだ、未完了のはずです。");
        let id = item.id;
        todo.change_done(id, sess, true)
            .await
            .expect("状態更新に失敗。あってはならない。");
        let items = todo.get_todo_list(sess, true).await.unwrap();
        assert_eq!(
            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).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_eq!(item.done, true, "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(_) => assert!(false, "この id の todo があるはずがない。"),
            Err(TodoError::NotFoundTodo) => {}
            Err(e) => assert!(false, "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(_) => assert!(false, "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) => assert!(false, "このエラーもおかしい。[{e}]"),
        }
    }

    async fn login_for_test(todo: &Todo) -> Uuid {
        let user_name = "testdayo";
        let user_pass = "passrordnona";
        todo.add_user(user_name, user_pass).await.unwrap();
        todo.login(user_name, user_pass).await.unwrap()
    }

    async fn create_todo_for_test(todo: &Todo, sess: Uuid) {
        use chrono::Days;
        let items = [
            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: false,
            },
            ItemTodo {
                id: 100,
                user_name: "kore_naihazu".to_string(),
                title: "テストアイテム 2 件目 (work=null)".to_string(),
                work: Some("".to_string()),
                update_date: None,
                start_date: Some(Local::now().date_naive() - Days::new(1)),
                end_date: Some(Local::now().date_naive() + Days::new(5)),
                done: false,
            },
            ItemTodo {
                id: 100,
                user_name: "kore_naihazu".to_string(),
                title: "テストアイテム 3 件目 (work=space)".to_string(),
                work: Some(" \t  ".to_string()),
                update_date: None,
                start_date: Some(Local::now().date_naive() - Days::new(1)),
                end_date: Some(Local::now().date_naive() + Days::new(5)),
                done: false,
```

```
            },
        ];
        for item in items {
            todo.add_todo(sess, &item).await.unwrap();
        }
    }
}
```

1.7 データベースアクセス database.rs

```rust
//! データベースの操作を司る

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

/// neko_db データベース操作関数群
#[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,
    ) -> Result<Vec<ItemTodo>, DbError> {
        let sql1 = r#"
            select t.id, t.user_name, title, work, update_date, start_date, end_date, done
            from todo t join sessions s on s.user_name = t.user_name
            where s.id=? and t.start_date <= ?
            "#;
        let sql2 = " and done = false";
        let sql = if only_incomplete {
            format!("{} {};", sql1, sql2)
        } else {
            format!("{} ;", sql1)
        };
        let items = query_as::<_, ItemTodo>(&sql)
            .bind(sess.to_string())
            .bind(ref_date)
            .fetch_all(&self.pool)
            .await
            .map_err(DbError::FailDbAccess)?;

        Ok(items)
    }

```

```rust
    /// 指定 id の Todo 項目を取得する。
    /// 有効なセッションが指定されていなければ、未発見とする。
    pub async fn get_todo_item_with_id(&self, id: u32, sess: Uuid) -> Result<ItemTodo, DbError> {
        let sql = r#"
            select t.id, t.user_name, t.title, t.work, t.update_date, t.start_date, t.end_date,
            ↪  t.done
            from todo t join sessions s on s.user_name = t.user_name
            where s.id=? and t.id=?
            "#;
        query_as::<_, ItemTodo>(sql)
            .bind(sess.to_string())
            .bind(id)
            .fetch_one(&self.pool)
            .await
            .map_err(|e| match e {
                sqlx::Error::RowNotFound => DbError::NotFoundTodo,
                e => DbError::FailDbAccess(e),
            })
    }

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

    /// ユーザーの追加
    pub async fn add_user(&self, name: &str, pass: &str) -> Result<(), DbError> {
        let sql = "insert into users(name, password) values (?, ?);";
        query(sql)
            .bind(name)
            .bind(pass)
            .execute(&self.pool)
            .await
            .map_err(|e| match e {
```

```
                sqlx::Error::Database(ref db_err) => {
                    if db_err.kind() == sqlx::error::ErrorKind::UniqueViolation {
                        DbError::DuplicateUserName(e)
                    } else {
                        DbError::FailDbAccess(e)
                    }
                }
                _ => DbError::FailDbAccess(e),
            })?;
        Ok(())
    }

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

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

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

    /// セッション情報を新規作成する。
    ///   生成した uuid を返す。
```

```rust
    pub async fn make_new_session(&self, user_name: &str) -> Result<Uuid, DbError> {
        let sql = "insert into sessions(id, user_name) values (?,?);";
        // キー情報の作成
        let id = Uuid::now_v7();

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

        Ok(id)
    }

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

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

```
223             .get("user_name");
224
225         // 旧セッションの削除
226         let sql_del_curr_sess = "delete from sessions where id = ?;";
227         query(sql_del_curr_sess)
228             .bind(id.to_string())
229             .execute(&mut *tr)
230             .await
231             .map_err(DbError::FailDbAccess)?;
232
233         // 新セッションの生成
234         let sql_create_sess = "insert into sessions(id, user_name) values (?, ?);";
235         let id = Uuid::now_v7();
236         query(sql_create_sess)
237             .bind(id.to_string())
238             .bind(user)
239             .execute(&mut *tr)
240             .await
241             .map_err(DbError::FailDbAccess)?;
242
243         tr.commit().await.map_err(DbError::FailDbAccess)?;
244         Ok(id)
245     }
246
247     /// 指定されたセッション ID が有効であるか確認する。
248     /// データベースエラーが発生した場合は、Err(DbError::FailDbAccess) を返す。
249     pub async fn is_session_valid(&self, sess: &Uuid) -> Result<bool, DbError> {
250         // 期限切れのセッションを削除する。
251         let sql_old_del = "delete from sessions where expired < now();";
252         query(sql_old_del)
253             .execute(&self.pool)
254             .await
255             .map_err(DbError::FailDbAccess)?;
256         // 指定セッション ID の有無を確認する。
257         let sql_find_sess = "select count(*) as cnt from sessions where id = ?;";
258         let sess_cnt: i64 = query(sql_find_sess)
259             .bind(sess.to_string())
260             .fetch_one(&self.pool)
261             .await
262             .map_err(DbError::FailDbAccess)?
263             .get("cnt");
264         if sess_cnt == 1 {
265             Ok(true)
266         } else {
267             Ok(false)
```

```
268             }
269         }
270     }
271
272     #[derive(FromRow, Debug, PartialEq)]
273     pub struct User {
274         pub name: String,
275         pub password: String,
276     }
277
278     #[derive(FromRow, Serialize, Deserialize, Debug, PartialEq, Clone)]
279     pub struct ItemTodo {
280         pub id: u32,
281         pub user_name: String,
282         pub title: String,
283         pub work: Option<String>,
284         pub update_date: Option<NaiveDate>,
285         pub start_date: Option<NaiveDate>,
286         pub end_date: Option<NaiveDate>,
287         pub done: bool,
288     }
289
290     #[derive(Error, Debug)]
291     pub enum DbError {
292         #[error("データベースへの接続に失敗。")]
293         FailConnect(sqlx::Error),
294         #[error("データベース操作失敗 (一般)")]
295         FailDbAccess(sqlx::Error),
296         #[error("User 挿入失敗 (name 重複)")]
297         DuplicateUserName(sqlx::Error),
298         #[error("ユーザーが見つかりません。")]
299         NotFoundUser,
300         #[error("指定されたセッション id が見つかりません。")]
301         NotFoundSession,
302         #[error("指定された id の todo が見つかりません。")]
303         NotFoundTodo,
304     }
305
306     #[cfg(test)]
307     mod test {
308         use chrono::Days;
309
310         use super::*;
311
312         /// テスト用の Database 生成。テスト用 Pool をインジェクション
```

```
313     impl Database {
314         pub(crate) fn new_test(pool: MySqlPool) -> Self {
315             Self { pool }
316         }
317     }
318 
319     /// ユーザー生成のテスト
320     #[sqlx::test]
321     async fn test_add_user_and_get_user(pool: MySqlPool) {
322         let db = Database::new_test(pool);
323         db.add_user("hyara", "password").await.unwrap();
324         let user = db.get_user("hyara").await.unwrap();
325         assert_eq!(user.name, "hyara");
326         assert_eq!(user.password, "password");
327         let error_user = db.get_user("naiyo").await;
328         match error_user {
329             Ok(_) => assert!(false, "結果が帰ってくるはずがない。"),
330             Err(DbError::NotFoundUser) => assert!(true),
331             Err(e) => assert!(false, "このエラーはおかしい。{e}"),
332         }
333     }
334 
335     /// セッション生成関係の一連のテスト。
336     #[sqlx::test]
337     async fn test_make_new_session(pool: MySqlPool) {
338         println!("まずはテスト用のユーザーの生成");
339         let db = Database::new_test(pool);
340         let user_name = "nekodayo";
341         let password = "password";
342         db.add_user(user_name, password).await.unwrap();
343 
344         println!("次に、普通にセッションを作ってみる。");
345         let sess1 = db.make_new_session(user_name).await.unwrap();
346         println!("セッション生成成功 id=[{}]", sess1.to_string());
347 
348         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。");
349         let sess2 = db.make_new_session("detarame").await;
350         match sess2 {
351             Ok(_) => assert!(false, "このユーザーは存在しなかったはず。"),
352             Err(DbError::NotFoundUser) => assert!(true),
353             Err(e) => assert!(false, "このエラーもおかしい。[{}]", e),
354         }
355 
356         println!("普通に、セッションを更新してみる。");
357         let sess3 = db.update_session(&sess1).await.unwrap();
```

```
358         assert_ne!(sess1, sess3);
359
360         println!("ないはずのセッションを更新しようとしてみる。");
361         let sess4 = Uuid::now_v7();
362         let sess5 = db.update_session(&sess4).await;
363         match sess5 {
364             Ok(_) => assert!(false, "このセッションはないはずなのに。"),
365             Err(DbError::NotFoundSession) => assert!(true),
366             Err(e) => assert!(false, "セッション更新２回め。失敗するにしてもこれはない{e}"),
367         }
368     }
369
370     /// セッションが有効かどうかを確認するテスト
371     #[sqlx::test]
372     async fn test_is_session_valid(pool: MySqlPool) {
373         let db = Database::new_test(pool);
374
375         println!("テスト用ユーザーの作成");
376         let name = "nekodayo";
377         let pass = "nekodamon";
378         db.add_user(name, pass).await.unwrap();
379
380         println!("新規セッションを生成する。");
381         let sess = db.make_new_session(name).await.unwrap();
382         println!("生成したセッション ID は、[{}] です。", &sess);
383
384         println!("今作ったセッション ID の妥当性を問い合わせてみる。");
385         assert!(db.is_session_valid(&sess).await.unwrap());
386
387         println!("偽セッション ID をいれて、問い合わせてみる。");
388         assert!(!db.is_session_valid(&Uuid::now_v7()).await.unwrap());
389     }
390
391     /// todo の書き込みと、単純な読み出しのテスト
392     #[sqlx::test]
393     async fn test_add_todo(pool: MySqlPool) {
394         let db = Database::new_test(pool);
395         let sess = login_for_test(&db).await;
396         let name = db.get_user_from_sess(sess).await.unwrap().name;
397
398         println!("テストデータをインサート");
399         let mut item = ItemTodo {
400             id: 0,
401             user_name: name.to_string(),
402             title: "インサートできるかな?".to_string(),
```

```
            work: Some("中身入り".to_string()),
            update_date: None,
            start_date: Some(Local::now().date_naive()),
            end_date: Some(Local::now().date_naive() + Days::new(3)),
            done: true,
        };
        db.add_todo_item(&item).await.unwrap();

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

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

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

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

```
448     }
449
450     /// todo の書き込みと読み出し
451     /// done=true と false の挙動テスト
452     #[sqlx::test]
453     async fn test_get_todo_done_param(pool: MySqlPool) {
454         let db = Database::new_test(pool.clone());
455         let sess = login_for_test(&db).await;
456         let name = db.get_user_from_sess(sess).await.unwrap().name;
457
458         println!("テストデータをインサート");
459         let item = ItemTodo {
460             id: 0,
461             user_name: name.to_string(),
462             title: "インサートできるかな?".to_string(),
463             work: None,
464             update_date: None,
465             start_date: Some(Local::now().date_naive()),
466             end_date: Some(Local::now().date_naive() + Days::new(3)),
467             done: true,
468         };
469         db.add_todo_item(&item).await.unwrap();
470
471         println!("テストデータを読み出す。一件しかないはず");
472         let last_day = Local::now().date_naive() + Days::new(1);
473         let res = db.get_todo_item(sess, last_day, false).await.unwrap();
474         assert_eq!(res.len(), 1, "全部読み出しだけど一件あるはず。");
475         let res = db.get_todo_item(sess, last_day, true).await.unwrap();
476         assert_eq!(res.len(), 1, "未完了だけだけど、一件あるはず。");
477
478         println!("今作った job を完了済みにする。");
479         let sql = "update todo set done=true where id=?;";
480         query(sql).bind(res[0].id).execute(&pool).await.unwrap();
481         let res = db.get_todo_item(sess, last_day, false).await.unwrap();
482         assert_eq!(res.len(), 1, "全部読み出しだけど一件あるはず。");
483         let res = db.get_todo_item(sess, last_day, true).await.unwrap();
484         assert_eq!(res.len(), 0, "未完了だけだけだから、なにもないはず。");
485     }
486
487     /// todo の書き込みと読み出し
488     /// 基準日の挙動テスト
489     #[sqlx::test]
490     async fn test_get_todo_ref_date(pool: MySqlPool) {
491         let db = Database::new_test(pool.clone());
492         let sess = login_for_test(&db).await;
```

```
493         let name = db.get_user_from_sess(sess).await.unwrap().name;
494
495         println!("テストデータをインサート");
496         let item = ItemTodo {
497             id: 0,
498             user_name: name.to_string(),
499             title: "インサートできるかな?".to_string(),
500             work: None,
501             update_date: None,
502             start_date: Some(Local::now().date_naive()),
503             end_date: Some(Local::now().date_naive() + Days::new(3)),
504             done: false,
505         };
506         db.add_todo_item(&item).await.unwrap();
507
508         let ref_date = Local::now().date_naive();
509         let res = db.get_todo_item(sess, ref_date, true).await.unwrap();
510         assert_eq!(res.len(), 1, "基準日と開始日が同じだからみつかる。");
511         let res = db
512             .get_todo_item(sess, ref_date + Days::new(1), true)
513             .await
514             .unwrap();
515         assert_eq!(res.len(), 1, "開始日の翌日が基準日だからみつかる。");
516         let res = db
517             .get_todo_item(sess, ref_date - Days::new(1), true)
518             .await
519             .unwrap();
520         assert_eq!(res.len(), 0, "基準日が開始日の前日だからみつからない。");
521         let res = db
522             .get_todo_item(sess, ref_date + Days::new(4), true)
523             .await
524             .unwrap();
525         assert_eq!(res.len(), 1, "基準日が期限を過ぎているけどみつかるの。");
526     }
527
528     #[sqlx::test]
529     async fn test_get_user_from_sess(pool: MySqlPool) {
530         let db = Database::new_test(pool.clone());
531
532         let sess = login_for_test(&db).await;
533         let name = db.get_user_from_sess(sess).await.unwrap().name;
534
535         let user = db.get_user_from_sess(sess).await.unwrap();
536         assert_eq!(user.name, name, "これはみつかるはず");
537         let dummy_sess = Uuid::now_v7();
```

```
538         let user = db.get_user_from_sess(dummy_sess).await;
539         match user {
540             Ok(_) => assert!(false, "見つかるわけないでしょう。"),
541             Err(DbError::NotFoundSession) => {}
542             Err(e) => assert!(false, "トラブルです。{e}"),
543         };
544     }
545
546     #[sqlx::test]
547     async fn test_change_done(pool: MySqlPool) {
548         let db = Database::new_test(pool);
549         let sess = login_for_test(&db).await;
550         let ref_date = Local::now().date_naive();
551         create_todo_for_test(&db, sess).await;
552
553         let items = db.get_todo_item(sess, ref_date, true).await.unwrap();
554         let item = items
555             .iter()
556             .find(|&i| i.title.find("二件目").is_some())
557             .unwrap();
558         db.change_done(item.id, true).await.unwrap();
559
560         let items = db.get_todo_item(sess, ref_date, true).await.unwrap();
561         let item = items.iter().find(|&i| i.title.find("二件目").is_some());
562         assert!(item.is_none(), "状態を完了にしたので見つからないはず。");
563
564         let items = db.get_todo_item(sess, ref_date, false).await.unwrap();
565         let item = items.iter().find(|&i| i.title.find("二件目").is_some());
566         match item {
567             Some(i) => assert!(i.done, "完了済みになっているはずですね?"),
568             None => assert!(false, "状態を変えたら、レコードなくなった???"),
569         }
570         assert_eq!(
571             items.len(),
572             3,
573             "全件見ているのでレコードは 3 件あるはずですが?"
574         );
575     }
576
577     #[sqlx::test]
578     async fn test_get_todo_with_id(pool: MySqlPool) {
579         let db = Database::new_test(pool);
580         let sess = login_for_test(&db).await;
581         create_todo_for_test(&db, sess).await;
582
```

```
583         let items = db
584             .get_todo_item(sess, Local::now().date_naive(), false)
585             .await
586             .unwrap();
587         let id = items
588             .iter()
589             .find(|&i| i.title.find("一件目").is_some())
590             .expect("これはあるはず")
591             .id;
592         let non_exist_id = items.iter().max_by_key(|&i| i.id).unwrap().id + 1;
593 
594         // 正常な読み出し
595         let res = db
596             .get_todo_item_with_id(id, sess)
597             .await
598             .expect("これは正常に読み出せるはず。エラーはだめ");
599         res.work
600             .expect("このレーコードは work を持つはずです。")
601             .find("働いてます。")
602             .expect("work の内容がおかしい。");
603 
604         // 間違った id
605         let res = db.get_todo_item_with_id(non_exist_id, sess).await;
606         match res {
607             Ok(_) => assert!(false, "そんな ID は存在しなかったはずなのに。"),
608             Err(DbError::NotFoundTodo) => { /* 正常 */ }
609             Err(e) => assert!(false, "データベースエラーだよ。({e})"),
610         }
611 
612         // 間違ったセッション
613         let res = db.get_todo_item_with_id(id, Uuid::now_v7()).await;
614         match res {
615             Ok(_) => assert!(false, "そんなセッションはないはず。"),
616             Err(DbError::NotFoundTodo) => { /* 正常 */ }
617             Err(e) => assert!(false, "データベースエラー発生。({e})"),
618         }
619     }
620 
621     async fn login_for_test(db: &Database) -> Uuid {
622         println!("テスト用ユーザー及びセッションの生成");
623         let name = "test";
624         let pass = "test";
625         db.add_user(name, pass).await.unwrap();
626         db.make_new_session(name).await.unwrap()
627     }
```

```
628
629     async fn create_todo_for_test(db: &Database, sess: Uuid) {
630         let name = db.get_user_from_sess(sess).await.unwrap().name;
631
632         println!("テストデータをインサート");
633         let item = ItemTodo {
634             id: 0,
635             user_name: name.to_string(),
636             title: "一件目（work 有り）".to_string(),
637             work: Some("働いてます。".to_string()),
638             update_date: None,
639             start_date: Some(Local::now().date_naive()),
640             end_date: Some(Local::now().date_naive() + Days::new(3)),
641             done: false,
642         };
643         db.add_todo_item(&item).await.unwrap();
644
645         let item = ItemTodo {
646             id: 0,
647             user_name: name.to_string(),
648             title: "二件目（work 無し）".to_string(),
649             work: None,
650             update_date: None,
651             start_date: Some(Local::now().date_naive()),
652             end_date: Some(Local::now().date_naive() + Days::new(3)),
653             done: false,
654         };
655         db.add_todo_item(&item).await.unwrap();
656
657         let item = ItemTodo {
658             id: 0,
659             user_name: name.to_string(),
660             title: "三件目（work 無し）".to_string(),
661             work: None,
662             update_date: None,
663             start_date: Some(Local::now().date_naive()),
664             end_date: Some(Local::now().date_naive() + Days::new(3)),
665             done: false,
666         };
667         db.add_todo_item(&item).await.unwrap();
668     }
669 }
```

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

const query_client = new QueryClient();

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

2.4 アプリケーションメイン App.jsx

```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";

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>
        </>
    ));

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

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

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

}

export default Init;
```

2.7 ユーザー登録画面 RegistUser.jsx

```jsx
/* ユーザー登録画面 */

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

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

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

```
                <Text>{sendMessage}</Text>
            </VStack>
        </>
    );
}

export default RegistUser;

```

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";

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

    const onSubmit = async (data) => {
        try {
            setSendMessage('処理中です。');
            await invoke('login', { name: data.name, password: data.pass });
            navi('/');
        } catch (e) {
            setSendMessage('エラーが発生しました。{' + e + '}');
            console.log(e);
        }
    };

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

```
                <Button type="submit" w="30%" ml="auto" mr="auto"> ログイン </Button>
                <Text>{sendMessage}</Text>
            </VStack>
        </>
    );
}

export default Login;

```

2.9 todo リストの表示 TodoList.jsx

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

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

function TodoList() {

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

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

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

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

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

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

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


export default TodoList;
```

2.10 todo アイテム表示 todoitem.jsx

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

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

    const onClick = () => {
        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);

    // アイコンボタンのアイコン選択
    let done_icon;
    if (item.done) {
        done_icon = <BsEmojiGrin/>;
    } else if (!!end_date && geDate(new Date(), end_date)) {
        done_icon = <BsAlarm/>;
    } else {
        done_icon = <BsWrenchAdjustable/>
    }

    return (
```

```
        <>
            <SimpleGrid w="full" columns={{base: 2, md: 1}} gap="md">
                <GridItem>
                    <IconButton size="xs" icon={done_icon} onClick={onClick}/>
                </GridItem>

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

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

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

```

2.11 todo アイテムの追加 AddTodo.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 { invoke } from "@tauri-apps/api/core";
import { useNavigate } from "react-router-dom";
import { useMutation } from "@tanstack/react-query";

function AddTodo() {
    const form = useForm();
    const { register, handleSubmit, formState: {errors} } = form;
    const [ sendMessage, setSendMessage ] = useState("");
    const navi = useNavigate();

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

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

```
                    <InputDate name="start" label="開始"/>
                    <InputDate name="end" label="終了"/>
                    <Button type="submit" w="30%" ml="auto" mr="auto">送信</Button>
                    <Text> {isPending ? "送信中です。" : null} </Text>
                    <Text> {sendMessage} </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>
        </>
    );
}

```

```
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;
        }
    } catch(e) {
        return null;
    }
```

```
134     if (Number.isNaN(ret_date.getTime())) { return null; }
135
136     return ret_date;
137 }
138
139 export default AddTodo;
```

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)
    );

```