

neko_todo ソースリスト

美都

2025 年 2 月 22 日

目次

1	Rust ソース	2
1.1	メインモジュール main.rs	2
1.2	アプリケーションステータス app_status.rs	5
1.3	コンフィグ設定処理 config.rs	6
1.4	アプリケーション設定情報の処理 setup.rs	12
1.5	todo モデル処理 todo.rs	15
1.6	データベースアクセス database.rs	27
1.7	command モジュール tauri::command 関数群 command.rs	48
1.7.1	todo リストの表示・編集 todo.rs	49
1.7.2	ユーザー操作関係 user.rs	53
1.7.3	セッション操作関連 session.rs	54
1.7.4	アプリケーションの状態操作 app_state.rs	56
2	フロントエンド React 関係	58
2.1	index.html	58
2.2	メイン CSS ファイル	59
2.3	main.jsx	62
2.4	アプリケーションメイン App.jsx	63
2.5	全体のベースページ BasePage.jsx	64
2.6	アプリケーションの初期化 Init.jsx	65
2.7	ユーザー登録画面 RegistUser.jsx	66
2.8	ログイン画面 Login.jsx	68
2.9	todo リストの表示 TodoList.jsx	70
2.10	todo アイテム表示 TodoItem.jsx	72
2.11	todo リスト画面 ツールバー TodoListToolbar.jsx	75
2.12	todo アイテムの追加 AddTodo.jsx	77
2.13	todo アイテムの編集 EditTodo.jsx	78
2.14	todo アイテムの複製 PasteTodo.jsx	80
2.15	todo アイテム内容の入力フォーム InputTodo.jsx	82
2.16	todo アイテム処理のための日付処理ユーティリティ str2date.jsx	85
3	データベース構成	87
3.1	テーブル生成スクリプト create_table.sql	87

1 Rust ソース

1.1 メインモジュール main.rs

```
1  // Prevents additional console window on Windows in release, DO NOT REMOVE!!
2  #![cfg_attr(not(debug_assertions), windows_subsystem = "windows")]
3
4  mod app_status;
5  mod command;
6  mod config;
7  mod database;
8  mod setup;
9  mod todo;
10
11 use app_status::AppStatus;
12 use command::app_state::{
13     get_is_incomplete, get_item_sort_order, set_is_incomplete, set_item_sort_order,
14 };
15 use command::session::is_valid_session;
16 use command::todo::{add_todo, edit_todo, get_todo_list, get_todo_with_id, update_done};
17 use command::user::{login, regist_user};
18 use directories::ProjectDirs;
19 use log::{error, info};
20 use setup::setup;
21 use tauri::Manager;
22
23 fn main() {
24     let mut log_file: std::path::PathBuf = ProjectDirs::from("jp", "laki", "nekotodo")
25         .unwrap()
26         .config_dir()
27         .into();
28     if !log_file.exists() {
29         std::fs::create_dir_all(&log_file).unwrap();
30     }
31     log_file.push("nekotodo.log");
32
33     fern::Dispatch::new()
34         .format(|out, message, record| {
35             out.finish(format_args!(
36                 "{} [{}] {}:{} {}",
37                 chrono::Local::now().format("%Y/%m/%d %H:%M:%S"),
38                 record.level(),
39                 record.file().unwrap(),
40                 record.line().unwrap(),
41                 message

```

```

42         ))
43     })
44     ///.level(log::LevelFilter::Info)
45     .level(log::LevelFilter::Debug)
46     .chain(std::io::stderr())
47     .chain(fern::log_file(log_file).unwrap())
48     .apply()
49     .unwrap();
50
51     run()
52 }
53
54 #[cfg_attr(mobile, tauri::mobile_entry_point)]
55 pub fn run() {
56     let app_status = match setup() {
57         Ok(s) => s,
58         Err(e) => {
59             error!("{}", e);
60             std::process::exit(1)
61         }
62     };
63
64     let app = tauri::Builder::default()
65         .plugin(tauri_plugin_shell::init())
66         .manage(app_status)
67         .invoke_handler(tauri::generate_handler![
68             get_todo_list,
69             get_todo_with_id,
70             regist_user,
71             login,
72             is_valid_session,
73             add_todo,
74             update_done,
75             edit_todo,
76             set_is_incomplete,
77             get_is_incomplete,
78             set_item_sort_order,
79             get_item_sort_order,
80         ])
81         .build(tauri::generate_context!())
82         .expect("error thile build tauri application");
83     app.run(|app, event| {
84         if let tauri::RunEvent::Exit = event {
85             info!("終了処理開始");
86             let state = app.state::<AppState>();
87             state.config().lock().unwrap().save().unwrap();

```

```
88         }  
89     });  
90 }
```

1.2 アプリケーションステータス app_status.rs

```
1  //! アプリケーション全体のステータスを保持する。
2
3  use crate::{config::NekoTodoConfig, todo::Todo};
4  use std::sync::{Arc, Mutex};
5
6  pub struct AppStatus {
7      config: Arc<Mutex<NekoTodoConfig>>,
8      todo: Todo,
9  }
10
11  impl AppStatus {
12      pub fn new(config: NekoTodoConfig, todo: Todo) -> Self {
13          Self {
14              config: Arc::new(Mutex::new(config)),
15              todo,
16          }
17      }
18
19      pub fn config(&self) -> &Mutex<NekoTodoConfig> {
20          &self.config
21      }
22
23      pub fn todo(&self) -> &Todo {
24          &self.todo
25      }
26  }
```

1.3 コンフィグ設定処理 config.rs

```
1  /// アプリケーション設定の取得関係
2
3  use directories::ProjectDirs;
4  use std::{
5      fs::OpenOptions,
6      io::{BufWriter, ErrorKind, Result, Write},
7      path::PathBuf,
8  };
9  use uuid::Uuid;
10
11  const CONF_FILE_NAME: &str = "neko_todo.conf";
12  const DB_HOST: &str = "NEKO_DB_DB_HOST";
13  const DB_USER: &str = "NEKO_DB_DB_USER";
14  const DB_PASS: &str = "NEKO_DB_DB_PASS";
15  const SESSION: &str = "NEKO_DB_SESSION_ID";
16
17  /// アプリケーション全体の状態設定
18  #[derive(Debug)]
19  pub struct NekoTodoConfig {
20      db_host: String,
21      db_user: String,
22      db_pass: String,
23      session_id: Option<Uuid>,
24      dirty: bool,
25      is_incomplete: bool,
26      item_sort_order: ItemSortOrder,
27  }
28
29  impl NekoTodoConfig {
30      pub fn new() -> dotenvy::Result<Self> {
31          let file = Self::get_config_file_path().map_err(dotenvy::Error::Io)?;
32          dotenvy::from_path(file)?;
33          let session_id = std::env::var(SESSION)
34              .ok()
35              .map(|s| Uuid::parse_str(&s).expect("環境ファイル異常:SESSION_ID 不正"));
36
37          Ok(Self {
38              db_host: std::env::var(DB_HOST).unwrap_or_default(),
39              db_user: std::env::var(DB_USER).unwrap_or_default(),
40              db_pass: std::env::var(DB_PASS).unwrap_or_default(),
41              session_id,
42              dirty: false,
43              is_incomplete: true,
44              item_sort_order: ItemSortOrder::EndAsc,
```

```

45         })
46     }
47
48     pub fn get_db_host(&self) -> &str {
49         &self.db_host
50     }
51
52     pub fn get_db_user(&self) -> &str {
53         &self.db_user
54     }
55
56     pub fn get_db_pass(&self) -> &str {
57         &self.db_pass
58     }
59
60     pub fn get_session_id(&self) -> Option<Uuid> {
61         self.session_id
62     }
63
64     pub fn get_is_incomplete(&self) -> bool {
65         self.is_incomplete
66     }
67
68     pub fn get_item_sort_order(&self) -> ItemSortOrder {
69         self.item_sort_order
70     }
71
72     pub fn set_db_host(&mut self, val: &str) {
73         self.db_host = val.to_string();
74         self.dirty = true;
75     }
76
77     pub fn set_db_user(&mut self, val: &str) {
78         self.db_user = val.to_string();
79         self.dirty = true;
80     }
81
82     pub fn set_db_pass(&mut self, val: &str) {
83         self.db_pass = val.to_string();
84         self.dirty = true;
85     }
86
87     pub fn set_session_id(&mut self, uuid: &Uuid) {
88         self.session_id = Some(*uuid);
89         self.dirty = true;
90     }

```

```

91
92 pub fn set_is_incomplete(&mut self, is_incomplete: bool) {
93     self.is_incomplete = is_incomplete;
94 }
95
96 pub fn set_item_sort_order(&mut self, item_sort_order: ItemSortOrder) {
97     self.item_sort_order = item_sort_order;
98 }
99
100 pub fn save(&mut self) -> Result<()> {
101     if !self.dirty {
102         return Ok(());
103     }
104     let path = Self::get_config_file_path()?;
105     let file = OpenOptions::new().write(true).truncate(true).open(&path)?;
106     let mut buffer = BufWriter::new(file);
107     writeln!(buffer, "{}={}", DB_HOST, self.get_db_host()?);
108     writeln!(buffer, "{}={}", DB_USER, self.get_db_user()?);
109     writeln!(buffer, "{}={}", DB_PASS, self.get_db_pass()?);
110     if let Some(s) = self.session_id {
111         writeln!(buffer, "{}={}", SESSION, s)?;
112     }
113     self.dirty = false;
114     Ok(())
115 }
116
117 /// コンフィグファイルのファイル名を生成する
118 /// 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
119 /// 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。
120 fn get_config_file_path() -> Result<PathBuf> {
121     use std::io;
122     // 環境依存コンフィグ用ディレクトリの取得
123     // 必要であれば、自分用のディレクトリを生成する。
124     // ここでエラーになるのは、OSシステムに問題がある。
125     let mut path: PathBuf = ProjectDirs::from("jp", "laki", "nekotodo")
126         .ok_or(io::Error::new(ErrorKind::Other, "Not Found Home"))?
127         .config_dir()
128         .into();
129     if let Err(e) = std::fs::create_dir(&path) {
130         if e.kind() != ErrorKind::AlreadyExists {
131             return Err(e);
132         }
133     }
134
135     // 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。
136     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 {

```

```

183         "StartAsc" => Ok(Self::StartAsc),
184         "StartDesc" => Ok(Self::StartDesc),
185         "EndAsc" => Ok(Self::EndAsc),
186         "EndDesc" => Ok(Self::EndDesc),
187         "UpdateAsc" => Ok(Self::UpdateAsc),
188         "UpdateDesc" => Ok(Self::UpdateDesc),
189         _ => Err(ItemSortOrderParseError::InvalidArgument),
190     }
191 }
192 }
193
194 #[derive(thiserror::Error, Debug)]
195 pub enum ItemSortOrderParseError {
196     #[error("Invalid Argument")]
197     InvalidArgument,
198 }
199
200 #[cfg(test)]
201 mod tests {
202     use super::*;
203
204     /// 環境設定の挙動テスト
205     #[test]
206     #[ignore]
207     fn test_env_val() {
208         let val_db_host = "test_host";
209         let val_db_user = "test_user";
210         let val_db_pass = "test_pass";
211         save_curr_conf_file();
212         {
213             let mut conf = NekoTodoConfig::new().unwrap();
214             // 初期状態では空文字列が返るはず
215             assert_eq!(conf.get_db_host(), "");
216             assert_eq!(conf.get_db_user(), "");
217             assert_eq!(conf.get_db_pass(), "");
218             // test_host をセットしてセットされているか確認。
219             conf.set_db_host(val_db_host);
220             conf.set_db_user(val_db_user);
221             conf.set_db_pass(val_db_pass);
222             assert_eq!(conf.get_db_host(), val_db_host);
223             assert_eq!(conf.get_db_user(), val_db_user);
224             assert_eq!(conf.get_db_pass(), val_db_pass);
225         } // この時点で一旦環境ファイルを保存してみる。
226         // 環境ファイルをもう一度ロードして、環境を確認
227         delete_env_val();
228         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_PASS);
272 }
273 }

```

1.4 アプリケーション設定情報の処理 setup.rs

```
1  ///! アプリケーション環境の構築を実施する
2  use clap::Parser;
3  use log::{error, info};
4  use std::process::exit;
5  use tauri::async_runtime::block_on;
6  use thiserror::Error;
7
8  use crate::{
9      app_status::AppStatus,
10     config::NekoTodoConfig,
11     todo::{Todo, TodoError},
12 };
13
14 ///! アプリケーション環境の構築を行う。
15 pub fn setup() -> Result<AppStatus, SetupError> {
16     let args = Args::parse();
17     if args.setup {
18         database_param_setup(&args)?;
19     }
20
21     let conf = NekoTodoConfig::new()?;
22
23     if conf.get_db_host().is_empty()
24         || conf.get_db_user().is_empty()
25         || conf.get_db_pass().is_empty()
26     {
27         return Err(SetupError::Argument);
28     }
29
30     let todo = block_on(async {
31         Todo::new(conf.get_db_host(), conf.get_db_user(), conf.get_db_pass()).await
32     })?;
33
34     Ok(AppStatus::new(conf, todo))
35 }
36
37 ///! 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。
38 fn database_param_setup(args: &Args) -> Result<(), SetupError> {
39     let Some(ref host) = args.server else {
40         return Err(SetupError::Argument);
41     };
42     let Some(ref user) = args.user else {
43         return Err(SetupError::Argument);
44     };
45 }
```

```

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

```

```
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  }
```

1.5 todo モデル処理 todo.rs

```
1 use bcrypt::{hash, verify, DEFAULT_COST};
2 use chrono::Local;
3 use log::error;
4 use thiserror::Error;
5 use uuid::Uuid;
6
7 use crate::{config::ItemSortOrder, database::*};
8
9 /// todo リストの処理全般
10 pub struct Todo {
11     database: Database,
12 }
13
14 impl Todo {
15     /// 初期化
16     pub async fn new(host: &str, user: &str, pass: &str) -> Result<Self, TodoError> {
17         let db = Database::new(host, user, pass).await.map_err(|e| match e {
18             DbError::FailConnect(e2) => TodoError::DbInit(e2),
19             e => unreachable!("[ToDo::new] Database::new() [{e}]"),
20         })?;
21         Ok(Self { database: db })
22     }
23
24     /// todo の一覧を取得する。(仮実装。インターフェース未確定)
25     pub async fn get_todo_list(
26         &self,
27         sess: Uuid,
28         only_imcomplete: bool,
29         sort_order: ItemSortOrder,
30     ) -> Result<Vec<ItemTodo>, TodoError> {
31         let ref_date = Local::now().date_naive();
32         self.database
33             .get_todo_item(sess, ref_date, only_imcomplete, sort_order)
34             .await
35             .map_err(|e| match e {
36                 DbError::FailDbAccess(e) => TodoError::FailDbAccess(e),
37                 e => unreachable!("[get_todo_list]get_todo_item[{e}]"),
38             })
39     }
40
41     /// 新規の todo を追加する
42     /// 引数 item の id, user_name, update_date, update_date は無視される。
43     pub async fn add_todo(&self, sess: Uuid, item: &ItemTodo) -> Result<(), TodoError> {
44         // ユーザー名を取得
```

```

45     let user = self
46         .database
47         .get_user_from_sess(sess)
48         .await
49         .map_err(|e| match e {
50             DbError::NotFoundSession => TodoError::NotFoundSession,
51             DbError::FailDbAccess(e) => {
52                 error!("[Todo::add_todo]get_user_from_sess:[{e}]");
53                 TodoError::FailDbAccess(e)
54             }
55             e => unreachable!("[add_todo]get_user_from_sess[{e}]"),
56         })?;
57     // アイテムを登録
58     let mut item = item.clone();
59     item.user_name = user.name.clone();
60     if let Some(ref s) = item.work {
61         if s.trim().is_empty() {
62             item.work = None;
63         }
64     }
65     self.database
66         .add_todo_item(&item)
67         .await
68         .map_err(|e| match e {
69             DbError::FailDbAccess(e) => {
70                 error!("[Todo::add_todo]add_todo_item:[{e}]");
71                 TodoError::FailDbAccess(e)
72             }
73             e => unreachable!("[add_todo]add_todo_item[{e}]"),
74         })
75 }
76
77 /// idとsessを指定してtodoを取得する。
78 /// 一致するtodoがなければ、エラー、TodoError::NotFoundTodoを返す。
79 pub async fn get_todo_with_id(&self, id: u32, sess: Uuid) -> Result<ItemTodo, TodoError> {
80     self.database
81         .get_todo_item_with_id(id, sess)
82         .await
83         .map_err(|e| match e {
84             DbError::NotFoundTodo => TodoError::NotFoundTodo,
85             DbError::FailDbAccess(e) => {
86                 error!("[Todo::get_todo_with_id]get_todo_item_with_id:[{e}]");
87                 TodoError::FailDbAccess(e)
88             }
89             e => unreachable!("[Todo::get_todo_with_id]get_todo_item_with_id[{e}]"),
90         })

```



```

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));

```

```

137         }
138         _ => {}
139     }
140 }
141 Ok(())
142 }
143
144 /// ログイン処理を行う。
145 pub async fn login(&self, name: &str, password: &str) -> Result<Uuid, TodoError> {
146     // 認証
147     let user = self.database.get_user(name).await.map_err(|e| match e {
148         DbError::NotFoundUser => TodoError::NotFoundUser,
149         DbError::FailDbAccess(e) => TodoError::FailDbAccess(e),
150         e => unreachable!("[Todo::login] Database::get_user: [{e}]"),
151     })?;
152     if !verify(password, &user.password)? {
153         return Err(TodoError::WrongPassword);
154     }
155     // セッションの生成
156     let session = self
157         .database
158         .make_new_session(&user.name)
159         .await
160         .map_err(|e| match e {
161             DbError::NotFoundUser => TodoError::NotFoundUser,
162             DbError::FailDbAccess(e) => TodoError::FailDbAccess(e),
163             e => {
164                 unreachable!("[Todo::login] Database::make_new_session: [{e}]")
165             }
166         })?;
167     Ok(session)
168 }
169
170 /// 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。
171 /// もし指定されたセッション ID が無効な場合は、None を返す。
172 /// 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。
173 pub async fn is_valid_session(&self, sess: &Uuid) -> Result<Option<Uuid>, TodoError> {
174     let is_valid = self
175         .database
176         .is_session_valid(sess)
177         .await
178         .map_err(|e| match e {
179             DbError::FailDbAccess(e) => TodoError::FailDbAccess(e),
180             e => {
181                 unreachable!("[Todo::is_valid_session] is_session_valid: [{e}]")
182             }
183         })

```

```

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;

```

```

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

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

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

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

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

```

```

275     let sess = todo.login(user_name, user_pass).await.unwrap();
276
277     // 正しいセッションを検索する。
278     let new_sess = todo.is_valid_session(&sess).await.unwrap();
279     match new_sess {
280         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。"),
281         None => unreachable!("正しいセッションが見つからなかった。"),
282     };
283
284     // 間違ったセッションを検索する。
285     let none_sess = todo.is_valid_session(&Uuid::now_v7()).await.unwrap();
286     if none_sess.is_some() {
287         unreachable!("こんなセッションがあるわけがない。");
288     }
289 }
290
291 #[sqlx::test]
292 async fn add_todo_test(pool: MySqlPool) {
293     use chrono::Days;
294
295     let todo = Todo::test_new(pool);
296     let sess = login_for_test(&todo).await;
297
298     let item1 = ItemTodo {
299         id: 100,
300         user_name: "kore_naihazu".to_string(),
301         title: "テストアイテム 1 件目".to_string(),
302         work: Some("これは、中身を入れる.".to_string()),
303         update_date: None,
304         start_date: Some(Local::now().date_naive() - Days::new(1)),
305         end_date: Some(Local::now().date_naive() + Days::new(5)),
306         done: true,
307     };
308     let item2 = ItemTodo {
309         id: 100,
310         user_name: "kore_naihazu".to_string(),
311         title: "テストアイテム 2 件目 (work=null)".to_string(),
312         work: Some("").to_string(),
313         update_date: None,
314         start_date: Some(Local::now().date_naive() - Days::new(1)),
315         end_date: Some(Local::now().date_naive() + Days::new(5)),
316         done: true,
317     };
318     let item3 = ItemTodo {
319         id: 100,
320         user_name: "kore_naihazu".to_string(),

```

```

321     title: "テストアイテム 3 件目 (work=space)".to_string(),
322     work: Some(" \t ".to_string()),
323     update_date: None,
324     start_date: Some(Local::now().date_naive() - Days::new(1)),
325     end_date: Some(Local::now().date_naive() + Days::new(5)),
326     done: true,
327 };
328 todo.add_todo(sess, &item1)
329     .await
330     .expect("1 件目の追加に失敗");
331 let res = todo
332     .get_todo_list(sess, true, ItemSortOrder::EndAsc)
333     .await
334     .expect("1 件目の取得に失敗");
335 assert_eq!(res.len(), 1, "一件目が取得できなかった?");
336 assert_eq!(res[0].title, item1.title, "一件目の title が違う");
337 assert_eq!(res[0].work, item1.work, "一件目の work が違う");
338 assert_eq!(res[0].user_name, "testdayo", "一件目の user_name が違う");
339 assert_eq!(
340     res[0].update_date,
341     Some(Local::now().date_naive()),
342     "一件目の update_date が違う"
343 );
344 assert_eq!(res[0].start_date, item1.start_date, "一件目の開始日が違う");
345 assert_eq!(res[0].end_date, item1.end_date, "一件目の終了日が違う");
346 assert!(res[0].done, "一件目の完了マークが違う");
347
348 todo.add_todo(sess, &item2)
349     .await
350     .expect("二件目の追加に失敗");
351 let res = todo
352     .get_todo_list(sess, true, ItemSortOrder::EndAsc)
353     .await
354     .expect("二件目の取得に失敗");
355 assert_eq!(res.len(), 2, "二件あるはずなんだけど");
356 assert!(
357     res.iter()
358         .find(&x| match x.title.find("work=null") {
359             Some(n) => n > 0,
360             None => false,
361         })
362         .expect("二件目に追加したデータがない")
363         .work
364         .is_none(),
365     "二件目の work は None のはず"
366 );

```

```

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(),

```

```

413         2,
414         "一件完了済みにしたので、このリストは2件しかない。"
415     );
416     let items = todo
417         .get_todo_list(sess, false, ItemSortOrder::EndAsc)
418         .await
419         .unwrap();
420     assert_eq!(items.len(), 3, "完了済みを含むので、3件になる。");
421     let item = items
422         .iter()
423         .find(|&i| i.id == id)
424         .expect("さっきあったidだから必ずある。");
425     assert!(item.done, "さっき完了済みに変更した。");
426
427     let max_id = items.iter().max_by_key(|&x| x.id).unwrap().id;
428     let res = todo.change_done(max_id + 1, sess, false).await;
429     match res {
430         Ok(_) => unreachable!("このidのtodoがあるはずがない。"),
431         Err(TodoError::NotFoundTodo) => {}
432         Err(e) => unreachable!("このエラーもありえない。[{e}]"),
433     };
434
435     // 間違ったセッションのテスト
436     let res = todo.change_done(id, Uuid::now_v7(), true).await;
437     match res {
438         Ok(_) => unreachable!("このセッションでは、更新を許してはいけない。"),
439         Err(TodoError::NotFoundTodo) => { /* 正常 */ }
440         Err(e) => unreachable!("このエラーもおかしい。[{e}]"),
441     }
442 }
443
444 #[sqlx::test]
445 async fn edit_todo_test(pool: MySqlPool) {
446     let todo = Todo::test_new(pool);
447     let sess = login_for_test(&todo).await;
448     create_todo_for_test(&todo, sess).await;
449
450     let items = todo
451         .get_todo_list(sess, false, ItemSortOrder::EndAsc)
452         .await
453         .unwrap();
454     let mut item = items
455         .iter()
456         .find(|&i| i.title.contains("1 件目"))
457         .unwrap()
458         .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,

```

```

505         user_name: "kore_naihazu".to_string(),
506         title: "テストアイテム 2 件目 (work=null)".to_string(),
507         work: Some("").to_string(),
508         update_date: None,
509         start_date: Some(Local::now().date_naive() - Days::new(1)),
510         end_date: Some(Local::now().date_naive() + Days::new(5)),
511         done: false,
512     },
513     ItemTodo {
514         id: 100,
515         user_name: "kore_naihazu".to_string(),
516         title: "テストアイテム 3 件目 (work=space)".to_string(),
517         work: Some(" \t ").to_string(),
518         update_date: None,
519         start_date: Some(Local::now().date_naive() - Days::new(1)),
520         end_date: Some(Local::now().date_naive() + Days::new(5)),
521         done: false,
522     },
523 ];
524 for item in items {
525     todo.add_todo(sess, &item).await.unwrap();
526 }
527 }
528 }

```

1.6 データベースアクセス database.rs

```
1  /// データベースの操作を司る
2
3  use chrono::{Local, NaiveDate};
4  use log::error;
5  use serde::{Deserialize, Serialize};
6  use sqlx::{
7      mysql::{MySQLPool, MySQLPoolOptions},
8      prelude::*,
9      query, query_as,
10 };
11 use thiserror::Error;
12 use uuid::Uuid;
13
14 use crate::config::ItemSortOrder;
15
16 /// neko_db データベース操作関数郡
17 #[derive(Clone, Debug)]
18 pub struct Database {
19     pool: MySQLPool,
20 }
21
22 impl Database {
23     /// 新規生成。
24     pub async fn new(host: &str, user: &str, pass: &str) -> Result<Self, DbError> {
25         let db_url = format!("mariadb://{user}:{pass}@{host}/nekotodo", user, pass, host);
26         let pool = MySQLPoolOptions::new()
27             .max_connections(10)
28             .min_connections(3)
29             .connect(&db_url)
30             .await
31             .map_err(DbError::FailConnect)?;
32         Ok(Self { pool })
33     }
34
35     /// Todo 項目を追加する。
36     /// item 引数のうち、id, update_date, done は、無視される
37     /// 各々、自動値・今日の日付・false がはいる。
38     /// start_date, end_date のデフォルト値は、今日・NaiveDate::MAX である。
39     pub async fn add_todo_item(&self, item: &ItemTodo) -> Result<(), DbError> {
40         let sql = r#"
41             insert into todo(user_name, title, work, update_date, start_date, end_date, done)
42             values (?, ?, ?, curdate(), ?, ?, false);
43         "#;
44         let start_date = item.start_date.unwrap_or(Local::now().date_naive());
```

```

45     let end_date = item
46         .end_date
47         .unwrap_or(NaiveDate::from_ymd_opt(9999, 12, 31).unwrap());
48     query(sql)
49         .bind(&item.user_name)
50         .bind(&item.title)
51         .bind(&item.work)
52         .bind(start_date)
53         .bind(end_date)
54         .execute(&self.pool)
55         .await
56         .map_err(DbError::FailDbAccess)?;
57     Ok(())
58 }
59
60 /// Todo の一覧を取得する。
61 /// 基準日 (ref_date) 以降のアイテムを選別する。
62 /// セッション ID を必要とする。
63 /// 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。
64 pub async fn get_todo_item(
65     &self,
66     sess: Uuid,
67     ref_date: NaiveDate,
68     only_incomplete: bool,
69     sort_order: ItemSortOrder,
70 ) -> Result<Vec<ItemTodo>, DbError> {
71     let sql1 = r#"
72         select t.id, t.user_name, title, work, update_date, start_date, end_date, done
73         from todo t join sessions s on s.user_name = t.user_name
74         where s.id=? and t.start_date <= ?
75     "#;
76     let sql2 = " and done = false";
77     let sql3 = match sort_order {
78         ItemSortOrder::EndAsc => " order by end_date, update_date",
79         ItemSortOrder::EndDesc => " order by end_date desc, update_date",
80         ItemSortOrder::StartAsc => " order by start_date, update_date",
81         ItemSortOrder::StartDesc => " order by start_date desc, update_date",
82         ItemSortOrder::UpdateAsc => " order by update_date, end_date",
83         ItemSortOrder::UpdateDesc => " order by update_date desc, end_date",
84     };
85     let sql = if only_incomplete {
86         format!("{}", sql1, sql2, sql3)
87     } else {
88         format!("{}", sql1, sql3)
89     };
90     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,
105                ↪ t.done
106             from todo t join sessions s on s.user_name = t.user_name
107             where s.id=? and t.id=?
108             "#;
109         query_as::<_, ItemTodo>(sql)
110             .bind(sess.to_string())
111             .bind(id)
112             .fetch_one(&self.pool)
113             .await
114             .map_err(|e| match e {
115                 sqlx::Error::RowNotFound => DbError::NotFoundTodo,
116                 e => DbError::FailDbAccess(e),
117             })
118     }
119
120     /// Todo の完了状態を更新する。
121     pub async fn change_done(&self, id: u32, done: bool) -> Result<(), DbError> {
122         let sql = "update todo set done = ? where id = ?";
123         let res = query(sql)
124             .bind(done)
125             .bind(id)
126             .execute(&self.pool)
127             .await
128             .map_err(DbError::FailDbAccess)?;
129         if res.rows_affected() > 0 {
130             Ok(())
131         } else {
132             Err(DbError::NotFoundTodo)
133         }
134     }
135
136     /// 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 {

```

```

228         sqlx::Error::Database(ref e) => {
229             if e.is_foreign_key_violation() {
230                 // 外部キーエラー。存在しないユーザーを指定した。
231                 return DbError::NotFoundUser;
232             }
233             DbError::FailDbAccess(err)
234         }
235         _ => DbError::FailDbAccess(err),
236     })?;
237
238     Ok(id)
239 }
240
241 /// 指定されたセッションを新規セッションに更新する。
242 /// 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。
243 pub async fn update_session(&self, id: &uuid::Uuid) -> Result<Uuid, DbError> {
244     let mut tr = self.pool.begin().await.map_err(DbError::FailDbAccess)?;
245     // 期限切れのセッション削除
246     let sql_old_del = "delete from sessions where expired < now()";
247     query(sql_old_del)
248         .execute(&mut *tr)
249         .await
250         .map_err(DbError::FailDbAccess)?;
251
252     // ユーザー ID の特定
253     let sql_query_user = "select user_name from sessions where id=?";
254     let user: String = query(sql_query_user)
255         .bind(id.to_string())
256         .fetch_one(&mut *tr)
257         .await
258         .map_err(|e| match e {
259             sqlx::Error::RowNotFound => DbError::NotFoundSession,
260             e => DbError::FailDbAccess(e),
261         })?
262         .get("user_name");
263
264     // 旧セッションの削除
265     let sql_del_curr_sess = "delete from sessions where id = ?";
266     query(sql_del_curr_sess)
267         .bind(id.to_string())
268         .execute(&mut *tr)
269         .await
270         .map_err(DbError::FailDbAccess)?;
271
272     // 新セッションの生成
273     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してみる。");
424     assert!(db.is_session_valid(&sess).await.unwrap());
425
426     println!("偽セッション ID をいれて、問い合わせしてみる。");
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, "全部読み出した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, "未完了ただけだけど、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, "全部読み出した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, "未完了ただけだから、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INSERT");
553     let item = ItemTodo {
554         id: 0,
555         user_name: name.to_string(),
556         title: "INSERT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 }

```



```

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("働いています。".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();

```

```
918
919     let item = ItemTodo {
920         id: 0,
921         user_name: name.to_string(),
922         title: "三件目 (work 無し)".to_string(),
923         work: None,
924         update_date: None,
925         start_date: Some(Local::now().date_naive()),
926         end_date: Some(Local::now().date_naive() + Days::new(3)),
927         done: false,
928     };
929     db.add_todo_item(&item).await.unwrap();
930 }
931 }
```

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

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


1.7.1 todo リストの表示・編集 todo.rs

```
1  ///! todo リスト操作インターフェース
2
3  use super::session::{get_cur_session_with_update, get_curr_session};
4  use crate::app_status::AppStatus;
5  use crate::database::ItemTodo;
6  use log::{debug, info};
7  use serde::Deserialize;
8  use tauri::State;
9
10 /// todo のリストを取得する。
11 #[tauri::command]
12 pub async fn get_todo_list(app_status: State<'_, AppStatus>) -> Result<Vec<ItemTodo>, String> {
13     let sess = match get_curr_session(&app_status) {
14         Some(u) => u,
15         None => return Err("NotLogin".to_string()),
16     };
17
18     let is_incomplete;
19     let sort_order;
20     {
21         let conf = app_status.config().lock().unwrap();
22         is_incomplete = conf.get_is_incomplete();
23         sort_order = conf.get_item_sort_order();
24     }
25
26     let ret = app_status
27         .todo()
28         .get_todo_list(sess, is_incomplete, sort_order)
29         .await
30         .map_err(|e| e.to_string())?;
31     info!("todo リスト、{}件、取得完了", ret.len());
32     Ok(ret)
33 }
34
35 /// todo アイテムを取得する
36 #[tauri::command]
37 pub async fn get_todo_with_id(
38     app_status: State<'_, AppStatus>,
39     id: u32,
40 ) -> Result<ItemTodo, String> {
41     let Some(sess) = get_curr_session(&app_status) else {
42         return Err("NotLogin".to_string());
43     };
44
45     let ret = app_status
```

```

46         .todo()
47         .get_todo_with_id(id, sess)
48         .await
49         .map_err(|e| e.to_string())?;
50     info!("todo 一件の取得完了 id=>{}", id);
51     Ok(ret)
52 }
53
54 /// todoを追加する。
55 #[tauri::command]
56 pub async fn add_todo(app_status: State<'_, AppStatus>, item: FormTodo) -> Result<(), String> {
57     let sess = match get_cur_session_with_update(&app_status).await {
58         Ok(Some(u)) => u,
59         Ok(None) => return Err("NotLogin".to_string()),
60         Err(e) => return Err(e),
61     };
62
63     debug!("input = {:?}", &item);
64     app_status
65         .todo()
66         .add_todo(sess, &item.into())
67         .await
68         .map_err(|e| e.to_string())?;
69     info!("todo の追加完了");
70     Ok(())
71 }
72
73 /// todo の完了状態を変更する。
74 #[tauri::command]
75 pub async fn update_done(
76     app_status: State<'_, AppStatus>,
77     id: u32,
78     done: bool,
79 ) -> Result<(), String> {
80     let sess = match get_cur_session_with_update(&app_status).await {
81         Ok(Some(s)) => s,
82         Ok(None) => return Err("NotLogin".to_string()),
83         Err(e) => return Err(e),
84     };
85     app_status
86         .todo()
87         .change_done(id, sess, done)
88         .await
89         .map_err(|e| e.to_string())?;
90     info!(
91         "todo の状態を変更。id=>{}, state=>{}",

```

```

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

```

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

1.7.2 ユーザー操作関係 user.rs

```
1  /// ユーザー操作インターフェース
2
3  use crate::app_status::AppState;
4  use log::info;
5  use tauri::{command, State};
6
7  // ユーザー登録
8  #[command]
9  pub async fn regist_user(
10     app_status: State<'_, AppState>,
11     name: String,
12     password: String,
13 ) -> Result<(), String> {
14     app_status
15         .todo()
16         .add_user(&name, &password)
17         .await
18         .map_err(|e| e.to_string())?;
19     info!("ユーザー登録完了:user->{}", &name);
20     Ok(())
21 }
22
23 /// ログイン
24 #[command]
25 pub async fn login(
26     app_status: State<'_, AppState>,
27     name: String,
28     password: String,
29 ) -> Result<String, String> {
30     let session = app_status.todo().login(&name, &password).await?;
31
32     let mut cnf = app_status.config().lock().unwrap();
33     cnf.set_session_id(&session);
34     //cnf.save().map_err(|e| format!("OtherError:{}", e))?;
35     info!("ログイン完了:user->{}", &name);
36     Ok(session.to_string())
37 }
```

1.7.3 セッション操作関連 session.rs

```
1  /// セッション関係の関数及びインターフェース
2
3  use crate::app_status::AppStatus;
4  use log::info;
5  use tauri::{command, State};
6  use uuid::Uuid;
7
8  /// 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)
9  #[command]
10 pub async fn is_valid_session(app_status: State<'_, AppStatus>) -> Result<bool, String> {
11     let sess = get_cur_session_with_update(&app_status)
12         .await
13         .map(|i| i.is_some());
14     match sess {
15         Ok(sess) => info!("セッション確認 ({})", if sess { "有効" } else { "無効" }),
16         Err(ref e) => info!("セッション確認エラー ({})", e),
17     }
18     sess
19 }
20
21 /// 現在、有効なセッションを返す。
22 /// 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。
23 pub async fn get_cur_session_with_update(app_status: &AppStatus) -> Result<Option<Uuid>, String> {
24     let cur_session = get_curr_session(app_status);
25     let Some(cur_session) = cur_session else {
26         return Ok(None);
27     };
28
29     match app_status.todo().is_valid_session(&cur_session).await {
30         Ok(Some(s)) => {
31             // 更新されたセッションを再登録
32             let mut cnf = app_status.config().lock().unwrap();
33             cnf.set_session_id(&s);
34             //cnf.save().map_err(|e| format!("FailSession:{e}"))?;
35             Ok(Some(s))
36         }
37         Ok(None) => Ok(None),
38         Err(e) => Err(format!("FailSession:{e}")),
39     }
40 }
41
42 /// 現在のセッションを取得する。
43 pub fn get_curr_session(app_status: &AppStatus) -> Option<Uuid> {
44     let conf = app_status.config().lock().unwrap();
45     conf.get_session_id()
```


1.7.4 アプリケーションの状態操作 app_state.rs

```
1  /// アプリケーションの全体ステータスの取得・設定用インターフェース
2
3  use crate::app_status::AppStatus;
4  use crate::config::ItemSortOrder;
5  use log::info;
6  use tauri::{command, State};
7
8  /// 完了済みのみを表示するかどうかを設定する。
9  #[tauri::command]
10 pub fn set_is_incomplete(app_status: State<'_, AppStatus>, is_incomplete: bool) {
11     let mut conf = app_status.config().lock().unwrap();
12     conf.set_is_incomplete(is_incomplete);
13     info!("未完了のみ表示モードを{}にセット", is_incomplete);
14 }
15
16 /// 完了済みのみ表示モードの現在の値を取得する。
17 #[tauri::command]
18 pub fn get_is_incomplete(app_status: State<'_, AppStatus>) -> bool {
19     app_status.config().lock().unwrap().get_is_incomplete()
20 }
21
22 /// 現在のアイテムリストのソート方法を返す
23 #[command]
24 pub fn get_item_sort_order(app_status: State<'_, AppStatus>) -> String {
25     app_status
26         .config()
27         .lock()
28         .unwrap()
29         .get_item_sort_order()
30         .to_string()
31 }
32
33 /// アイテムリストのソート方法を設定する
34 #[command]
35 pub fn set_item_sort_order(
36     app_status: State<'_, AppStatus>,
37     sort_order: String,
38 ) -> Result<(), String> {
39     let sort_order = sort_order
40         .parse:::<ItemSortOrder>()
41         .map_err(|e| e.to_string())?;
42     app_status
43         .config()
44         .lock()
45         .unwrap()
```



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

2 フロントエンド React 関係

2.1 index.html

```
1  <!doctype html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>Tauri + React</title>
8    </head>
9
10   <body>
11     <div id="root"></div>
12     <script type="module" src="/src/main.jsx"></script>
13   </body>
14 </html>
```

2.2 メイン CSS ファイル

```
1  .logo.vite:hover {
2      filter: drop-shadow(0 0 2em #747bff);
3  }
4
5  .logo.react:hover {
6      filter: drop-shadow(0 0 2em #61dafb);
7  }
8  :root {
9      font-family: Inter, Avenir, Helvetica, Arial, sans-serif;
10     font-size: 16px;
11     line-height: 24px;
12     font-weight: 400;
13
14     color: #0f0f0f;
15     background-color: #f6f6f6;
16
17     font-synthesis: none;
18     text-rendering: optimizeLegibility;
19     -webkit-font-smoothing: antialiased;
20     -moz-osx-font-smoothing: grayscale;
21     -webkit-text-size-adjust: 100%;
22 }
23
24 .container {
25     margin: 0;
26     padding-top: 10vh;
27     display: flex;
28     flex-direction: column;
29     justify-content: center;
30     text-align: center;
31 }
32
33 .logo {
34     height: 6em;
35     padding: 1.5em;
36     will-change: filter;
37     transition: 0.75s;
38 }
39
40 .logo.tauri:hover {
41     filter: drop-shadow(0 0 2em #24c8db);
42 }
43
44 .row {
```

```

45     display: flex;
46     justify-content: center;
47 }
48
49 a {
50     font-weight: 500;
51     color: #646cff;
52     text-decoration: inherit;
53 }
54
55 a:hover {
56     color: #535bf2;
57 }
58
59 h1 {
60     text-align: center;
61 }
62
63 input,
64 button {
65     border-radius: 8px;
66     border: 1px solid transparent;
67     padding: 0.6em 1.2em;
68     font-size: 1em;
69     font-weight: 500;
70     font-family: inherit;
71     color: #0f0f0f;
72     background-color: #ffffff;
73     transition: border-color 0.25s;
74     box-shadow: 0 2px 2px rgba(0, 0, 0, 0.2);
75 }
76
77 button {
78     cursor: pointer;
79 }
80
81 button:hover {
82     border-color: #396cd8;
83 }
84 button:active {
85     border-color: #396cd8;
86     background-color: #e8e8e8;
87 }
88
89 input,
90 button {

```

```
91     outline: none;
92 }
93
94 #greet-input {
95     margin-right: 5px;
96 }
97
98 @media (prefers-color-scheme: dark) {
99     :root {
100         color: #f6f6f6;
101         background-color: #2f2f2f;
102     }
103
104     a:hover {
105         color: #24c8db;
106     }
107
108     input,
109     button {
110         color: #ffffff;
111         background-color: #0f0f0f98;
112     }
113     button:active {
114         background-color: #0f0f0f69;
115     }
116 }
```

2.3 main.jsx

```
1  import React from "react";
2  import ReactDOM from "react-dom/client";
3  import { UIProvider } from "@yamada-ui/react";
4  import App from "./App";
5  import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
6  import { theme } from "./theme";
7
8  const query_client = new QueryClient();
9
10 ReactDOM.createRoot(document.getElementById("root")).render(
11   <React.StrictMode>
12     <UIProvider theme={theme}>
13       <QueryClientProvider client={query_client}>
14         <App />
15       </QueryClientProvider>
16     </UIProvider>
17   </React.StrictMode>,
18 );
```

2.4 アプリケーションメイン App.jsx

```
1  //import reactLogo from "./assets/react.svg";
2  import "./App.css";
3  import { createBrowserRouter ,createRoutesFromElements, Route, RouterProvider, } from
   ↪  "react-router-dom";
4
5  import BasePage from "./BasePage.jsx";
6  import TodoList from "./TodoList.jsx";
7  import AddTodo from "./AddTodo.jsx";
8  import Login from "./Login.jsx";
9  import RegistUser from "./RegistUser.jsx";
10 import Init from "./Init.jsx";
11 import EditTodo from "./EditTodo";
12 import PasteTodo from "./PasteTodo.jsx";
13
14 export const routes = createBrowserRouter(
15   createRoutesFromElements(
16     <>
17       <Route element={ <BasePage/> }>
18         <Route path="/" element={ <Init/> }/>
19         <Route path="/login" element={ <Login/> }/>
20         <Route path="/regist_user" element={ <RegistUser/> }/>
21         <Route path="/todo" element={ <TodoList/> }/>
22         <Route path="/addtodo" element={ <AddTodo/> }/>
23         <Route path="/edittodo/:id" element={ <EditTodo/> }/>
24         <Route path="/pastetodo/:id" element={ <PasteTodo/> }/>
25       </Route>
26     </>
27   ));
28
29 function App() {
30   return (
31     <RouterProvider router={routes}/>
32   );
33 }
34 export default App;
```

2.5 全体のベースページ BasePage.jsx

```
1  import { Outlet } from "react-router-dom";
2  import "./App.css";
3
4
5  function BasePage() {
6
7      return (
8          <>
9              <h1> 猫 todo </h1>
10             <Outlet/>
11          </>
12      );
13  }
14
15
16  export default BasePage;
```


2.6 アプリケーションの初期化 Init.jsx

```
1  /* アプリケーションの初期化 */
2  /* 有効なセッションがあれば、ログイン済みに */
3  /* でなければ、ログイン画面へ遷移 */
4
5  import { Container, Heading } from "@yamada-ui/react";
6  import { invoke } from "@tauri-apps/api/core";
7  import { useNavigate } from "react-router-dom";
8  import { useQuery } from "@tanstack/react-query";
9  import { useEffect } from "react";
10
11 function Init() {
12
13     const navi = useNavigate();
14
15     const { data, isFetching, isSuccess, isError, error } = useQuery({
16         queryKey: ['check_login'],
17         queryFn: async () => invoke('is_valid_session')
18     });
19
20     useEffect( () => {
21         if (isSuccess && !isFetching) {
22             if (data === true) {
23                 navi('/todo');
24             } else {
25                 navi('/login');
26             }
27         }
28     },[isSuccess, isFetching])
29
30     return (
31         <>
32             <Container centerContent>
33                 <Heading> ただいま、初期化中です。 </Heading>
34                 <p> しばらくお待ちください。 </p>
35                 <p> 現在、ログイン状態の検査中です。 </p>
36                 <p> { isError && "error 発生:" + error } </p>
37             </Container>
38         </>
39     );
40
41 }
42
43 export default Init;
```

2.7 ユーザー登録画面 RegistUser.jsx

```
1  /* ユーザー登録画面 */
2
3  import { useForm } from "react-hook-form";
4  import { VStack, FormControl, Input, Button, Text } from "@yamada-ui/react";
5  import { invoke } from "@tauri-apps/api/core";
6  import { useState } from 'react';
7  import { useNavigate } from "react-router-dom";
8
9  function RegistUser() {
10     const { register, handleSubmit, formState: {errors} } = useForm();
11     const [ sendMessage, setSendMessage ] = useState('');
12     const navi = useNavigate();
13     const onSubmit = async (data) => {
14         try {
15             setSendMessage(' 送信中です。');
16             await invoke('regist_user', { name: data.name, password: data.pass });
17             navi('/login');
18         } catch (e) {
19             setSendMessage(' エラーが発生しました。{' + e + '}');
20             console.log(e);
21         }
22     };
23
24     return (
25         <>
26         <h1> 新規ユーザー登録 </h1>
27         <p> すべての欄を入力してください。 </p>
28         <VStack as="form" onSubmit={handleSubmit(onSubmit)}>
29             <FormControl
30                 isValid={!!errors.name}
31                 label="ユーザー名"
32                 errorMessage={errors?.name?.message}
33             >
34                 <Input {...register("name", {required: "入力が必要です。"})}/>
35             </FormControl>
36             <FormControl
37                 isValid={!!errors.pass}
38                 label="パスワード"
39                 errorMessage={errors?.pass?.message}
40             >
41                 <Input {...register("pass", {required: "入力が必要です。"})}/>
42             </FormControl>
43             <Button type="submit"> 送信 </Button>
44             <Text>{sendMessage}</Text>
```

```
45         </VStack>
46     </>
47 );
48 }
49
50 export default RegisterUser;
51
```

2.8 ログイン画面 Login.jsx

```
1  /* ログイン画面 */
2
3  import { useForm } from "react-hook-form";
4  import { VStack, FormControl, Input, Button, Text } from "@yamada-ui/react";
5  import { invoke } from "@tauri-apps/api/core";
6  import { Link, useNavigate } from "react-router-dom";
7  import { useState } from "react";
8  import {useQueryClient} from "@tanstack/react-query";
9
10 function Login() {
11     const { register, handleSubmit, formState: {errors} } = useForm();
12     const [ sendMessage, setSendMessage ] = useState('');
13     const navi = useNavigate();
14     const queryClient = useQueryClient();
15
16     const onSubmit = async (data) => {
17         try {
18             setSendMessage(' 処理中です。 ');
19             await invoke('login', { name: data.name, password: data.pass });
20             queryClient.invalidateQueries("check_login");
21             navi('/');
22         } catch (e) {
23             setSendMessage(' エラーが発生しました。{' + e + '}');
24             console.log(e);
25         }
26     };
27
28     return (
29         <>
30         <Link to="/regist_user">新規ユーザー登録</Link>
31         <h1> ログイン </h1>
32         <VStack as="form" onSubmit={handleSubmit(onSubmit)}>
33             <FormControl
34                 isValid={!!errors.name}
35                 label="ユーザー名"
36                 errorMessage={errors?.name?.message}
37             >
38                 <Input {...register("name", {required: " 入力は必須です。" })}/>
39             </FormControl>
40             <FormControl
41                 isValid={!!errors.pass}
42                 label="パスワード"
43                 errorMessage={errors?.pass?.message}
44             >
```

```

45         <Input {...register("pass", {required: "入力必須です。"})}/>
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

```
1  import { useQuery, } from "@tanstack/react-query";
2  import { Container, Grid, GridItem, } from "@yamada-ui/react";
3  import { invoke } from "@tauri-apps/api/core";
4  import "./App.css";
5  import TodoItem from "./TodoItem.jsx";
6  import TodoItemToolbar from "./TodoListToolbar.jsx";
7
8  const get_todo_list = async () => invoke('get_todo_list') ;
9
10 function TodoList() {
11
12     const { data: todos, isLoading: isTodoListLoading , isError, error} = useQuery({
13         queryKey: ['todo_list'],
14         queryFn: get_todo_list,
15     });
16
17     if (isTodoListLoading) {
18         return ( <p> loading... </p>);
19     }
20
21     if (isError) {
22         return ( <p> エラーだよ。{error}</p> );
23     }
24
25     console.log(todos);
26     return (
27         <>
28             <Container gap="0" bg="background">
29                 <TodoItemToolbar/>
30
31                 <h1>現在の予定</h1>
32                 <Grid templateColumns="repeat(4, 1fr)" gap="md" >
33                     {todos?.map( todo_item => {
34                         return (
35                             <GridItem key={todo_item.id} w="full" rounded="md" bg="primary">
36                                 <TodoItem item={todo_item}/>
37                             </GridItem>
38                         )}
39                     )}
40                 </Grid>
41             </Container>
42         </>
43     );
44 }
```

45

46

47 `export default` TodoList;

2.10 todo アイテム表示 TodoItem.jsx

```
1  // todo リストの各アイテム
2  import {useNavigate} from "react-router-dom";
3  import {useMutation, useQueryClient} from "@tanstack/react-query";
4  import {invoke} from "@tauri-apps/api/core";
5  import { SimpleGrid, GridItem, IconButton, Text, HStack, Container } from "@yamada-ui/react";
6  import { GrWorkshop } from "react-icons/gr";
7  import { BsAlarm } from "react-icons/bs";
8  import { BsEmojiGrin } from "react-icons/bs";
9  import { BiPencil } from "react-icons/bi";
10 import { FaRegCopy } from "react-icons/fa6";
11
12 export default function TodoItem({item}) {
13     const navi = useNavigate();
14     const queyrClient = useQueryClient();
15     const {mutate} = useMutation({
16         mutationFn: () => {
17             return invoke("update_done", {id: item.id, done: !item.done});
18         },
19         onSuccess: () => {
20             queyrClient.invalidateQueries({ queryKey: ["todo_list"]});
21         }
22     });
23
24     const onEditClick = () => {
25         navi("/edittodo/"+item.id);
26     }
27
28     const onPasteClick = () => {
29         navi("/pastetodo/"+item.id);
30     }
31
32     const onDoneClick = () => {
33         console.log(item.id + " : " + item.title);
34         mutate();
35     }
36
37     // 日付の表示内容生成
38     let end_date = new Date(item.end_date);
39     if (item.end_date === "9999-12-31") {
40         end_date = null;
41     }
42     const start_date = new Date(item.start_date);
43     const update_date = new Date(item.update_date);
44 }
```



```

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}

```

```

91         </Text>
92         <Text fontSize="sm">
93             {item.work}
94         </Text>
95         <Text fontSize="sm">
96             {start_date?.toLocaleDateString()} {end_date?.toLocaleDateString()}
97         </Text>
98     </Container>
99 );
100 }
101
102 function geDate(val1, val2) {
103     const year1 = val1.getFullYear();
104     const month1 = val1.getMonth();
105     const day1 = val1.getDate();
106     const year2 = val2.getFullYear();
107     const month2 = val2.getMonth();
108     const day2 = val2.getDate();
109
110     if (year1 === year2) {
111         if (month1 === month2) {
112             return day1 >= day2;
113         } else {
114             return month1 > month2;
115         }
116     } else {
117         return year1 > year2;
118     }
119 }
120

```

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

```
1  import { useNavigate } from "react-router-dom";
2  import { useMutation, useQuery, useQueryClient } from "@tanstack/react-query";
3  import { HStack, IconButton, Select, Switch, Option } from "@yamada-ui/react";
4  import { invoke } from "@tauri-apps/api/core";
5  import { AiOutlineFileAdd } from "react-icons/ai";
6  import "./App.css";
7
8
9  export default function TodoListToolbar() {
10
11      const navi = useNavigate();
12      const handleAddTodo = () => navi('/addtodo');
13
14      return (
15          <>
16              <HStack>
17                  <IconButton icon={<AiOutlineFileAdd/>} onClick={handleAddTodo}/>
18                  <SwitchIncomplete/>
19                  <SelectItemSortOrder/>
20              </HStack>
21          </>
22      );
23  }
24
25  function SwitchIncomplete() {
26      const queryClient = useQueryClient();
27      const {data: IsIncomplete, isPending} = useQuery({
28          queryKey: ['is_incomplete'],
29          queryFn: () => invoke('get_is_incomplete') ,
30      });
31
32      const {mutate} = useMutation({
33          mutationFn: (checked) => invoke('set_is_incomplete', {isIncomplete: checked}) ,
34          onSuccess: () => {
35              queryClient.invalidateQueries({queryKey: ['is_incomplete']});
36              queryClient.invalidateQueries({queryKey: ['todo_list']});
37          }
38      });
39
40      const onIsIncompleteChange = (e) => mutate(e.target.checked) ;
41
42      if (isPending) {
43          return <p> Loading... </p>;
44      }
```

```

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

```
1  import { invoke } from "@tauri-apps/api/core";
2  import { str2date } from "../str2date.jsx";
3  import { InputTodo } from "../InputTodo.jsx";
4
5  function AddTodo() {
6
7      const send_data = async (data) => {
8          const res = {item : {
9              title : data.title,
10             work : data.work,
11             start : str2date(data.start)?.toLocaleDateString(),
12             end : str2date(data.end)?.toLocaleDateString(),
13         }};
14         await invoke('add_todo', res);
15     };
16
17     const init_val = {
18         title : "",
19         work : "",
20         start : "",
21         end : "",
22     };
23
24     return (
25         <>
26         <InputTodo send_data={send_data} init_val={init_val}/>
27         </>
28     );
29 }
30
31 export default AddTodo;
```

2.13 todo アイテムの編集 EditTodo.jsx

```
1  import {useQuery} from "@tanstack/react-query";
2  import {useParams} from "react-router-dom";
3  import {invoke} from "@tauri-apps/api/core";
4
5  import {InputTodo} from "../InputTodo.jsx";
6  import { str2date } from "../str2date.jsx";
7
8  export default function EditTodo() {
9      const { id } = useParams();
10
11     const { data: todo, isLoading, isError, error} = useQuery({
12         queryKey: ['todo_item_'+id],
13         queryFn: async () => invoke('get_todo_with_id', {id: Number(id)}),
14     });
15
16     const handleSendData = async (data) => {
17         const res = {
18             id: Number(id),
19             item: {
20                 title: data.title,
21                 work: data.work,
22                 start: str2date(data.start)?.toLocaleDateString(),
23                 end: str2date(data.end)?.toLocaleDateString(),
24             }
25         };
26         await invoke("edit_todo", res);
27     };
28
29     if (isLoading) {
30         return ( <p> loading... </p> );
31     }
32
33     if (isError) {
34         return ( <p> Error: {error} </p> );
35     }
36
37     const initForm = {
38         title: todo.title,
39         work: todo.work,
40         start: todo.start_date?.replace(/-/g, "/"),
41         end: todo.end_date=== "9999-12-31" ? "" : todo.end_date.replace(/-/g, "/"),
42     }
43
44     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

```
1  import {useQuery} from "@tanstack/react-query";
2  import {useParams} from "react-router-dom";
3  import {invoke} from "@tauri-apps/api/core";
4
5  import {InputTodo} from "../InputTodo.jsx";
6  import { str2date } from "../str2date.jsx";
7
8  export default function PasteTodo() {
9      const { id } = useParams();
10
11     const { data: todo, isLoading, isError, error} = useQuery({
12         queryKey: ['todo_item_'+id],
13         queryFn: async () => invoke('get_todo_with_id', {id: Number(id)}),
14     });
15
16     const handleSendData = async (data) => {
17         const res = {
18             item: {
19                 title: data.title,
20                 work: data.work,
21                 start: str2date(data.start)?.toLocaleDateString(),
22                 end: str2date(data.end)?.toLocaleDateString(),
23             }
24         };
25         await invoke("add_todo", res);
26     };
27
28     if (isLoading) {
29         return ( <p> loading... </p> );
30     }
31
32     if (isError) {
33         return ( <p> Error: {error} </p> );
34     }
35
36     const initForm = {
37         title: todo.title,
38         work: todo.work,
39         start: todo.start_date?.replace(/-/g, "/"),
40         end: todo.end_date=== "9999-12-31" ? "" : todo.end_date.replace(/-/g, "/"),
41     }
42
43     return (
44         <>
```



```
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

```
1 import { FormProvider, useForm, useFormContext } from "react-hook-form";
2 import { Button, FormControl, HStack, Input, Text, Textarea, VStack } from "@yamada-ui/react";
3 import { useEffect, useState } from "react";
4 import { useNavigate } from "react-router-dom";
5 import { useMutation } from "@tanstack/react-query";
6 import { str2date } from "../str2date.jsx";
7
8 export function InputTodo({send_data, init_val}) {
9   const form = useForm({
10     defaultValues: {
11       title: init_val.title,
12       work: init_val.work,
13       start: init_val.start,
14       end: init_val.end
15     },
16   });
17   const { register, handleSubmit, formState: {errors} } = form;
18   const [ errorMessage, setErrorMessage ] = useState("");
19   const navi = useNavigate();
20
21   const {mutate, isPending} = useMutation( {
22     mutationFn: (data) => send_data(data),
23     onSuccess: () => navi('/'),
24     onError: (error) => setErrorMessage(error),
25   });
26
27   const onCancelClick = () => { navi('/'); };
28
29   return (
30     <>
31       <FormProvider {...form}>
32         <VStack as="form" onSubmit={handleSubmit((data)=>mutate(data))>
33           <FormControl
34             invalid={!!errors.title}
35             label="タイトル"
36             errorMessage={errors?.title?.message}
37           >
38             <Input placeholder="やること"
39               {...register("title", {required:"入力必須です。"})}/>
40           </FormControl>
41           <FormControl label="詳細">
42             <Textarea {...register("work")} />
43           </FormControl>
44           <InputDate name="start" label="開始"/>
```

```

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

```

91 }

92

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

```
1 // 日付処理ユーティリティ
2
3 export function str2date(str) {
4     if (str == null) { return null; }
5     if (str.length === 0) { return null; }
6     const date_item = str.split('/');
7     for (const s of date_item) {
8         if (Number.isNaN(Number(s))) { return null; }
9     }
10
11     const cur_date = new Date();
12     const cur_year = cur_date.getFullYear();
13
14     let ret_date = cur_date;
15     try {
16         switch (date_item.length) {
17             case 0:
18                 return null;
19             case 1:
20                 if (date_item[0][0] == '+') {
21                     ret_date.setDate(ret_date.getDate() + Number(date_item[0]));
22                 } else {
23                     ret_date.setDate(Number(date_item[0]));
24                     if (ret_date < new Date()) {
25                         ret_date.setMonth(ret_date.getMonth() + 1);
26                     }
27                 }
28
29                 break;
30             case 2:
31                 ret_date = new Date(cur_year, Number(date_item[0])-1, Number(date_item[1]))
32                 if (ret_date < new Date()) {
33                     ret_date.setFullYear(ret_date.getFullYear() + 1);
34                 }
35                 break;
36             case 3:
37                 const year = Number(date_item[0]);
38                 const month = Number(date_item[1]);
39                 const date = Number(date_item[2]);
40                 ret_date = new Date(year, month-1, date);
41                 break;
42             default:
43                 return null;
44         }
45     }
```

```
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

```
1  # 猫todo 関係のすべての mariadb オブジェクトの生成
2
3  create database if not exists nekotodo;
4
5  use nekotodo;
6
7  create table if not exists users (
8      name varchar(128) primary key,
9      password varchar(61)
10 );
11
12 create table if not exists todo (
13     id int unsigned auto_increment primary key,
14     user_name varchar(128) not null references users(name),
15     title varchar(128) not null,
16     work varchar(2048),
17     update_date date not null,
18     start_date date not null,
19     end_date date not null,
20     done bool not null
21 );
22
23 create table if not exists tag (
24     name varchar(128) primary key
25 );
26
27 create table if not exists todo_tag (
28     todo_id int unsigned references todo(id),
29     tag_name varchar(128) references tag(name),
30     primary key(todo_id, tag_name)
31 );
32
33 create table if not exists sessions (
34     id varchar(40) primary key,
35     user_name varchar(128) references users(name),
36     expired timestamp default date_add(current_timestamp, interval 48 hour)
37 );
38
```