

neko_todo ソースリスト

美都

2025 年 1 月 21 日

目次

1	Rust ソース	2
1.1	メインモジュール main.rs	2
1.2	ライブラリメインモジュール lib.rs	3
1.3	アプリケーションステータス app_status.rs	8
1.4	コンフィグ設定処理 config.rs	9
1.5	アプリケーション設定情報の処理 setup.rs	14
1.6	todo モデル処理 todo.rs	17
1.7	データベースアクセス database.rs	28
2	フロントエンド React 関係	43
2.1	index.html	43
2.2	メイン CSS ファイル	44
2.3	main.jsx	47
2.4	アプリケーションメイン App.jsx	48
2.5	全体のベースページ BasePage.jsx	49
2.6	アプリケーションの初期化 Init.jsx	50
2.7	ユーザー登録画面 RegistUser.jsx	51
2.8	ログイン画面 Login.jsx	53
2.9	todo リストの表示 TodoList.jsx	55
2.10	todo アイテム表示 todoitem.jsx	57
2.11	todo アイテムの追加 AddTodo.jsx	59
2.12	todo アイテム内容の入力フォーム InputTodo.jsx	60
2.13	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	62
3	データベース構成	64
3.1	テーブル生成スクリプト create_table.sql	64

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 use env_logger::builder;
5
6 fn main() {
7     builder().init();
8
9     neko_todo_lib::run()
10 }
```

1.2 ライブラリメインモジュール lib.rs

```
1  /// tauri メインプロセス
2  mod app_status;
3  mod config;
4  mod database;
5  mod setup;
6  mod todo;
7
8  use app_status::AppStatus;
9  use database::ItemTodo;
10 use log::{error, info};
11 use serde::Deserialize;
12 use setup::setup;
13 use tauri::{command, Manager, State};
14 use todo::TodoError;
15 use uuid::Uuid;
16
17 #[tauri::command]
18 fn greet(name: &str) -> String {
19     format!("Hello, {}! You've been greeted from Rust!", name)
20 }
21
22 #[cfg_attr(mobile, tauri::mobile_entry_point)]
23 pub fn run() {
24     let app_status = match setup() {
25         Ok(s) => s,
26         Err(e) => {
27             error!("{}", e);
28             std::process::exit(1)
29         }
30     };
31
32     let app = tauri::Builder::default()
33         .plugin(tauri_plugin_shell::init())
34         .manage(app_status)
35         .invoke_handler(tauri::generate_handler![
36             greet,
37             get_todo_list,
38             regist_user,
39             login,
40             is_valid_session,
41             add_todo,
42             update_done,
43         ])
```

```

44     .build(tauri::generate_context!())
45     .expect("error thile build tauri application");
46 app.run(|app, event| match event {
47     tauri::RunEvent::Exit => {
48         eprintln!("終了処理開始");
49         let state = app.state::<AppState>();
50         state.config().lock().unwrap().save().unwrap();
51     }
52     _ => {}
53 });
54 }
55
56 /// todo のリストを取得する。
57 #[tauri::command]
58 async fn get_todo_list(app_status: State<'_, AppState>) -> Result<Vec<ItemTodo>, String> {
59     println!("todo 取得");
60     let sess = match get_curr_session(&app_status) {
61         Some(u) => u,
62         None => return Err("NotLogin".to_string()),
63     };
64
65     app_status
66         .todo()
67         .get_todo_list(sess, true)
68         .await
69         .map_err(|e| e.to_string())
70 }
71
72 /// todo を追加する。
73 #[tauri::command]
74 async fn add_todo(app_status: State<'_, AppState>, item: FormTodo) -> Result<(), String> {
75     let sess = match get_cur_session_with_update(&app_status).await {
76         Ok(Some(u)) => u,
77         Ok(None) => return Err("NotLogin".to_string()),
78         Err(e) => return Err(e),
79     };
80
81     eprintln!("input = {:?}", &item);
82     app_status
83         .todo()
84         .add_todo(sess, &item.into())
85         .await
86         .map_err(|e| match e {
87             todo::TodoError::NotFoundSession => "NotFoundSession".to_string(),
88             e => format!("OtherError: [{e}]"),

```

```

89         })
90     }
91
92     /// Todo 項目追加画面データ取得用
93     #[derive(Deserialize, Debug, Clone)]
94     struct FormTodo {
95         title: String,
96         work: Option<String>,
97         start: Option<String>,
98         end: Option<String>,
99     }
100
101     impl From<FormTodo> for ItemTodo {
102         fn from(val: FormTodo) -> Self {
103             let start = val.start.map(|d| d.replace("/", "-").parse().unwrap());
104             let end = val.end.map(|d| d.replace("/", "-").parse().unwrap());
105             ItemTodo {
106                 id: 0,
107                 user_name: "".to_string(),
108                 title: val.title,
109                 work: val.work,
110                 update_date: None,
111                 start_date: start,
112                 end_date: end,
113                 done: false,
114             }
115         }
116     }
117
118     /// todo の完了状態を変更する。
119     #[tauri::command]
120     async fn update_done(app_status: State<'_, AppStatus>, id: u32, done: bool) -> Result<(), String> {
121         let sess = match get_cur_session_with_update(&app_status).await {
122             Ok(Some(s)) => s,
123             Ok(None) => return Err("NotLogin".to_string()),
124             Err(e) => return Err(e),
125         };
126         app_status
127             .todo()
128             .change_done(id, sess, done)
129             .await
130             .map_err(|e| match e {
131                 TodoError::NotFoundTodo => "ignore_id".to_string(),
132                 e => format!("Database Error: [{e}]"),
133             })

```

```

134 }
135
136 /// ユーザー登録
137 #[tauri::command]
138 async fn regist_user(
139     app_status: State<'_, AppStatus>,
140     name: String,
141     password: String,
142 ) -> Result<(), String> {
143     match app_status.todo().add_user(&name, &password).await {
144         Ok(_) => Ok(()),
145         Err(e) => match e {
146             TodoError::DuplicateUser(_) => Err("DuplicateUser".to_string()),
147             TodoError::HashUserPassword(e) => Err(format!("InvalidPassword. [{}]", e)),
148             TodoError::FailDbAccess(e) => Err(e.to_string()),
149             _ => unimplemented!("[lib.rs regist_user] add_user 返回值異常"),
150         },
151     }
152 }
153
154 /// ログイン
155 #[command]
156 async fn login(
157     app_status: State<'_, AppStatus>,
158     name: String,
159     password: String,
160 ) -> Result<String, String> {
161     let session = app_status
162         .todo()
163         .login(&name, &password)
164         .await
165         .map_err(|e| match e {
166             TodoError::NotFoundUser => "NotFoundUser.".to_string(),
167             TodoError::HashUserPassword(e) => format!("InvalidPassword. [{}]", e),
168             TodoError::WrongPassword => "WrongPassword".to_string(),
169             TodoError::FailDbAccess(e) => format!("OtherError:{}", e),
170             _ => unimplemented!("[lib.rs::login] login から予期しないエラー"),
171         })?;
172
173     let mut cnf = app_status.config().lock().unwrap();
174     cnf.set_session_id(&session);
175     //cnf.save().map_err(|e| format!("OtherError:{}", e))?;
176     Ok(session.to_string())
177 }
178

```

```

179 /// 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)
180 #[command]
181 async fn is_valid_session(app_status: State<'_, AppStatus>) -> Result<bool, String> {
182     let sess = get_cur_session_with_update(&app_status)
183         .await
184         .map(|i| i.is_some());
185     match sess {
186         Ok(sess) => info!("セッション確認 ({})", if sess { "有効" } else { "無効" }),
187         Err(ref e) => info!("セッション確認エラー ({})", e),
188     }
189     sess
190 }
191
192 /// 現在、有効なセッションを返す。
193 /// 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。
194 async fn get_cur_session_with_update(app_status: &AppStatus) -> Result<Option<Uuid>, String> {
195     let cur_session = get_curr_session(app_status);
196     let Some(cur_session) = cur_session else {
197         return Ok(None);
198     };
199
200     match app_status.todo().is_valid_session(&cur_session).await {
201         Ok(Some(s)) => {
202             // 更新されたセッションを再登録
203             let mut cnf = app_status.config().lock().unwrap();
204             cnf.set_session_id(&s);
205             //cnf.save().map_err(|e| format!("FailSession:{e}"))?;
206             Ok(Some(s))
207         }
208         Ok(None) => Ok(None),
209         Err(e) => Err(format!("FailSession:{e}")),
210     }
211 }
212
213 /// 現在のセッションを取得する。
214 fn get_curr_session(app_status: &AppStatus) -> Option<Uuid> {
215     let conf = app_status.config().lock().unwrap();
216     conf.get_session_id()
217 }

```

1.3 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  }
```


1.4 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::BaseDirs;
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 CONF_DIR_NAME: &str = "neko_todo";
13  const DB_HOST: &str = "NEKO_DB_DB_HOST";
14  const DB_USER: &str = "NEKO_DB_DB_USER";
15  const DB_PASS: &str = "NEKO_DB_DB_PASS";
16  const SESSION: &str = "NEKO_DB_SESSION_ID";
17
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  }
26
27  impl NekoTodoConfig {
28      pub fn new() -> dotenvy::Result<Self> {
29          let file = Self::get_config_file_path().map_err(dotenvy::Error::Io)?;
30          dotenvy::from_path(file)?;
31          let session_id = std::env::var(SESSION)
32              .ok()
33              .map(|s| Uuid::parse_str(&s).expect("環境ファイル異常:SESSION_ID 不正"));
34
35          Ok(Self {
36              db_host: std::env::var(DB_HOST).unwrap_or_default(),
37              db_user: std::env::var(DB_USER).unwrap_or_default(),
38              db_pass: std::env::var(DB_PASS).unwrap_or_default(),
39              session_id,
40              dirty: false,
41          })
42      }
43  }
```

```

44 pub fn get_db_host(&self) -> &str {
45     &self.db_host
46 }
47
48 pub fn get_db_user(&self) -> &str {
49     &self.db_user
50 }
51
52 pub fn get_db_pass(&self) -> &str {
53     &self.db_pass
54 }
55
56 pub fn get_session_id(&self) -> Option<Uuid> {
57     self.session_id
58 }
59
60 pub fn set_db_host(&mut self, val: &str) {
61     self.db_host = val.to_string();
62     self.dirty = true;
63 }
64
65 pub fn set_db_user(&mut self, val: &str) {
66     self.db_user = val.to_string();
67     self.dirty = true;
68 }
69
70 pub fn set_db_pass(&mut self, val: &str) {
71     self.db_pass = val.to_string();
72     self.dirty = true;
73 }
74
75 pub fn set_session_id(&mut self, uuid: &Uuid) {
76     self.session_id = Some(*uuid);
77     self.dirty = true;
78 }
79
80 pub fn save(&mut self) -> Result<()> {
81     if !self.dirty {
82         return Ok(());
83     }
84     let path = Self::get_config_file_path()?;
85     let file = OpenOptions::new().write(true).truncate(true).open(&path)?;
86     let mut buffer = BufWriter::new(file);
87     writeln!(buffer, "{}={}", DB_HOST, self.get_db_host())?;
88     writeln!(buffer, "{}={}", DB_USER, self.get_db_user())?;

```

```

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

```

```

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

```

```

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

```

1.5 アプリケーション設定情報の処理 setup.rs

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

```

```
89 }
90
91 #[derive(Error, Debug)]
92 pub enum SetupError {
93     #[error("設定ファイルへのアクセスに失敗")]
94     SetupFile(#[from] dotenvy::Error),
95     #[error("--setup 時には、server,user,pass の設定が必須です")]
96     Argument,
97     #[error("データベースへの接続に失敗")]
98     ConnectDatabase(#[from] TodoError),
99 }
```


1.6 todo モデル処理 todo.rs

```
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::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 => unimplemented!("[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     ) -> Result<Vec<ItemTodo>, TodoError> {
30         let ref_date = Local::now().date_naive();
31         self.database
32             .get_todo_item(sess, ref_date, only_imcomplete)
33             .await
34             .map_err(|e| match e {
35                 DbError::FailDbAccess(e) => TodoError::FailDbAccess(e),
36                 e => unimplemented!("[get_todo_list]get_todo_item[{e}]"),
37             })
38     }
39
40     /// 新規の todo を追加する
41     /// 引数 item の id, user_name, update_date, update_date は無視される。
42     pub async fn add_todo(&self, sess: Uuid, item: &ItemTodo) -> Result<(), TodoError> {
43         // ユーザー名を取得
```

```

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

```

```

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

```

```

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

```

```

179  #[derive(Error, Debug)]
180  pub enum TodoError {
181      #[error("データベース初期化失敗")]
182      DbInit(sqlx::Error),
183      #[error("すでに、このユーザー名は使用されています。")]
184      DuplicateUser(sqlx::Error),
185      #[error("ユーザーパスワードのハッシュに失敗。")]
186      HashUserPassword(#[from] bcrypt::BcryptError),
187      #[error("ユーザーが見つかりません。")]
188      NotFoundUser,
189      #[error("パスワードが違います。")]
190      WrongPassword,
191      #[error("セッションが見つかりません。")]
192      NotFoundSession,
193      #[error("指定 id の todo が見つかりません。")]
194      NotFoundTodo,
195      #[error("データベースアクセスに失敗")]
196      FailDbAccess(sqlx::Error),
197  }
198
199  #[cfg(test)]
200  mod test {
201      use super::*;
202      use sqlx::MySqlPool;
203
204      impl Todo {
205          fn test_new(pool: MySqlPool) -> Self {
206              Self {
207                  database: Database::new_test(pool),
208              }
209          }
210      }
211
212      #[sqlx::test]
213      async fn new_user_and_login(pool: MySqlPool) {
214          let todo = Todo::test_new(pool);
215          // ユーザー生成
216          let user_name = "testdayo";
217          let user_pass = "passnano";
218          todo.add_user(user_name, user_pass).await.unwrap();
219
220          // 正しいユーザーでログイン
221          let _sess = todo.login(user_name, user_pass).await.unwrap();
222
223          // 間違ったユーザー名でログイン

```

```

224 let res = todo.login("detarame", user_pass).await;
225 match res {
226     Ok(_) => assert!(false, "こんなユーザーいないのに、なんでログインできたの?"),
227     Err(TodoError::NotFoundUser) => {},
228     Err(e) => assert!(false, "おなしいエラーが帰ってきた。{e}"),
229 }
230
231 // 間違ったパスワードでログイン
232 let res = todo.login(user_name, "detarame").await;
233 match res {
234     Ok(_) => assert!(false, "間違ったパスワードでログインできちゃだめ"),
235     Err(TodoError::WrongPassword) => {},
236     Err(e) => assert!(false, "こんなえらーだめです。{e}"),
237 }
238 }
239
240 #[sqlx::test]
241 async fn is_valid_session_test(pool: MySqlPool) {
242     let todo = Todo::test_new(pool);
243
244     // テスト用ユーザーの生成及び、ログイン
245     let user_name = "testdayo";
246     let user_pass = "passwordnano";
247
248     todo.add_user(user_name, &user_pass).await.unwrap();
249     let sess = todo.login(user_name, user_pass).await.unwrap();
250
251     // 正しいセッションを検索する。
252     let new_sess = todo.is_valid_session(&sess).await.unwrap();
253     match new_sess {
254         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。"),
255         None => assert!(false, "正しいセッションが見つからなかった。"),
256     };
257
258     // 間違ったセッションを検索する。
259     let none_sess = todo.is_valid_session(&Uuid::now_v7()).await.unwrap();
260     match none_sess {
261         Some(_) => assert!(false, "こんなセッションがあるわけがない。"),
262         None => {},
263     }
264 }
265
266 #[sqlx::test]
267 async fn add_todo_test(pool: MySqlPool) {
268     use chrono::Days;

```

```

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

```

```

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

```



```

359         "三件目のデータは None に変換してくれてるはず。"
360     );
361 }
362
363 #[sqlx::test]
364 async fn change_done_test(pool: MySqlPool) {
365     let todo = Todo::test_new(pool);
366     let sess = login_for_test(&todo).await;
367     create_todo_for_test(&todo, sess).await;
368
369     let items = todo.get_todo_list(sess, true).await.unwrap();
370     let item = items
371         .iter()
372         .find(|&i| i.title.find("1 件目").is_some())
373         .expect("「1 件目」を含むアイテムは必ずあるはず");
374     assert!(!item.done, "まだ、未完了のはずです。");
375     let id = item.id;
376     todo.change_done(id, sess, true)
377         .await
378         .expect("状態更新に失敗。あってはならない。");
379     let items = todo.get_todo_list(sess, true).await.unwrap();
380     assert_eq!(
381         items.len(),
382         2,
383         "一件完了済みにしたので、このリストは 2 件しかない。"
384     );
385     let items = todo.get_todo_list(sess, false).await.unwrap();
386     assert_eq!(items.len(), 3, "完了済みを含むので、3 件になる。");
387     let item = items
388         .iter()
389         .find(|&i| i.id == id)
390         .expect("さっきあった id だから必ずある。");
391     assert_eq!(item.done, true, "さっき完了済みに変更した。");
392
393     let max_id = items.iter().max_by_key(|&x| x.id).unwrap().id;
394     let res = todo.change_done(max_id + 1, sess, false).await;
395     match res {
396         Ok(_) => assert!(false, "この id の todo があるはずがない。"),
397         Err(TodoError::NotFoundTodo) => {}
398         Err(e) => assert!(false, "このエラーもありえない。[{e}]"),
399     };
400
401     // 間違ったセッションのテスト
402     let res = todo.change_done(id, Uuid::now_v7(), true).await;
403     match res {

```

```

404     Ok(_) => assert!(false, "このセッションでは、更新を許してはいけない。"),
405     Err(TodoError::NotFoundTodo) => { /* 正常 */ }
406     Err(e) => assert!(false, "このエラーもおかしい。[{e}]"),
407 }
408 }
409
410 async fn login_for_test(todo: &Todo) -> Uuid {
411     let user_name = "testdayo";
412     let user_pass = "passrordnona";
413     todo.add_user(user_name, user_pass).await.unwrap();
414     todo.login(user_name, user_pass).await.unwrap()
415 }
416
417 async fn create_todo_for_test(todo: &Todo, sess: Uuid) {
418     use chrono::Days;
419     let items = [
420         ItemTodo {
421             id: 100,
422             user_name: "kore_naihazu".to_string(),
423             title: "テストアイテム 1 件目".to_string(),
424             work: Some("これは、中身を入れる.".to_string()),
425             update_date: None,
426             start_date: Some(Local::now().date_naive() - Days::new(1)),
427             end_date: Some(Local::now().date_naive() + Days::new(5)),
428             done: false,
429         },
430         ItemTodo {
431             id: 100,
432             user_name: "kore_naihazu".to_string(),
433             title: "テストアイテム 2 件目 (work=null)".to_string(),
434             work: Some("").to_string(),
435             update_date: None,
436             start_date: Some(Local::now().date_naive() - Days::new(1)),
437             end_date: Some(Local::now().date_naive() + Days::new(5)),
438             done: false,
439         },
440         ItemTodo {
441             id: 100,
442             user_name: "kore_naihazu".to_string(),
443             title: "テストアイテム 3 件目 (work=space)".to_string(),
444             work: Some(" \t ".to_string()),
445             update_date: None,
446             start_date: Some(Local::now().date_naive() - Days::new(1)),
447             end_date: Some(Local::now().date_naive() + Days::new(5)),
448             done: false,

```

```
449         },
450     ];
451     for item in items {
452         todo.add_todo(sess, &item).await.unwrap();
453     }
454 }
455 }
```

1.7 データベースアクセス database.rs

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

```

44         .end_date
45         .unwrap_or(NaiveDate::from_ymd_opt(9999, 12, 31).unwrap());
46     query(sql)
47         .bind(&item.user_name)
48         .bind(&item.title)
49         .bind(&item.work)
50         .bind(start_date)
51         .bind(end_date)
52         .execute(&self.pool)
53         .await
54         .map_err(DbError::FailDbAccess)?;
55     Ok(())
56 }
57
58 /// Todo の一覧を取得する。
59 /// 基準日 (ref_date) 以降のアイテムを選別する。
60 /// セッション ID を必要とする。
61 /// 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。
62 pub async fn get_todo_item(
63     &self,
64     sess: Uuid,
65     ref_date: NaiveDate,
66     only_incomplete: bool,
67 ) -> Result<Vec<ItemTodo>, DbError> {
68     let sql1 = r#"
69         select t.id, t.user_name, title, work, update_date, start_date, end_date, done
70         from todo t join sessions s on s.user_name = t.user_name
71         where s.id=? and t.start_date <= ?
72         "#;
73     let sql2 = " and done = false";
74     let sql = if only_incomplete {
75         format!("{}", sql1, sql2)
76     } else {
77         format!("{}", sql1)
78     };
79     let items = query_as::<_, ItemTodo>(&sql)
80         .bind(sess.to_string())
81         .bind(ref_date)
82         .fetch_all(&self.pool)
83         .await
84         .map_err(DbError::FailDbAccess)?;
85
86     Ok(items)
87 }
88

```

```

89  /// 指定 id の Todo 項目を取得する。
90  /// 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。
91  pub async fn get_todo_item_with_id(&self, id: u32, sess: Uuid) -> Result<ItemTodo, DbError> {
92      let sql = r#"
93          select t.id, t.user_name, t.title, t.work, t.update_date, t.start_date, t.end_date,
94             ↪ t.done
95          from todo t join sessions s on s.user_name = t.user_name
96          where s.id=? and t.id=?
97          "#;
98      query_as:::<_, ItemTodo>(sql)
99          .bind(sess.to_string())
100          .bind(id)
101          .fetch_one(&self.pool)
102          .await
103          .map_err(|e| match e {
104              sqlx::Error::RowNotFound => DbError::NotFoundTodo,
105              e => DbError::FailDbAccess(e),
106          })
107  }
108
109  /// Todo の完了状態を更新する。
110  pub async fn change_done(&self, id: u32, done: bool) -> Result<(), DbError> {
111      let sql = "update todo set done = ? where id = ?";
112      let res = query(sql)
113          .bind(done)
114          .bind(id)
115          .execute(&self.pool)
116          .await
117          .map_err(DbError::FailDbAccess)?;
118      if res.rows_affected() > 0 {
119          Ok(())
120      } else {
121          Err(DbError::NotFoundTodo)
122      }
123  }
124
125  /// ユーザーの追加
126  pub async fn add_user(&self, name: &str, pass: &str) -> Result<(), DbError> {
127      let sql = "insert into users(name, password) values (?, ?)";
128      query(sql)
129          .bind(name)
130          .bind(pass)
131          .execute(&self.pool)
132          .await
133          .map_err(|e| match e {

```

```

133         sqlx::Error::Database(ref db_err) => {
134             if db_err.kind() == sqlx::error::ErrorKind::UniqueViolation {
135                 DbError::DuplicateUserName(e)
136             } else {
137                 DbError::FailDbAccess(e)
138             }
139         }
140         _ => DbError::FailDbAccess(e),
141     })?;
142     Ok(())
143 }
144
145 /// ユーザー名をキーとして、ユーザー情報を取得
146 pub async fn get_user(&self, name: &str) -> Result<User, DbError> {
147     let sql = "select name, password from users where name = ?;";
148     query_as(sql)
149         .bind(name)
150         .fetch_one(&self.pool)
151         .await
152         .map_err(|e| match e {
153             sqlx::Error::RowNotFound => DbError::NotFoundUser,
154             e => DbError::FailDbAccess(e),
155         })
156 }
157
158 /// セッション ID をキーにしてユーザー情報を取得
159 pub async fn get_user_from_sess(&self, sess: Uuid) -> Result<User, DbError> {
160     let sql = r#"
161         select u.name, u.password
162         from users u join sessions s on u.name=s.user_name
163         where s.id = ?;
164     "#;
165
166     query_as(sql)
167         .bind(sess.to_string())
168         .fetch_one(&self.pool)
169         .await
170         .map_err(|e| match e {
171             sqlx::Error::RowNotFound => DbError::NotFoundSession,
172             e => DbError::FailDbAccess(e),
173         })
174 }
175
176 /// セッション情報を新規作成する。
177 /// 生成した uuid を返す。

```

```

178 pub async fn make_new_session(&self, user_name: &str) -> Result<Uuid, DbError> {
179     let sql = "insert into sessions(id, user_name) values (?,?);";
180     // キー情報の作成
181     let id = Uuid::now_v7();
182
183     query(sql)
184         .bind(id.to_string())
185         .bind(user_name)
186         .execute(&self.pool)
187         .await
188         .map_err(|err| match err {
189             sqlx::Error::Database(ref e) => {
190                 if e.is_foreign_key_violation() {
191                     // 外部キーエラー。存在しないユーザーを指定した。
192                     return DbError::NotFoundUser;
193                 }
194                 DbError::FailDbAccess(err)
195             }
196             _ => DbError::FailDbAccess(err),
197         })?;
198
199     Ok(id)
200 }
201
202 /// 指定されたセッションを新規セッションに更新する。
203 /// 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。
204 pub async fn update_session(&self, id: &uuid::Uuid) -> Result<Uuid, DbError> {
205     let mut tr = self.pool.begin().await.map_err(DbError::FailDbAccess)?;
206     // 期限切れのセッション削除
207     let sql_old_del = "delete from sessions where expired < now();";
208     query(sql_old_del)
209         .execute(&mut *tr)
210         .await
211         .map_err(DbError::FailDbAccess)?;
212
213     // ユーザー ID の特定
214     let sql_query_user = "select user_name from sessions where id=?";
215     let user: String = query(sql_query_user)
216         .bind(id.to_string())
217         .fetch_one(&mut *tr)
218         .await
219         .map_err(|e| match e {
220             sqlx::Error::RowNotFound => DbError::NotFoundSession,
221             e => DbError::FailDbAccess(e),
222         })?

```



```

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

```

```

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

```

```

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

```

```

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

```

```

403         work: Some("中身入り".to_string()),
404         update_date: None,
405         start_date: Some(Local::now().date_naive()),
406         end_date: Some(Local::now().date_naive() + Days::new(3)),
407         done: true,
408     };
409     db.add_todo_item(&item).await.unwrap();
410
411     println!("テストデータを読み出す。一件しかないはず");
412     let last_day = Local::now().date_naive() + Days::new(1);
413     let res = db.get_todo_item(sess, last_day, true).await.unwrap();
414     assert_eq!(res.len(), 1, "あれ?一件のはずだよ");
415     item.id = res[0].id;
416     item.update_date = Some(Local::now().date_naive());
417     item.done = false;
418 }
419
420 /// todo の書き込みと読み出し。
421 /// work が未入力の場合。
422 #[sqlx::test]
423 async fn test_add_todo_without_work(pool: MySqlPool) {
424     let db = Database::new_test(pool);
425     let sess = login_for_test(&db).await;
426     let name = db.get_user_from_sess(sess).await.unwrap().name;
427
428     println!("テストデータをインサート");
429     let mut item = ItemTodo {
430         id: 0,
431         user_name: name.to_string(),
432         title: "インサートできるかな?".to_string(),
433         work: None,
434         update_date: None,
435         start_date: Some(Local::now().date_naive()),
436         end_date: Some(Local::now().date_naive() + Days::new(3)),
437         done: true,
438     };
439     db.add_todo_item(&item).await.unwrap();
440
441     println!("テストデータを読み出す。一件しかないはず");
442     let last_day = Local::now().date_naive() + Days::new(1);
443     let res = db.get_todo_item(sess, last_day, true).await.unwrap();
444     assert_eq!(res.len(), 1, "あれ?一件のはずだよ");
445     item.id = res[0].id;
446     item.update_date = Some(Local::now().date_naive());
447     item.done = false;

```

```

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

```

```

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

```

```

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

```



```

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

```

```

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

```

2 フロントエンド React 関係

2.1 index.html

```
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
12 export const routes = createBrowserRouter(
13   createRoutesFromElements(
14     <>
15       <Route element={ <BasePage/> }>
16         <Route path="/" element={ <Init/> }/>
17         <Route path="/login" element={ <Login/> }/>
18         <Route path="/regist_user" element={ <RegistUser/> }/>
19         <Route path="/todo" element={ <TodoList/> }/>
20         <Route path="/addtodo" element={ <AddTodo/> }/>
21       </Route>
22     </>
23   ));
24
25 function App() {
26   return (
27     <RouterProvider router={routes}/>
28   );
29 }
30 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 Register.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 Register() {
10     const { register, handleSubmit, formState: {errors} } = useForm();
11     const [ sendMessage, setSendMessage ] = useState('');
12     const navi = useNavigate();
13     const onSubmit = async (data) => {
14         try {
15             setSendMessage(' 送信中です。 ');
16             await invoke('regist_user', { name: data.name, password: data.pass });
17             navi('/login');
18         } catch (e) {
19             setSendMessage(' エラーが発生しました。{' + e + '} ');
20             console.log(e);
21         }
22     };
23
24     return (
25         <>
26         <h1> 新規ユーザー登録 </h1>
27         <p> すべての欄を入力してください。 </p>
28         <VStack as="form" onSubmit={handleSubmit(onSubmit)}>
29             <FormControl
30                 isValid={!!errors.name}
31                 label="ユーザー名"
32                 errorMessage={errors?.name?.message}
33             >
34                 <Input {...register("name", {required: " 入力必須です。 "})}/>
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                 isInvalid={!!errors.name}
35                 label="ユーザー名"
36                 errorMessage={errors?.name?.message}
37             >
38                 <Input {...register("name", {required: "入力必須です。"})}/>
39             </FormControl>
40             <FormControl
41                 isInvalid={!!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 { useNavigate } from "react-router-dom";
2  import { useQuery } from "@tanstack/react-query";
3  import { Grid, GridItem, HStack, IconButton } from "@yamada-ui/react";
4  import { invoke } from "@tauri-apps/api/core";
5  import { AiOutlineFileAdd } from "react-icons/ai";
6  import "./App.css";
7  import TodoItem from "./todoitem";
8
9  const get_todo_list = async () => invoke('get_todo_list') ;
10
11 function TodoList() {
12
13     const { data: todos, isLoading: isTodoListLoading , isError, error} = useQuery({
14         queryKey: ['todo_list'],
15         queryFn: get_todo_list,
16     });
17
18     const navi = useNavigate();
19     const handleAddTodo = () => navi('/addtodo');
20
21     if (isTodoListLoading) {
22         return ( <p> loading... </p> );
23     }
24
25     if (isError) {
26         return ( <p> エラーだよ。{error}</p> );
27     }
28
29     console.log(todos);
30     return (
31         <>
32             <HStack>
33                 <IconButton icon={<AiOutlineFileAdd/>} onClick={handleAddTodo}/>
34             </HStack>
35
36             <h1>現在の予定</h1>
37             <Grid templateColumns="repeat(4, 1fr)" gap="md">
38                 {todos?.map( todo_item => {
39                     return (
40                         <GridItem key={todo_item.id} w="full" rounded="md" bg="primary">
41                             <TodoItem item={todo_item}/>
42                         </GridItem>
43                     )
44                 })}
```

```
44         })
45         </Grid>
46     </>
47     );
48 }
49
50
51 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 {useMutation, useQueryClient} from "@tanstack/react-query";
3 import {invoke} from "@tauri-apps/api/core";
4 import { SimpleGrid, GridItem, IconButton, Text } from "@yamada-ui/react";
5 import { BsWrenchAdjustable } from "react-icons/bs";
6 import { BsAlarm } from "react-icons/bs";
7 import { BsEmojiGrin } from "react-icons/bs";
8
9 export default function TodoItem({item}) {
10     const queryClient = useQueryClient();
11     const {mutate} = useMutation({
12         mutationFn: () => {
13             return invoke("update_done", {id: item.id, done: !item.done});
14         },
15         onSuccess: () => {
16             queryClient.invalidateQueries({ queryKey: ["todo_list"]});
17         }
18     });
19
20     const onClick = () => {
21         console.log(item.id + " : " + item.title);
22         mutate();
23     }
24
25     // 日付の表示内容生成
26     let end_date = new Date(item.end_date);
27     if (item.end_date === "9999-12-31") {
28         end_date = null;
29     }
30     const start_date = new Date(item.start_date);
31     const update_date = new Date(item.update_date);
32
33     // アイコンボタンのアイコン選択
34     let done_icon;
35     if (item.done) {
36         done_icon = <BsEmojiGrin/>;
37     } else if (!!end_date && getDate(new Date(), end_date)) {
38         done_icon = <BsAlarm/>;
39     } else {
40         done_icon = <BsWrenchAdjustable/>
41     }
42
43     return (
```

```

44     <>
45         <SimpleGrid w="full" columns={{base: 2, md: 1}} gap="md">
46             <GridItem>
47                 <IconButton size="xs" icon={done_icon} onClick={onClick}/>
48             </GridItem>
49
50             <GridItem>
51                 <Text fontSize="xs" align="right">
52                     {update_date?.toLocaleDateString()}
53                 </Text>
54             </GridItem>
55         </SimpleGrid>
56         <Text align="center" fontSize="lg" as="b">
57             {item.title}
58         </Text>
59         <Text fontSize="sm">
60             {item.work}
61         </Text>
62         <Text fontSize="sm">
63             {start_date?.toLocaleDateString()} ~ {end_date?.toLocaleDateString()}
64         </Text>
65     </>
66 );
67 }
68
69 function geDate(val1, val2) {
70     const year1 = val1.getFullYear();
71     const month1 = val1.getMonth();
72     const day1 = val1.getDate();
73     const year2 = val2.getFullYear();
74     const month2 = val2.getMonth();
75     const day2 = val2.getDate();
76
77     if (year1 === year2) {
78         if (month1 === month2) {
79             return day1 >= day2;
80         } else {
81             return month1 > month2;
82         }
83     } else {
84         return year1 > year2;
85     }
86 }
87

```

2.11 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;
```

2.12 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('/todo'),
24         onError: (error) => setErrorMessage(error),
25     });
26
27     return (
28         <>
29             <FormProvider {...form}>
30                 <VStack as="form" onSubmit={handleSubmit((data)=>mutate(data))>
31                     <FormControl
32                         invalid={!errors.title}
33                         label="タイトル"
34                         errorMessage={errors?.title?.message}
35                     >
36                         <Input placeholder="やること"
37                             {...register("title", {required:"入力必須です。"})}/>
38                     </FormControl>
39                     <FormControl label="詳細">
40                         <Textarea {...register("work")} />
41                     </FormControl>
42                     <InputDate name="start" label="開始"/>
43                     <InputDate name="end" label="終了"/>
```

```

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

```

2.13 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     } 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
```