

neko_todo ソースリスト

美都

2025 年 2 月 22 日

目次

1	Rust ソース	3
1.1	メインモジュール main.rs	3
1.2	アプリケーションステータス app_status.rs	6
1.3	コンフィグ設定処理 config.rs	7
1.4	アプリケーション設定情報の処理 setup.rs	14
1.5	todo モデル処理 todo.rs	17
1.6	データベースアクセスモジュール database.rs	29
1.6.1	新規作成 new.rs	31
1.6.2	セッション管理 session.rs	32
1.6.3	todo アイテム管理 todo.rs	35
1.6.4	ユーザーデータ管理 user.rs	38
1.6.5	データベースモジュールテスト test.rs	40
1.7	command モジュール tauri::command 関数群 command.rs	53
1.7.1	todo リストの表示・編集 todo.rs	54
1.7.2	ユーザー操作関係 user.rs	58
1.7.3	セッション操作関連 session.rs	59
1.7.4	アプリケーションの状態操作 app_state.rs	61
2	フロントエンド React 関係	63
2.1	index.html	63
2.2	メイン CSS ファイル	64
2.3	main.jsx	67
2.4	アプリケーションメイン App.jsx	68
2.5	全体のベースページ BasePage.jsx	69
2.6	アプリケーションの初期化 Init.jsx	70
2.7	ユーザー登録画面 RegistUser.jsx	71
2.8	ログイン画面 Login.jsx	73
2.9	todo リストの表示 TodoList.jsx	75
2.10	todo アイテム表示 TodoItem.jsx	77
2.11	todo リスト画面 ツールバー TodoListToolbar.jsx	80
2.12	todo アイテムの追加 AddTodo.jsx	82
2.13	todo アイテムの編集 EditTodo.jsx	83
2.14	todo アイテムの複製 PasteTodo.jsx	85
2.15	todo アイテム内容の入力フォーム InputTodo.jsx	87

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	90
3	データベース構成	92
3.1	テーブル生成スクリプト create_table.sql	92

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     setup_log();
25     run()
26 }
27
28 /// ロギング機構のセットアップ
29 fn setup_log() {
30     let mut log_file: std::path::PathBuf = ProjectDirs::from("jp", "laki", "nekotodo")
31         .unwrap()
32         .config_dir()
33         .into();
34     if !log_file.exists() {
35         std::fs::create_dir_all(&log_file).unwrap();
36     }
37     log_file.push("nekotodo.log");
38
39     fern::Dispatch::new()
40         .format(|out, message, record| {
41             out.finish(format_args!(
```

```

42         "{} [{}] {}:{} {}",
43         chrono::Local::now().format("%Y/%m/%d %H:%M:%S"),
44         record.level(),
45         record.file().unwrap(),
46         record.line().unwrap(),
47         message
48     ))
49 })
50 .level(log::LevelFilter::Info)
51 //.level(log::LevelFilter::Debug)
52 .chain(std::io::stderr())
53 .chain(fern::log_file(log_file).unwrap())
54 .apply()
55 .unwrap();
56 }
57
58 /// アプリケーション本体部分
59 #[cfg_attr(mobile, tauri::mobile_entry_point)]
60 fn run() {
61     let app_status = match setup() {
62         Ok(s) => s,
63         Err(e) => {
64             error!("{}", e);
65             std::process::exit(1)
66         }
67     };
68
69     let app = tauri::Builder::default()
70         .plugin(tauri_plugin_shell::init())
71         .manage(app_status)
72         .invoke_handler(tauri::generate_handler![
73             get_todo_list,
74             get_todo_with_id,
75             regist_user,
76             login,
77             is_valid_session,
78             add_todo,
79             update_done,
80             edit_todo,
81             set_is_incomplete,
82             get_is_incomplete,
83             set_item_sort_order,
84             get_item_sort_order,
85         ])
86         .build(tauri::generate_context!())
87         .expect("error thile build tauri application");

```

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

1.2 アプリケーションステータス app_status.rs

```
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     unsafe {
270         std::env::remove_var(DB_HOST);
271         std::env::remove_var(DB_USER);
272         std::env::remove_var(DB_USER);
273     }
274 }

```


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;

```

```

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

```

```

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モジュール database.rs

```
1  /// データベースの操作を司る
2  mod new;
3  mod session;
4  #[cfg(test)]
5  mod test;
6  mod todo;
7  mod user;
8
9  use chrono::NaiveDate;
10 use log::error;
11 use serde::{Deserialize, Serialize};
12 use sqlx::{
13     mysql::{MySQLPool, MySQLPoolOptions},
14     prelude::*,
15 };
16 use thiserror::Error;
17
18 /// neko_db データベース操作関数郡
19 #[derive(Clone, Debug)]
20 pub struct Database {
21     pool: MySQLPool,
22 }
23
24 #[derive(FromRow, Debug, PartialEq)]
25 pub struct User {
26     pub name: String,
27     pub password: String,
28 }
29
30 #[derive(FromRow, Serialize, Deserialize, Debug, PartialEq, Clone)]
31 pub struct ItemTodo {
32     pub id: u32,
33     pub user_name: String,
34     pub title: String,
35     pub work: Option<String>,
36     pub update_date: Option<NaiveDate>,
37     pub start_date: Option<NaiveDate>,
38     pub end_date: Option<NaiveDate>,
39     pub done: bool,
40 }
41
42 #[derive(Error, Debug)]
43 pub enum DbError {
44     #[error("データベースへの接続に失敗。")]
```

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

1.6.1 新規作成 new.rs

```
1  ///! database 構造体新規作成
2
3  use super::*;
4
5  impl Database {
6      /// 新規生成。
7      pub async fn new(host: &str, user: &str, pass: &str) -> Result<Self, DbError> {
8          let db_url = format!("mariadb://{}:{}/nekotodo", user, pass, host);
9          let pool = MySQLPoolOptions::new()
10              .max_connections(10)
11              .min_connections(3)
12              .connect(&db_url)
13              .await
14              .map_err(DbError::FailConnect)?;
15          Ok(Self { pool })
16      }
17  }
```

1.6.2 セッション管理 session.rs

```
1  /// セッション情報の操作
2  use super::*;
3  use sqlx::{prelude::*, query};
4  use uuid::Uuid;
5
6  impl Database {
7      /// セッション情報を新規作成する。
8      /// 生成した uuid を返す。
9      pub async fn make_new_session(&self, user_name: &str) -> Result<Uuid, DbError> {
10         let sql = "insert into sessions(id, user_name) values (?,?)";
11         // キー情報の作成
12         let id = Uuid::now_v7();
13
14         query(sql)
15             .bind(id.to_string())
16             .bind(user_name)
17             .execute(&self.pool)
18             .await
19             .map_err(|err| match err {
20                 sqlx::Error::Database(ref e) => {
21                     if e.is_foreign_key_violation() {
22                         // 外部キーエラー。存在しないユーザーを指定した。
23                         return DbError::NotFoundUser;
24                     }
25                     DbError::FailDbAccess(err)
26                 }
27                 _ => DbError::FailDbAccess(err),
28             })?;
29
30         Ok(id)
31     }
32
33     /// 指定されたセッションを新規セッションに更新する。
34     /// 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。
35     pub async fn update_session(&self, id: &uuid::Uuid) -> Result<Uuid, DbError> {
36         let mut tr = self.pool.begin().await.map_err(DbError::FailDbAccess)?;
37         // 期限切れのセッション削除
38         let sql_old_del = "delete from sessions where expired < now()";
39         query(sql_old_del)
40             .execute(&mut *tr)
41             .await
42             .map_err(DbError::FailDbAccess)?;
43
44         // ユーザー ID の特定
45         let sql_query_user = "select user_name from sessions where id=?";
```



```

46     let user: String = query(sql_query_user)
47         .bind(id.to_string())
48         .fetch_one(&mut *tr)
49         .await
50         .map_err(|e| match e {
51             sqlx::Error::RowNotFound => DbError::NotFoundSession,
52             e => DbError::FailDbAccess(e),
53         })?
54         .get("user_name");
55
56     // 旧セッションの削除
57     let sql_del_curr_sess = "delete from sessions where id = ?;";
58     query(sql_del_curr_sess)
59         .bind(id.to_string())
60         .execute(&mut *tr)
61         .await
62         .map_err(DbError::FailDbAccess)?;
63
64     // 新セッションの生成
65     let sql_create_sess = "insert into sessions(id, user_name) values (?, ?);";
66     let id = Uuid::now_v7();
67     query(sql_create_sess)
68         .bind(id.to_string())
69         .bind(user)
70         .execute(&mut *tr)
71         .await
72         .map_err(DbError::FailDbAccess)?;
73
74     tr.commit().await.map_err(DbError::FailDbAccess)?;
75     Ok(id)
76 }
77
78 /// 指定されたセッション ID が有効であるか確認する。
79 /// データベースエラーが発生した場合は、Err(DbError::FailDbAccess) を返す。
80 pub async fn is_session_valid(&self, sess: &Uuid) -> Result<bool, DbError> {
81     // 期限切れのセッションを削除する。
82     let sql_old_del = "delete from sessions where expired < now();";
83     query(sql_old_del)
84         .execute(&self.pool)
85         .await
86         .map_err(DbError::FailDbAccess)?;
87     // 指定セッション ID の有無を確認する。
88     let sql_find_sess = "select count(*) as cnt from sessions where id = ?;";
89     let sess_cnt: i64 = query(sql_find_sess)
90         .bind(sess.to_string())
91         .fetch_one(&self.pool)

```

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

1.6.3 todo アイテム管理 todo.rs

```
1  ///! todo アイテム操作
2  use super::*;
3  use crate::config::ItemSortOrder;
4  use chrono::{Local, NaiveDate};
5  use sqlx::{query, query_as};
6  use uuid::Uuid;
7
8  impl Database {
9      /// Todo 項目を追加する。
10     /// item 引数のうち、id, update_date, done は、無視される
11     /// 各々、自動値・今日の日付・false がはいる。
12     /// start_date, end_date のデフォルト値は、今日・NaiveDate::MAX である。
13     pub async fn add_todo_item(&self, item: &ItemTodo) -> Result<(), DbError> {
14         let sql = r#"
15             insert into todo(user_name, title, work, update_date, start_date, end_date, done)
16             values (?, ?, ?, curdate(), ?, ?, false);
17         "#;
18         let start_date = item.start_date.unwrap_or(Local::now().date_naive());
19         let end_date = item
20             .end_date
21             .unwrap_or(NaiveDate::from_ymd_opt(9999, 12, 31).unwrap());
22         query(sql)
23             .bind(&item.user_name)
24             .bind(&item.title)
25             .bind(&item.work)
26             .bind(start_date)
27             .bind(end_date)
28             .execute(&self.pool)
29             .await
30             .map_err(DbError::FailDbAccess)?;
31         Ok(())
32     }
33
34     /// Todo の一覧を取得する。
35     /// 基準日 (ref_date) 以降のアイテムを選別する。
36     /// セッション ID を必要とする。
37     /// 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。
38     pub async fn get_todo_item(
39         &self,
40         sess: Uuid,
41         ref_date: NaiveDate,
42         only_incomplete: bool,
43         sort_order: ItemSortOrder,
44     ) -> Result<Vec<ItemTodo>, DbError> {
45         let sql1 = r#"

```

```

46         select t.id, t.user_name, title, work, update_date, start_date, end_date, done
47         from todo t join sessions s on s.user_name = t.user_name
48         where s.id=? and t.start_date <= ?
49         "#;
50     let sql2 = " and done = false";
51     let sql3 = match sort_order {
52         ItemSortOrder::EndAsc => " order by end_date, update_date",
53         ItemSortOrder::EndDesc => " order by end_date desc, update_date",
54         ItemSortOrder::StartAsc => " order by start_date, update_date",
55         ItemSortOrder::StartDesc => " order by start_date desc, update_date",
56         ItemSortOrder::UpdateAsc => " order by update_date, end_date",
57         ItemSortOrder::UpdateDesc => " order by update_date desc, end_date",
58     };
59     let sql = if only_incomplete {
60         format!("{}", {} {};", sql1, sql2, sql3)
61     } else {
62         format!("{}", {};", sql1, sql3)
63     };
64     let items = query_as::<_, ItemTodo>(&sql)
65         .bind(sess.to_string())
66         .bind(ref_date)
67         .fetch_all(&self.pool)
68         .await
69         .map_err(DbError::FailDbAccess)?;
70
71     Ok(items)
72 }
73
74 /// 指定 id の Todo 項目を取得する。
75 /// 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。
76 pub async fn get_todo_item_with_id(&self, id: u32, sess: Uuid) -> Result<ItemTodo, DbError> {
77     let sql = r#"
78         select t.id, t.user_name, t.title, t.work, t.update_date, t.start_date, t.end_date,
79         ↪ t.done
80         from todo t join sessions s on s.user_name = t.user_name
81         where s.id=? and t.id=?
82         "#;
83     query_as::<_, ItemTodo>(sql)
84         .bind(sess.to_string())
85         .bind(id)
86         .fetch_one(&self.pool)
87         .await
88         .map_err(|e| match e {
89             sqlx::Error::RowNotFound => DbError::NotFoundTodo,
90             e => DbError::FailDbAccess(e),
91         })

```

```

91     }
92
93     /// Todo の完了状態を更新する。
94     pub async fn change_done(&self, id: u32, done: bool) -> Result<(), DbError> {
95         let sql = "update todo set done = ? where id = ?";
96         let res = query(sql)
97             .bind(done)
98             .bind(id)
99             .execute(&self.pool)
100             .await
101             .map_err(DbError::FailDbAccess)?;
102         if res.rows_affected() > 0 {
103             Ok(())
104         } else {
105             Err(DbError::NotFoundTodo)
106         }
107     }
108
109     /// Todo の項目編集
110     pub async fn edit_todo(&self, item: &ItemTodo) -> Result<(), DbError> {
111         let start_date = item.start_date.unwrap_or(Local::now().date_naive());
112         let end_date = item
113             .end_date
114             .unwrap_or(NaiveDate::from_ymd_opt(9999, 12, 31).unwrap());
115
116         let sql = r#"
117             update todo
118             set title=?, work=?, update_date=curdate(), start_date=?, end_date=?
119             where id=?;
120             "#;
121         let res = query(sql)
122             .bind(&item.title)
123             .bind(&item.work)
124             .bind(start_date)
125             .bind(end_date)
126             .bind(item.id)
127             .execute(&self.pool)
128             .await
129             .map_err(DbError::FailDbAccess)?;
130         if res.rows_affected() > 0 {
131             Ok(())
132         } else {
133             Err(DbError::NotFoundTodo)
134         }
135     }
136 }

```

1.6.4 ユーザーデータ管理 user.rs

```
1  /// ユーザーデータの操作
2  use super::*;
3  use sqlx::{query, query_as};
4  use uuid::Uuid;
5
6  impl Database {
7      /// ユーザーの追加
8      pub async fn add_user(&self, name: &str, pass: &str) -> Result<(), DbError> {
9          let sql = "insert into users(name, password) values (?, ?)";
10         query(sql)
11             .bind(name)
12             .bind(pass)
13             .execute(&self.pool)
14             .await
15             .map_err(|e| match e {
16                 sqlx::Error::Database(ref db_err) => {
17                     if db_err.kind() == sqlx::error::ErrorKind::UniqueViolation {
18                         DbError::DuplicateUserName(e)
19                     } else {
20                         DbError::FailDbAccess(e)
21                     }
22                 }
23                 _ => DbError::FailDbAccess(e),
24             })?;
25         Ok(())
26     }
27
28     /// ユーザー名をキーとして、ユーザー情報を取得
29     pub async fn get_user(&self, name: &str) -> Result<User, DbError> {
30         let sql = "select name, password from users where name = ?";
31         query_as(sql)
32             .bind(name)
33             .fetch_one(&self.pool)
34             .await
35             .map_err(|e| match e {
36                 sqlx::Error::RowNotFound => DbError::NotFoundUser,
37                 e => DbError::FailDbAccess(e),
38             })
39     }
40
41     /// セッション ID をキーにしてユーザー情報を取得
42     pub async fn get_user_from_sess(&self, sess: Uuid) -> Result<User, DbError> {
43         let sql = r#"
44             select u.name, u.password
45             from users u join sessions s on u.name=s.user_name
```

```

46         where s.id = ?;
47         "#;
48
49     query_as(sql)
50         .bind(sess.to_string())
51         .fetch_one(&self.pool)
52         .await
53         .map_err(|e| match e {
54             sqlx::Error::RowNotFound => DbError::NotFoundSession,
55             e => DbError::FailDbAccess(e),
56         })
57     }
58 }

```

1.6.5 データベースモジュールテスト test.rs

```
1  /// database モジュールテスト
2
3  use crate::config::ItemSortOrder;
4  use chrono::{Days, Local};
5  use sqlx::query;
6  use uuid::Uuid;
7
8  use super::*;
9
10 /// テスト用の Database 生成。テスト用 Pool をインジェクション
11 impl Database {
12     pub(crate) fn new_test(pool: MySqlPool) -> Self {
13         Self { pool }
14     }
15 }
16
17 /// ユーザー生成のテスト
18 #[sqlx::test]
19 async fn test_add_user_and_get_user(pool: MySqlPool) {
20     let db = Database::new_test(pool);
21     db.add_user("hyara", "password").await.unwrap();
22     let user = db.get_user("hyara").await.unwrap();
23     assert_eq!(user.name, "hyara");
24     assert_eq!(user.password, "password");
25     let error_user = db.get_user("naiyo").await;
26     match error_user {
27         Ok(_) => unreachable!("結果が帰ってくるはずがない。"),
28         Err(DbError::NotFoundUser) => { /* 正常 */ }
29         Err(e) => unreachable!("このエラーはおかしい。{e}"),
30     }
31 }
32
33 /// セッション生成関係の一連のテスト。
34 #[sqlx::test]
35 async fn test_make_new_session(pool: MySqlPool) {
36     println!("まずはテスト用のユーザーの生成");
37     let db = Database::new_test(pool);
38     let user_name = "nekodayo";
39     let password = "password";
40     db.add_user(user_name, password).await.unwrap();
41
42     println!("次に、普通にセッションを作ってみる。");
43     let sess1 = db.make_new_session(user_name).await.unwrap();
44     println!("セッション生成成功 id={}", sess1);
45 }
```



```

46     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。");
47     let sess2 = db.make_new_session("detarame").await;
48     match sess2 {
49         Ok(_) => unreachable!("このユーザーは存在しなかったはず。"),
50         Err(DbError::NotFoundUser) => { /* 正常 */ }
51         Err(e) => unreachable!("このエラーもおかしい。[{}]", e),
52     }
53
54     println!("普通に、セッションを更新してみる。");
55     let sess3 = db.update_session(&sess1).await.unwrap();
56     assert_ne!(sess1, sess3);
57
58     println!("ないはずのセッションを更新しようとしてみる。");
59     let sess4 = Uuid::now_v7();
60     let sess5 = db.update_session(&sess4).await;
61     match sess5 {
62         Ok(_) => unreachable!("このセッションはないはずなのに。"),
63         Err(DbError::NotFoundSession) => { /* 正常 */ }
64         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}"),
65     }
66 }
67
68 /// セッションが有効かどうかを確認するテスト
69 #[sqlx::test]
70 async fn test_is_session_valid(pool: MySqlPool) {
71     let db = Database::new_test(pool);
72
73     println!("テスト用ユーザーの作成");
74     let name = "nekodayo";
75     let pass = "nekodamon";
76     db.add_user(name, pass).await.unwrap();
77
78     println!("新規セッションを生成する。");
79     let sess = db.make_new_session(name).await.unwrap();
80     println!("生成したセッション ID は、[{}] です。", &sess);
81
82     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。");
83     assert!(db.is_session_valid(&sess).await.unwrap());
84
85     println!("偽セッション ID をいれて、問い合わせしてみる。");
86     assert!(!db.is_session_valid(&Uuid::now_v7()).await.unwrap());
87 }
88
89 /// todo の書き込みと、単純な読み出しのテスト
90 #[sqlx::test]
91 async fn test_add_todo(pool: MySqlPool) {

```

```

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

```

```

138         done: true,
139     };
140     db.add_todo_item(&item).await.unwrap();
141
142     println!("テストデータを読み出す。一件しかないはず");
143     let last_day = Local::now().date_naive() + Days::new(1);
144     let res = db
145         .get_todo_item(sess, last_day, true, ItemSortOrder::EndAsc)
146         .await
147         .unwrap();
148     assert_eq!(res.len(), 1, "あれ?一件のはずだよ");
149     item.id = res[0].id;
150     item.update_date = Some(Local::now().date_naive());
151     item.done = false;
152 }
153
154 /// todo の書き込みと読み出し
155 /// done=true と false の挙動テスト
156 #[sqlx::test]
157 async fn test_get_todo_done_param(pool: MySqlPool) {
158     let db = Database::new_test(pool.clone());
159     let sess = login_for_test(&db).await;
160     let name = db.get_user_from_sess(sess).await.unwrap().name;
161
162     println!("テストデータをインサート");
163     let item = ItemTodo {
164         id: 0,
165         user_name: name.to_string(),
166         title: "インサートできるかな?".to_string(),
167         work: None,
168         update_date: None,
169         start_date: Some(Local::now().date_naive()),
170         end_date: Some(Local::now().date_naive() + Days::new(3)),
171         done: true,
172     };
173     db.add_todo_item(&item).await.unwrap();
174
175     println!("テストデータを読み出す。一件しかないはず");
176     let last_day = Local::now().date_naive() + Days::new(1);
177     let res = db
178         .get_todo_item(sess, last_day, false, ItemSortOrder::EndAsc)
179         .await
180         .unwrap();
181     assert_eq!(res.len(), 1, "全部読み出しだけど一件あるはず。");
182     let res = db
183         .get_todo_item(sess, last_day, true, ItemSortOrder::EndAsc)

```

```

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

```

```

230     let res = db
231         .get_todo_item(sess, ref_date + Days::new(1), true, ItemSortOrder::EndAsc)
232         .await
233         .unwrap();
234     assert_eq!(res.len(), 1, "開始日の翌日が基準日だからみつかる。");
235     let res = db
236         .get_todo_item(sess, ref_date - Days::new(1), true, ItemSortOrder::EndAsc)
237         .await
238         .unwrap();
239     assert_eq!(res.len(), 0, "基準日が開始日の前日だからみつからない。");
240     let res = db
241         .get_todo_item(sess, ref_date + Days::new(4), true, ItemSortOrder::EndAsc)
242         .await
243         .unwrap();
244     assert_eq!(res.len(), 1, "基準日が期限を過ぎているけどみつかるの。");
245 }
246
247 #[sqlx::test]
248 async fn test_get_user_from_sess(pool: MySqlPool) {
249     let db = Database::new_test(pool.clone());
250
251     let sess = login_for_test(&db).await;
252     let name = db.get_user_from_sess(sess).await.unwrap().name;
253
254     let user = db.get_user_from_sess(sess).await.unwrap();
255     assert_eq!(user.name, name, "これはみつかるはず");
256     let dummy_sess = Uuid::now_v7();
257     let user = db.get_user_from_sess(dummy_sess).await;
258     match user {
259         Ok(_) => unreachable!("見つかるわけないでしょう。"),
260         Err(DbError::NotFoundSession) => { /* 正常 */ }
261         Err(e) => unreachable!("トラブルです。{e}"),
262     };
263 }
264
265 #[sqlx::test]
266 async fn test_change_done(pool: MySqlPool) {
267     let db = Database::new_test(pool);
268     let sess = login_for_test(&db).await;
269     let ref_date = Local::now().date_naive();
270     create_todo_for_test(&db, sess).await;
271
272     let items = db
273         .get_todo_item(sess, ref_date, true, ItemSortOrder::EndAsc)
274         .await
275         .unwrap();

```

```

276 let item = items.iter().find(|&i| i.title.contains("二件目")).unwrap();
277 db.change_done(item.id, true).await.unwrap();
278
279 let items = db
280     .get_todo_item(sess, ref_date, true, ItemSortOrder::EndAsc)
281     .await
282     .unwrap();
283 let item = items.iter().find(|&i| i.title.contains("二件目"));
284 assert!(item.is_none(), "状態を完了にしたので見つからないはず。");
285
286 let items = db
287     .get_todo_item(sess, ref_date, false, ItemSortOrder::EndAsc)
288     .await
289     .unwrap();
290 let item = items.iter().find(|&i| i.title.contains("二件目"));
291 match item {
292     Some(i) => assert!(i.done, "完了済みになっているはずですね?"),
293     None => unreachable!("状態を変えたら、レコードなくなった???"),
294 }
295 assert_eq!(
296     items.len(),
297     3,
298     "全件見ているのでレコードは3件あるはずですが?"
299 );
300 }
301
302 #[sqlx::test]
303 async fn test_get_todo_with_id(pool: MySqlPool) {
304     let db = Database::new_test(pool);
305     let sess = login_for_test(&db).await;
306     create_todo_for_test(&db, sess).await;
307
308     let items = db
309         .get_todo_item(
310             sess,
311             Local::now().date_naive(),
312             false,
313             ItemSortOrder::EndAsc,
314         )
315         .await
316         .unwrap();
317     let id = items
318         .iter()
319         .find(|&i| i.title.contains("一件目"))
320         .expect("これはあるはず")
321         .id;

```

```

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

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

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

```
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::AppState;
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<'_, AppState>) -> 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<'_, AppState>,
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<'_, AppState>,
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::AppStatus;
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<'_, AppStatus>,
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<'_, AppStatus>,
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::AppState;
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<'_, AppState>) -> 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: &AppState) -> 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: &AppState) -> 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: "入力必須です。"})}/>
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: "入力必須です。"})}/>
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: "入力が必要です。"},)}>/>
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 queryClient = useQueryClient();
15     const {mutate} = useMutation({
16         mutationFn: () => {
17             return invoke("update_done", {id: item.id, done: !item.done});
18         },
19         onSuccess: () => {
20             queryClient.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。" : null} </Text>
51         <Text> {errorMessage} </Text>
52     </VStack>
53 </FormProvider>
54 </>
55 );
56 }
57
58
59 function InputDate({name, label}) {
60     const { register, watch, formState: {errors} } = useFormContext();
61
62     const val = watch(name);
63     const [ date, setDate, ] = useState(null);
64     useEffect(() => {
65         setDate(str2date(val));
66     }, [val]);
67
68     return (
69         <>
70             <FormControl
71                 invalid = {!!errors[name]}
72                 label={label}
73                 errorMessage={errors[name]?.message}>
74                 <HStack>
75                     <Input
76                         w="50%"
77                         placeholder="[[YYYY/]MM/]DD or +dd"
78                         {...register(name, {
79                             validate: (data) => {
80                                 if (data==null) { return }
81                                 if (data.length===0) { return }
82                                 if (str2date(data)==null) { return "日付の形式が不正です。" }
83                             }
84                         })}
85                     </>
86                     <Text> {date?.toLocaleDateString()} </Text>
87                 </HStack>
88             </FormControl>
89         </>
90     );

```


91 }

92

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

```
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
```