

neko_todo ソースリスト

美都

2025 年 1 月 28 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	9
1.4	コンフィグ設定処理 config.rs	10
1.5	アプリケーション設定情報の処理 setup.rs	15
1.6	todo モデル処理 todo.rs	18
1.7	データベースアクセス database.rs	30
2	フロントエンド React 関係	47
2.1	index.html	47
2.2	メイン CSS ファイル	48
2.3	main.jsx	51
2.4	アプリケーションメイン App.jsx	52
2.5	全体のベースページ BasePage.jsx	53
2.6	アプリケーションの初期化 Init.jsx	54
2.7	ユーザー登録画面 RegistUser.jsx	55
2.8	ログイン画面 Login.jsx	57
2.9	todo リストの表示 TodoList.jsx	59
2.10	todo アイテム表示 todoitem.jsx	61
2.11	todo アイテムの追加 AddTodo.jsx	64
2.12	todo アイテムの編集 EditTodo.jsx	65
2.13	todo アイテム内容の入力フォーム InputTodo.jsx	67
2.14	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	69
3	データベース構成	71
3.1	テーブル生成スクリプト create_table.sql	71

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 uuid::Uuid;
15
16 #[tauri::command]
17 fn greet(name: &str) -> String {
18     format!("Hello, {}! You've been greeted from Rust!", name)
19 }
20
21 #[cfg_attr(mobile, tauri::mobile_entry_point)]
22 pub fn run() {
23     let app_status = match setup() {
24         Ok(s) => s,
25         Err(e) => {
26             error!("{}", e);
27             std::process::exit(1)
28         }
29     };
30
31     let app = tauri::Builder::default()
32         .plugin(tauri_plugin_shell::init())
33         .manage(app_status)
34         .invoke_handler(tauri::generate_handler![
35             greet,
36             get_todo_list,
37             get_todo_with_id,
38             regist_user,
39             login,
40             is_valid_session,
41             add_todo,
42             update_done,
43             edit_todo,
```

```

44     })
45     .build(tauri::generate_context!())
46     .expect("error thile build tauri application");
47 app.run(|app, event| {
48     if let tauri::RunEvent::Exit = event {
49         eprintln!("終了処理開始");
50         let state = app.state::<AppState>();
51         state.config().lock().unwrap().save().unwrap();
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     let sess = match get_curr_session(&app_status) {
60         Some(u) => u,
61         None => return Err("NotLogin".to_string()),
62     };
63
64     app_status
65         .todo()
66         .get_todo_list(sess, true)
67         .await
68         .map_err(Into::into)
69 }
70
71 /// todo アイテムを取得する
72 #[tauri::command]
73 async fn get_todo_with_id(app_status: State<'_, AppState>, id: u32) -> Result<ItemTodo, String> {
74     let Some(sess) = get_curr_session(&app_status) else {
75         return Err("NotLogin".to_string());
76     };
77
78     app_status
79         .todo()
80         .get_todo_with_id(id, sess)
81         .await
82         .map_err(Into::into)
83 }
84
85 /// todo を追加する。
86 #[tauri::command]
87 async fn add_todo(app_status: State<'_, AppState>, item: FormTodo) -> Result<(), String> {
88     let sess = match get_cur_session_with_update(&app_status).await {

```

```

89         Ok(Some(u)) => u,
90         Ok(None) => return Err("NotLogin".to_string()),
91         Err(e) => return Err(e),
92     };
93
94     eprintln!("input = {:?}", &item);
95     app_status
96         .todo()
97         .add_todo(sess, &item.into())
98         .await
99         .map_err(Into::into)
100 }
101
102 /// todo の完了状態を変更する。
103 #[tauri::command]
104 async fn update_done(app_status: State<'_, AppStatus>, id: u32, done: bool) -> Result<(), String> {
105     let sess = match get_cur_session_with_update(&app_status).await {
106         Ok(Some(s)) => s,
107         Ok(None) => return Err("NotLogin".to_string()),
108         Err(e) => return Err(e),
109     };
110     app_status
111         .todo()
112         .change_done(id, sess, done)
113         .await
114         .map_err(Into::into)
115 }
116
117 /// todo の編集を行う。
118 #[tauri::command]
119 async fn edit_todo(
120     app_status: State<'_, AppStatus>,
121     id: u32,
122     item: FormTodo,
123 ) -> Result<(), String> {
124     let sess = match get_cur_session_with_update(&app_status).await {
125         Ok(Some(u)) => u,
126         Ok(None) => return Err("NotLogin".to_string()),
127         Err(e) => return Err(e),
128     };
129
130     eprintln!("input => id: {}, item: {:?}", id, &item);
131     let mut item: ItemTodo = item.into();
132     item.id = id;
133     app_status

```

```

134         .todo()
135         .edit_todo(&item, sess)
136         .await
137         .map_err(Into::into)
138     }
139
140     /// ユーザー登録
141     #[tauri::command]
142     async fn regist_user(
143         app_status: State<'_, AppStatus>,
144         name: String,
145         password: String,
146     ) -> Result<(), String> {
147         app_status
148             .todo()
149             .add_user(&name, &password)
150             .await
151             .map_err(Into::into)
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.todo().login(&name, &password).await?;
162
163         let mut cnf = app_status.config().lock().unwrap();
164         cnf.set_session_id(&session);
165         //cnf.save().map_err(|e| format!("OtherError:{}", e))?;
166         Ok(session.to_string())
167     }
168
169     /// 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)
170     #[command]
171     async fn is_valid_session(app_status: State<'_, AppStatus>) -> Result<bool, String> {
172         let sess = get_cur_session_with_update(&app_status)
173             .await
174             .map(|i| i.is_some());
175
176         match sess {
177             Ok(sess) => info!("セッション確認 ({}), if sess { "有効" } else { "無効" } ),
178             Err(ref e) => info!("セッション確認エラー ({}), e),
179         }
180     }

```

```

179     sess
180 }
181
182 /// 現在、有効なセッションを返す。
183 /// 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。
184 async fn get_cur_session_with_update(app_status: &AppStatus) -> Result<Option<Uuid>, String> {
185     let cur_session = get_curr_session(app_status);
186     let Some(cur_session) = cur_session else {
187         return Ok(None);
188     };
189
190     match app_status.todo().is_valid_session(&cur_session).await {
191         Ok(Some(s)) => {
192             // 更新されたセッションを再登録
193             let mut cnf = app_status.config().lock().unwrap();
194             cnf.set_session_id(&s);
195             //cnf.save().map_err(|e| format!("FailSession:{e}"))?;
196             Ok(Some(s))
197         }
198         Ok(None) => Ok(None),
199         Err(e) => Err(format!("FailSession:{e}")),
200     }
201 }
202
203 /// 現在のセッションを取得する。
204 fn get_curr_session(app_status: &AppStatus) -> Option<Uuid> {
205     let conf = app_status.config().lock().unwrap();
206     conf.get_session_id()
207 }
208
209 /// Todo 項目追加画面データ取得用
210 #[derive(Deserialize, Debug, Clone)]
211 struct FormTodo {
212     title: String,
213     work: Option<String>,
214     start: Option<String>,
215     end: Option<String>,
216 }
217
218 impl From<FormTodo> for ItemTodo {
219     fn from(val: FormTodo) -> Self {
220         let start = val.start.map(|d| d.replace("/", "-").parse().unwrap());
221         let end = val.end.map(|d| d.replace("/", "-").parse().unwrap());
222         ItemTodo {
223             id: 0,

```

```
224         user_name: "".to_string(),
225         title: val.title,
226         work: val.work,
227         update_date: None,
228         start_date: start,
229         end_date: end,
230         done: false,
231     }
232 }
233 }
```


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 => 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     ) -> 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 => unreachable!("[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 => unreachable!("[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 => unreachable!("[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 => unreachable!("[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 => unreachable!("[change_done] change_done [{e}]"),
105             })
106     }
107
108     /// Todo の編集を行う。
109     pub async fn edit_todo(&self, item: &ItemTodo, sess: Uuid) -> Result<(), TodoError> {
110         let mut item = item.clone();
111         if let Some(ref s) = item.work {
112             if s.trim().is_empty() {
113                 item.work = None;
114             }
115         }
116         self.get_todo_with_id(item.id, sess).await?;
117         self.database.edit_todo(&item).await.map_err(|e| match e {
118             DbError::FailDbAccess(e) => {
119                 error!("[Todo::edit_todo] edit_todo: [{e}]");
120                 TodoError::FailDbAccess(e)
121             }
122             DbError::NotFoundTodo => TodoError::NotFoundTodo,
123             e => unreachable!("[edit_todo] edit_todo [{e}]"),
124         })
125     }
126
127     /// ユーザーの追加を行う。
128     pub async fn add_user(&self, name: &str, password: &str) -> Result<(), TodoError> {
129         let hashed_pass = hash(password, DEFAULT_COST)?;
130         if let Err(e) = self.database.add_user(name, &hashed_pass).await {
131             match e {
132                 DbError::DuplicateUserName(e) => return Err(TodoError::DuplicateUser(e)),
133                 DbError::FailDbAccess(e) => {

```

```

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

```

```

179         e => {
180             unreachable!("[Todo::is_valid_session]is_session_valid:[{e}]")
181         }
182     }?;
183     if is_valid {
184         match self.database.update_session(sess).await {
185             Ok(s) => Ok(Some(s)),
186             Err(DbError::NotFoundSession) => Ok(None),
187             Err(DbError::FailDbAccess(e)) => Err(TodoError::FailDbAccess(e)),
188             Err(e) => {
189                 unreachable!("[Todo::is_valid_session]update_session:[{e}]")
190             }
191         }
192     } else {
193         Ok(None)
194     }
195 }
196 }
197
198 #[derive(Error, Debug)]
199 pub enum TodoError {
200     #[error("FailInitDatabase")]
201     DbInit(sqlx::Error),
202     #[error("DuplicateUserName")]
203     DuplicateUser(sqlx::Error),
204     #[error("InvalidPassword:{0}")]
205     HashUserPassword(#[from] bcrypt::BcryptError),
206     #[error("NotFoundUser")]
207     NotFoundUser,
208     #[error("WrongPassword")]
209     WrongPassword,
210     #[error("NotFoundSession")]
211     NotFoundSession,
212     #[error("NotFoundTodo")]
213     NotFoundTodo,
214     #[error("DatabaseError:{0}")]
215     FailDbAccess(sqlx::Error),
216 }
217
218 impl From<TodoError> for String {
219     fn from(value: TodoError) -> Self {
220         value.to_string()
221     }
222 }
223

```

```

224  #[cfg(test)]
225  mod test {
226      use super::*;
227      use sqlx::MySqlPool;
228
229      impl Todo {
230          fn test_new(pool: MySqlPool) -> Self {
231              Self {
232                  database: Database::new_test(pool),
233              }
234          }
235      }
236
237      #[sqlx::test]
238      async fn new_user_and_login(pool: MySqlPool) {
239          let todo = Todo::test_new(pool);
240          // ユーザー生成
241          let user_name = "testdayo";
242          let user_pass = "passnano";
243          todo.add_user(user_name, user_pass).await.unwrap();
244
245          // 正しいユーザーでログイン
246          let _sess = todo.login(user_name, user_pass).await.unwrap();
247
248          // 間違ったユーザー名でログイン
249          let res = todo.login("detarame", user_pass).await;
250          match res {
251              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?"),
252              Err(TodoError::NotFoundUser) => {}
253              Err(e) => unreachable!("おなじなエラーが帰ってきた。{e}"),
254          }
255
256          // 間違ったパスワードでログイン
257          let res = todo.login(user_name, "detarame").await;
258          match res {
259              Ok(_) => unreachable!("間違ったパスワードでログインできちゃだめ"),
260              Err(TodoError::WrongPassword) => {}
261              Err(e) => unreachable!("こんなえらーだめです。{e}"),
262          }
263      }
264
265      #[sqlx::test]
266      async fn is_valid_session_test(pool: MySqlPool) {
267          let todo = Todo::test_new(pool);
268

```

```

269 // テスト用ユーザーの生成及び、ログイン
270 let user_name = "testdayo";
271 let user_pass = "passwordnano";
272
273 todo.add_user(user_name, user_pass).await.unwrap();
274 let sess = todo.login(user_name, user_pass).await.unwrap();
275
276 // 正しいセッションを検索する。
277 let new_sess = todo.is_valid_session(&sess).await.unwrap();
278 match new_sess {
279     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。"),
280     None => unreachable!("正しいセッションが見つからなかった。"),
281 };
282
283 // 間違ったセッションを検索する。
284 let none_sess = todo.is_valid_session(&Uuid::now_v7()).await.unwrap();
285 if none_sess.is_some() {
286     unreachable!("こんなセッションがあるわけがない。");
287 }
288 }
289
290 #[sqlx::test]
291 async fn add_todo_test(pool: MySqlPool) {
292     use chrono::Days;
293
294     let todo = Todo::test_new(pool);
295     let sess = login_for_test(&todo).await;
296
297     let item1 = ItemTodo {
298         id: 100,
299         user_name: "kore_naihazu".to_string(),
300         title: "テストアイテム 1 件目".to_string(),
301         work: Some("これは、中身を入れる.".to_string()),
302         update_date: None,
303         start_date: Some(Local::now().date_naive() - Days::new(1)),
304         end_date: Some(Local::now().date_naive() + Days::new(5)),
305         done: true,
306     };
307     let item2 = ItemTodo {
308         id: 100,
309         user_name: "kore_naihazu".to_string(),
310         title: "テストアイテム 2 件目 (work=null)".to_string(),
311         work: Some("").to_string(),
312         update_date: None,
313         start_date: Some(Local::now().date_naive() - Days::new(1)),

```



```

314         end_date: Some(Local::now().date_naive() + Days::new(5)),
315         done: true,
316     };
317     let item3 = ItemTodo {
318         id: 100,
319         user_name: "kore_naihazu".to_string(),
320         title: "テストアイテム 3 件目 (work=space)".to_string(),
321         work: Some("\t ".to_string()),
322         update_date: None,
323         start_date: Some(Local::now().date_naive() - Days::new(1)),
324         end_date: Some(Local::now().date_naive() + Days::new(5)),
325         done: true,
326     };
327     todo.add_todo(sess, &item1)
328         .await
329         .expect("1 件目の追加に失敗");
330     let res = todo
331         .get_todo_list(sess, true)
332         .await
333         .expect("1 件目の取得に失敗");
334     assert_eq!(res.len(), 1, "一件目が取得できなかった?");
335     assert_eq!(res[0].title, item1.title, "一件目の title が違う");
336     assert_eq!(res[0].work, item1.work, "一件目の work が違う");
337     assert_eq!(res[0].user_name, "testdayo", "一件目の user_name が違う");
338     assert_eq!(
339         res[0].update_date,
340         Some(Local::now().date_naive()),
341         "一件目の update_date が違う "
342     );
343     assert_eq!(res[0].start_date, item1.start_date, "一件目の開始日が違う");
344     assert_eq!(res[0].end_date, item1.end_date, "一件目の終了日が違う");
345     assert!(res[0].done, "一件目の完了マークが違う");
346
347     todo.add_todo(sess, &item2)
348         .await
349         .expect("二件目の追加に失敗");
350     let res = todo
351         .get_todo_list(sess, true)
352         .await
353         .expect("二件目の取得に失敗");
354     assert_eq!(res.len(), 2, "二件あるはずなんだけど");
355     assert!(
356         res.iter()
357             .find(|&x| match x.title.find("work=null") {
358                 Some(n) => n > 0,

```

```

359         None => false,
360     })
361     .expect("二件目に追加したデータがない")
362     .work
363     .is_none(),
364     "二件目の work は None のはず"
365 );
366 todo.add_todo(sess, &item3)
367     .await
368     .expect("三件目の追加に失敗");
369 let res = todo
370     .get_todo_list(sess, true)
371     .await
372     .expect("三件目の取得に失敗");
373 assert_eq!(res.len(), 3, "三件あるはずですよ。");
374 assert!(
375     res.iter()
376         .find(|&x| match x.title.find("work=space") {
377             Some(n) => n > 0,
378             None => false,
379         })
380         .expect("三件目のデータがないよ?")
381         .work
382         .is_none(),
383     "三件目のデータは None に変換してくれてるはず。"
384 );
385 }
386
387 #[sqlx::test]
388 async fn change_done_test(pool: MySqlPool) {
389     let todo = Todo::test_new(pool);
390     let sess = login_for_test(&todo).await;
391     create_todo_for_test(&todo, sess).await;
392
393     let items = todo.get_todo_list(sess, true).await.unwrap();
394     let item = items
395         .iter()
396         .find(|&i| i.title.contains("1 件目"))
397         .expect("「1 件目」を含むアイテムは必ずあるはず");
398     assert!(!item.done, "まだ、未完了のはずです。");
399     let id = item.id;
400     todo.change_done(id, sess, true)
401         .await
402         .expect("状態更新に失敗。あってはならない。");
403     let items = todo.get_todo_list(sess, true).await.unwrap();

```

```

404     assert_eq!(
405         items.len(),
406         2,
407         "一件完了済みにしたので、このリストは2件しかない。"
408     );
409     let items = todo.get_todo_list(sess, false).await.unwrap();
410     assert_eq!(items.len(), 3, "完了済みを含むので、3件になる。");
411     let item = items
412         .iter()
413         .find(|&i| i.id == id)
414         .expect("さっきあったidだから必ずある。");
415     assert!(item.done, "さっき完了済みに変更した。");
416
417     let max_id = items.iter().max_by_key(|&x| x.id).unwrap().id;
418     let res = todo.change_done(max_id + 1, sess, false).await;
419     match res {
420         Ok(_) => unreachable!("このidのtodoがあるはずがない。"),
421         Err(TodoError::NotFoundTodo) => {}
422         Err(e) => unreachable!("このエラーもありえない。[{e}]"),
423     };
424
425     // 間違ったセッションのテスト
426     let res = todo.change_done(id, Uuid::now_v7(), true).await;
427     match res {
428         Ok(_) => unreachable!("このセッションでは、更新を許してはいけない。"),
429         Err(TodoError::NotFoundTodo) => { /* 正常 */ }
430         Err(e) => unreachable!("このエラーもおかしい。[{e}]"),
431     }
432 }
433
434 #[sqlx::test]
435 async fn edit_todo_test(pool: MySqlPool) {
436     let todo = Todo::test_new(pool);
437     let sess = login_for_test(&todo).await;
438     create_todo_for_test(&todo, sess).await;
439
440     let items = todo.get_todo_list(sess, false).await.unwrap();
441     let mut item = items
442         .iter()
443         .find(|&i| i.title.contains("1 件目"))
444         .unwrap()
445         .clone();
446     item.title = "更新した一件目".to_string();
447     if let Err(e) = todo.edit_todo(&item, sess).await {
448         unreachable!("更新処理に失敗した。[{e}]");

```

```

449     }
450     let Some(item_new) = todo
451         .get_todo_list(sess, false)
452         .await
453         .unwrap()
454         .iter()
455         .find(|&i| i.title.contains("更新した一件目"))
456         .cloned()
457     else {
458         unreachable!("更新したレコードが見つからないよ?");
459     };
460     assert_eq!(item.id, item_new.id, "更新したレコードの id が化けてる");
461
462     // ニセセッションで試す
463     match todo.edit_todo(&item, Uuid::now_v7()).await {
464         Ok(_) => unreachable!("偽のセッションで更新成功してはならない。"),
465         Err(TodoError::NotFoundTodo) => { /* 正常 */ },
466         Err(e) => unreachable!("偽セッションのときのエラー:{e}"),
467     }
468 }
469
470 async fn login_for_test(todo: &Todo) -> Uuid {
471     let user_name = "testdayo";
472     let user_pass = "passrordnona";
473     todo.add_user(user_name, user_pass).await.unwrap();
474     todo.login(user_name, user_pass).await.unwrap()
475 }
476
477 async fn create_todo_for_test(todo: &Todo, sess: Uuid) {
478     use chrono::Days;
479     let items = [
480         ItemTodo {
481             id: 100,
482             user_name: "kore_naihazu".to_string(),
483             title: "テストアイテム 1 件目".to_string(),
484             work: Some("これは、中身を入れる.".to_string()),
485             update_date: None,
486             start_date: Some(Local::now().date_naive() - Days::new(1)),
487             end_date: Some(Local::now().date_naive() + Days::new(5)),
488             done: false,
489         },
490         ItemTodo {
491             id: 100,
492             user_name: "kore_naihazu".to_string(),
493             title: "テストアイテム 2 件目 (work=null)".to_string(),

```

```

494         work: Some("").to_string()),
495         update_date: None,
496         start_date: Some(Local::now().date_naive() - Days::new(1)),
497         end_date: Some(Local::now().date_naive() + Days::new(5)),
498         done: false,
499     },
500     ItemTodo {
501         id: 100,
502         user_name: "kore_naihazu".to_string(),
503         title: "テストアイテム 3 件目 (work=space)".to_string(),
504         work: Some("\t ".to_string()),
505         update_date: None,
506         start_date: Some(Local::now().date_naive() - Days::new(1)),
507         end_date: Some(Local::now().date_naive() + Days::new(5)),
508         done: false,
509     },
510 ];
511 for item in items {
512     todo.add_todo(sess, &item).await.unwrap();
513 }
514 }
515 }

```

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  /// Todo の項目編集
126  pub async fn edit_todo(&self, item: &ItemTodo) -> Result<(), DbError> {
127      let start_date = item.start_date.unwrap_or(Local::now().date_naive());
128      let end_date = item
129          .end_date
130          .unwrap_or(NaiveDate::from_ymd_opt(9999, 12, 31).unwrap());
131
132      let sql = r#"
133          update todo

```



```

133         set title=?, work=?, update_date=curdate(), start_date=?, end_date=?
134         where id=?;
135         "#;
136     let res = query(sql)
137         .bind(&item.title)
138         .bind(&item.work)
139         .bind(start_date)
140         .bind(end_date)
141         .bind(item.id)
142         .execute(&self.pool)
143         .await
144         .map_err(DbError::FailDbAccess)?;
145     if res.rows_affected() > 0 {
146         Ok(())
147     } else {
148         Err(DbError::NotFoundTodo)
149     }
150 }
151
152 /// ユーザーの追加
153 pub async fn add_user(&self, name: &str, pass: &str) -> Result<(), DbError> {
154     let sql = "insert into users(name, password) values (?, ?)";
155     query(sql)
156         .bind(name)
157         .bind(pass)
158         .execute(&self.pool)
159         .await
160         .map_err(|e| match e {
161             sqlx::Error::Database(ref db_err) => {
162                 if db_err.kind() == sqlx::error::ErrorKind::UniqueViolation {
163                     DbError::DuplicateUserName(e)
164                 } else {
165                     DbError::FailDbAccess(e)
166                 }
167             }
168             _ => DbError::FailDbAccess(e),
169         })?;
170     Ok(())
171 }
172
173 /// ユーザー名をキーとして、ユーザー情報を取得
174 pub async fn get_user(&self, name: &str) -> Result<User, DbError> {
175     let sql = "select name, password from users where name = ?";
176     query_as(sql)
177         .bind(name)

```

```

178         .fetch_one(&self.pool)
179     .await
180     .map_err(|e| match e {
181         sqlx::Error::RowNotFound => DbError::NotFoundUser,
182         e => DbError::FailDbAccess(e),
183     })
184 }
185
186 /// セッション ID をキーにしてユーザー情報を取得
187 pub async fn get_user_from_sess(&self, sess: Uuid) -> Result<User, DbError> {
188     let sql = r#"
189         select u.name, u.password
190         from users u join sessions s on u.name=s.user_name
191         where s.id = ?;
192     "#;
193
194     query_as(sql)
195         .bind(sess.to_string())
196         .fetch_one(&self.pool)
197         .await
198         .map_err(|e| match e {
199             sqlx::Error::RowNotFound => DbError::NotFoundSession,
200             e => DbError::FailDbAccess(e),
201         })
202 }
203
204 /// セッション情報を新規作成する。
205 /// 生成した uuid を返す。
206 pub async fn make_new_session(&self, user_name: &str) -> Result<Uuid, DbError> {
207     let sql = "insert into sessions(id, user_name) values (?,?);";
208     // キー情報の作成
209     let id = Uuid::now_v7();
210
211     query(sql)
212         .bind(id.to_string())
213         .bind(user_name)
214         .execute(&self.pool)
215         .await
216         .map_err(|err| match err {
217             sqlx::Error::Database(ref e) => {
218                 if e.is_foreign_key_violation() {
219                     // 外部キーエラー。存在しないユーザーを指定した。
220                     return DbError::NotFoundUser;
221                 }
222                 DbError::FailDbAccess(err)

```

```

223     }
224     _ => DbError::FailDbAccess(err),
225     })?;
226
227     Ok(id)
228 }
229
230 /// 指定されたセッションを新規セッションに更新する。
231 /// 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。
232 pub async fn update_session(&self, id: &uuid::Uuid) -> Result<Uuid, DbError> {
233     let mut tr = self.pool.begin().await.map_err(DbError::FailDbAccess)?;
234     // 期限切れのセッション削除
235     let sql_old_del = "delete from sessions where expired < now()";
236     query(sql_old_del)
237         .execute(&mut *tr)
238         .await
239         .map_err(DbError::FailDbAccess)?;
240
241     // ユーザー ID の特定
242     let sql_query_user = "select user_name from sessions where id=?";
243     let user: String = query(sql_query_user)
244         .bind(id.to_string())
245         .fetch_one(&mut *tr)
246         .await
247         .map_err(|e| match e {
248             sqlx::Error::RowNotFound => DbError::NotFoundSession,
249             e => DbError::FailDbAccess(e),
250         })?
251         .get("user_name");
252
253     // 旧セッションの削除
254     let sql_del_curr_sess = "delete from sessions where id = ?";
255     query(sql_del_curr_sess)
256         .bind(id.to_string())
257         .execute(&mut *tr)
258         .await
259         .map_err(DbError::FailDbAccess)?;
260
261     // 新セッションの生成
262     let sql_create_sess = "insert into sessions(id, user_name) values (?, ?)";
263     let id = Uuid::now_v7();
264     query(sql_create_sess)
265         .bind(id.to_string())
266         .bind(user)
267         .execute(&mut *tr)

```

```

268         .await
269         .map_err(DbError::FailDbAccess)?;
270
271     tr.commit().await.map_err(DbError::FailDbAccess)?;
272     Ok(id)
273 }
274
275 /// 指定されたセッション ID が有効であるか確認する。
276 /// データベースエラーが発生した場合は、Err(DbError::FailDbAccess) を返す。
277 pub async fn is_session_valid(&self, sess: &Uuid) -> Result<bool, DbError> {
278     // 期限切れのセッションを削除する。
279     let sql_old_del = "delete from sessions where expired < now();";
280     query(sql_old_del)
281         .execute(&self.pool)
282         .await
283         .map_err(DbError::FailDbAccess)?;
284     // 指定セッション ID の有無を確認する。
285     let sql_find_sess = "select count(*) as cnt from sessions where id = ?;";
286     let sess_cnt: i64 = query(sql_find_sess)
287         .bind(sess.to_string())
288         .fetch_one(&self.pool)
289         .await
290         .map_err(DbError::FailDbAccess)?
291         .get("cnt");
292     if sess_cnt == 1 {
293         Ok(true)
294     } else {
295         Ok(false)
296     }
297 }
298 }
299
300 #[derive(FromRow, Debug, PartialEq)]
301 pub struct User {
302     pub name: String,
303     pub password: String,
304 }
305
306 #[derive(FromRow, Serialize, Deserialize, Debug, PartialEq, Clone)]
307 pub struct ItemTodo {
308     pub id: u32,
309     pub user_name: String,
310     pub title: String,
311     pub work: Option<String>,
312     pub update_date: Option<NaiveDate>,

```

```

313     pub start_date: Option<NaiveDate>,
314     pub end_date: Option<NaiveDate>,
315     pub done: bool,
316 }
317
318 #[derive(Error, Debug)]
319 pub enum DbError {
320     #[error("データベースへの接続に失敗。")]
321     FailConnect(sqlx::Error),
322     #[error("データベース操作失敗 (一般)")]
323     FailDbAccess(sqlx::Error),
324     #[error("User 挿入失敗 (name 重複)")]
325     DuplicateUserName(sqlx::Error),
326     #[error("ユーザーが見つかりません。")]
327     NotFoundUser,
328     #[error("指定されたセッション idが見つかりません。")]
329     NotFoundSession,
330     #[error("指定された id の todoが見つかりません。")]
331     NotFoundTodo,
332 }
333
334 #[cfg(test)]
335 mod test {
336     use chrono::Days;
337
338     use super::*;
339
340     /// テスト用の Database 生成。テスト用 Pool をインジェクション
341     impl Database {
342         pub(crate) fn new_test(pool: MySqlPool) -> Self {
343             Self { pool }
344         }
345     }
346
347     /// ユーザー生成のテスト
348     #[sqlx::test]
349     async fn test_add_user_and_get_user(pool: MySqlPool) {
350         let db = Database::new_test(pool);
351         db.add_user("hyara", "password").await.unwrap();
352         let user = db.get_user("hyara").await.unwrap();
353         assert_eq!(user.name, "hyara");
354         assert_eq!(user.password, "password");
355         let error_user = db.get_user("naiyo").await;
356         match error_user {
357             Ok(_) => unreachable!("結果が帰ってくるはずがない。"),

```

```

358     Err(DbError::NotFoundUser) => { /* 正常 */ }
359     Err(e) => unreachable!("このエラーはおかしい。{e}"),
360 }
361 }
362
363 /// セッション生成関係の一連のテスト。
364 #[sqlx::test]
365 async fn test_make_new_session(pool: MySqlPool) {
366     println!("まずはテスト用のユーザーの生成");
367     let db = Database::new_test(pool);
368     let user_name = "nekodayo";
369     let password = "password";
370     db.add_user(user_name, password).await.unwrap();
371
372     println!("次に、普通にセッションを作ってみる。");
373     let sess1 = db.make_new_session(user_name).await.unwrap();
374     println!("セッション生成成功 id=[{}]", sess1);
375
376     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。");
377     let sess2 = db.make_new_session("detarame").await;
378     match sess2 {
379         Ok(_) => unreachable!("このユーザーは存在しなかったはず。"),
380         Err(DbError::NotFoundUser) => { /* 正常 */ }
381         Err(e) => unreachable!("このエラーもおかしい。[{}]", e),
382     }
383
384     println!("普通に、セッションを更新してみる。");
385     let sess3 = db.update_session(&sess1).await.unwrap();
386     assert_ne!(sess1, sess3);
387
388     println!("ないはずのセッションを更新しようとしてみる。");
389     let sess4 = Uuid::now_v7();
390     let sess5 = db.update_session(&sess4).await;
391     match sess5 {
392         Ok(_) => unreachable!("このセッションはないはずなのに。"),
393         Err(DbError::NotFoundSession) => { /* 正常 */ }
394         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}"),
395     }
396 }
397
398 /// セッションが有効かどうかを確認するテスト
399 #[sqlx::test]
400 async fn test_is_session_valid(pool: MySqlPool) {
401     let db = Database::new_test(pool);
402

```

```

403     println!("テスト用ユーザーの作成");
404     let name = "nekodayo";
405     let pass = "nekodamon";
406     db.add_user(name, pass).await.unwrap();
407
408     println!("新規セッションを生成する。");
409     let sess = db.make_new_session(name).await.unwrap();
410     println!("生成したセッション ID は、[{}] です。", &sess);
411
412     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。");
413     assert!(db.is_session_valid(&sess).await.unwrap());
414
415     println!("偽セッション ID をいれて、問い合わせしてみる。");
416     assert!(db.is_session_valid(&Uuid::now_v7()).await.unwrap());
417 }
418
419 /// todo の書き込みと、単純な読み出しのテスト
420 #[sqlx::test]
421 async fn test_add_todo(pool: MySqlPool) {
422     let db = Database::new_test(pool);
423     let sess = login_for_test(&db).await;
424     let name = db.get_user_from_sess(sess).await.unwrap().name;
425
426     println!("テストデータをインサート");
427     let mut item = ItemTodo {
428         id: 0,
429         user_name: name.to_string(),
430         title: "インサートできるかな?".to_string(),
431         work: Some("中身入り".to_string()),
432         update_date: None,
433         start_date: Some(Local::now().date_naive()),
434         end_date: Some(Local::now().date_naive() + Days::new(3)),
435         done: true,
436     };
437     db.add_todo_item(&item).await.unwrap();
438
439     println!("テストデータを読み出す。一件しかないはず");
440     let last_day = Local::now().date_naive() + Days::new(1);
441     let res = db.get_todo_item(sess, last_day, true).await.unwrap();
442     assert_eq!(res.len(), 1, "あれ?一件のはずだよ");
443     item.id = res[0].id;
444     item.update_date = Some(Local::now().date_naive());
445     item.done = false;
446 }
447

```

```

448 /// todo の書き込みと読み出し。
449 /// work が未入力の場合。
450 #[sqlx::test]
451 async fn test_add_todo_without_work(pool: MySQLPool) {
452     let db = Database::new_test(pool);
453     let sess = login_for_test(&db).await;
454     let name = db.get_user_from_sess(sess).await.unwrap().name;
455
456     println!("テストデータをインサート");
457     let mut item = ItemTodo {
458         id: 0,
459         user_name: name.to_string(),
460         title: "インサートできるかな?".to_string(),
461         work: None,
462         update_date: None,
463         start_date: Some(Local::now().date_naive()),
464         end_date: Some(Local::now().date_naive() + Days::new(3)),
465         done: true,
466     };
467     db.add_todo_item(&item).await.unwrap();
468
469     println!("テストデータを読み出す。一件しかないはず");
470     let last_day = Local::now().date_naive() + Days::new(1);
471     let res = db.get_todo_item(sess, last_day, true).await.unwrap();
472     assert_eq!(res.len(), 1, "あれ?一件のはずだよ");
473     item.id = res[0].id;
474     item.update_date = Some(Local::now().date_naive());
475     item.done = false;
476 }
477
478 /// todo の書き込みと読み出し
479 /// done=true と false の挙動テスト
480 #[sqlx::test]
481 async fn test_get_todo_done_param(pool: MySQLPool) {
482     let db = Database::new_test(pool.clone());
483     let sess = login_for_test(&db).await;
484     let name = db.get_user_from_sess(sess).await.unwrap().name;
485
486     println!("テストデータをインサート");
487     let item = ItemTodo {
488         id: 0,
489         user_name: name.to_string(),
490         title: "インサートできるかな?".to_string(),
491         work: None,
492         update_date: None,

```



```

493         start_date: Some(Local::now().date_naive()),
494         end_date: Some(Local::now().date_naive() + Days::new(3)),
495         done: true,
496     };
497     db.add_todo_item(&item).await.unwrap();
498
499     println!("テストデータを読み出す。一件しかないはず");
500     let last_day = Local::now().date_naive() + Days::new(1);
501     let res = db.get_todo_item(sess, last_day, false).await.unwrap();
502     assert_eq!(res.len(), 1, "全部読み出しただけど一件あるはず。");
503     let res = db.get_todo_item(sess, last_day, true).await.unwrap();
504     assert_eq!(res.len(), 1, "未完了ただけど、一件あるはず。");
505
506     println!("今作った job を完了済みにする。");
507     let sql = "update todo set done=true where id=?";
508     query(sql).bind(res[0].id).execute(&pool).await.unwrap();
509     let res = db.get_todo_item(sess, last_day, false).await.unwrap();
510     assert_eq!(res.len(), 1, "全部読み出しただけど一件あるはず。");
511     let res = db.get_todo_item(sess, last_day, true).await.unwrap();
512     assert_eq!(res.len(), 0, "未完了ただけだから、なにもないはず。");
513 }
514
515 /// todo の書き込みと読み出し
516 /// 基準日の挙動テスト
517 #[sqlx::test]
518 async fn test_get_todo_ref_date(pool: MySqlPool) {
519     let db = Database::new_test(pool.clone());
520     let sess = login_for_test(&db).await;
521     let name = db.get_user_from_sess(sess).await.unwrap().name;
522
523     println!("テストデータをインサート");
524     let item = ItemTodo {
525         id: 0,
526         user_name: name.to_string(),
527         title: "インサートできるかな?".to_string(),
528         work: None,
529         update_date: None,
530         start_date: Some(Local::now().date_naive()),
531         end_date: Some(Local::now().date_naive() + Days::new(3)),
532         done: false,
533     };
534     db.add_todo_item(&item).await.unwrap();
535
536     let ref_date = Local::now().date_naive();
537     let res = db.get_todo_item(sess, ref_date, true).await.unwrap();

```

```

538     assert_eq!(res.len(), 1, "基準日と開始日が同じだからみつかる。");
539     let res = db
540         .get_todo_item(sess, ref_date + Days::new(1), true)
541         .await
542         .unwrap();
543     assert_eq!(res.len(), 1, "開始日の翌日が基準日だからみつかる。");
544     let res = db
545         .get_todo_item(sess, ref_date - Days::new(1), true)
546         .await
547         .unwrap();
548     assert_eq!(res.len(), 0, "基準日が開始日の前日だからみつからない。");
549     let res = db
550         .get_todo_item(sess, ref_date + Days::new(4), true)
551         .await
552         .unwrap();
553     assert_eq!(res.len(), 1, "基準日が期限を過ぎているけどみつかるの。");
554 }
555
556 #[sqlx::test]
557 async fn test_get_user_from_sess(pool: MySqlPool) {
558     let db = Database::new_test(pool.clone());
559
560     let sess = login_for_test(&db).await;
561     let name = db.get_user_from_sess(sess).await.unwrap().name;
562
563     let user = db.get_user_from_sess(sess).await.unwrap();
564     assert_eq!(user.name, name, "これはみつかるはず");
565     let dummy_sess = Uuid::now_v7();
566     let user = db.get_user_from_sess(dummy_sess).await;
567     match user {
568         Ok(_) => unreachable!("見つかるわけないでしょう。"),
569         Err(DbError::NotFoundSession) => { /* 正常 */ }
570         Err(e) => unreachable!("トラブルです。{e}"),
571     };
572 }
573
574 #[sqlx::test]
575 async fn test_change_done(pool: MySqlPool) {
576     let db = Database::new_test(pool);
577     let sess = login_for_test(&db).await;
578     let ref_date = Local::now().date_naive();
579     create_todo_for_test(&db, sess).await;
580
581     let items = db.get_todo_item(sess, ref_date, true).await.unwrap();
582     let item = items.iter().find(|&i| i.title.contains("二件目")).unwrap();

```

```

583 db.change_done(item.id, true).await.unwrap();
584
585 let items = db.get_todo_item(sess, ref_date, true).await.unwrap();
586 let item = items.iter().find(|&i| i.title.contains("二件目"));
587 assert!(item.is_none(), "状態を完了にしたので見つからないはず。");
588
589 let items = db.get_todo_item(sess, ref_date, false).await.unwrap();
590 let item = items.iter().find(|&i| i.title.contains("二件目"));
591 match item {
592     Some(i) => assert!(i.done, "完了済みになっているはずですね?"),
593     None => unreachable!("状態を変えたら、レコードなくなった???"),
594 }
595 assert_eq!(
596     items.len(),
597     3,
598     "全件見ているのでレコードは3件あるはずですが?"
599 );
600 }
601
602 #[sqlx::test]
603 async fn test_get_todo_with_id(pool: MySqlPool) {
604     let db = Database::new_test(pool);
605     let sess = login_for_test(&db).await;
606     create_todo_for_test(&db, sess).await;
607
608     let items = db
609         .get_todo_item(sess, Local::now().date_naive(), false)
610         .await
611         .unwrap();
612     let id = items
613         .iter()
614         .find(|&i| i.title.contains("一件目"))
615         .expect("これはあるはず")
616         .id;
617     let non_exist_id = items.iter().max_by_key(|&i| i.id).unwrap().id + 1;
618
619     // 正常な読み出し
620     let res = db
621         .get_todo_item_with_id(id, sess)
622         .await
623         .expect("これは正常に読み出せるはず。エラーはだめ");
624     res.work
625         .expect("このレコードは work を持つはずです。")
626         .find("働いています。")
627         .expect("work の内容がおかしい。");

```

```

628
629 // 間違った id
630 let res = db.get_todo_item_with_id(non_exist_id, sess).await;
631 match res {
632     Ok(_) => unreachable!("そんな ID は存在しなかったはずなのに。"),
633     Err(DbError::NotFoundTodo) => { /* 正常 */ }
634     Err(e) => unreachable!("データベースエラーだよ。({e})"),
635 }
636
637 // 間違ったセッション
638 let res = db.get_todo_item_with_id(id, Uuid::now_v7()).await;
639 match res {
640     Ok(_) => unreachable!("そんなセッションはないはず。"),
641     Err(DbError::NotFoundTodo) => { /* 正常 */ }
642     Err(e) => unreachable!("データベースエラー発生。({e})"),
643 }
644 }
645
646 #[sqlx::test]
647 async fn test_edit(pool: MySqlPool) {
648     let db = Database::new_test(pool);
649     let sess = login_for_test(&db).await;
650     create_todo_for_test(&db, sess).await;
651
652     // 書き込みテスト用レコードの取得
653     let today = Local::now().date_naive();
654     let items = db.get_todo_item(sess, today, false).await.unwrap();
655     let mut item = items
656         .iter()
657         .find(|&i| i.title.contains("一件目"))
658         .expect("ないはずがない。")
659         .clone();
660     item.title = "更新しました.".to_string();
661     item.work = Some("書き換え後".to_string());
662     item.start_date = Some(today - Days::new(5));
663     item.end_date = Some(today + Days::new(10));
664     db.edit_todo(&item).await.expect("更新がエラーを起こした。");
665     // 書き込み後の照合
666     let items_new = db.get_todo_item(sess, today, false).await.unwrap();
667     let item_new = items_new
668         .iter()
669         .find(|&i| i.title.contains("更新しました。"))
670         .expect("更新されたレコードが存在しない。");
671     assert_eq!(
672         item_new.work,

```

```

673         Some("書き換え後".to_string()),
674         "更新後の work がおかしい"
675     );
676     assert_eq!(
677         item_new.start_date,
678         Some(today - Days::new(5)),
679         "更新後の start_date がおかしい"
680     );
681     assert_eq!(
682         item_new.end_date,
683         Some(today + Days::new(10)),
684         "更新後の end_date がおかしい"
685     );
686
687     // 存在しないレコードの更新
688     let id_max_plus_one = items.iter().max_by_key(|&i| i.id).unwrap().id + 1;
689     item.id = id_max_plus_one;
690     let res = db.edit_todo(&item).await;
691     match res {
692         Ok(_) => unreachable!("更新できちゃだめっ"),
693         Err(DbError::NotFoundTodo) => {}
694         Err(e) => unreachable!("db_err: {e}"),
695     }
696 }
697
698 async fn login_for_test(db: &Database) -> Uuid {
699     println!("テスト用ユーザー及びセッションの生成");
700     let name = "test";
701     let pass = "test";
702     db.add_user(name, pass).await.unwrap();
703     db.make_new_session(name).await.unwrap()
704 }
705
706 async fn create_todo_for_test(db: &Database, sess: Uuid) {
707     let name = db.get_user_from_sess(sess).await.unwrap().name;
708
709     println!("テストデータをインサート");
710     let item = ItemTodo {
711         id: 0,
712         user_name: name.to_string(),
713         title: "一件目 (work 有り)".to_string(),
714         work: Some("働いています.".to_string()),
715         update_date: None,
716         start_date: Some(Local::now().date_naive()),
717         end_date: Some(Local::now().date_naive() + Days::new(3)),
718         done: false,

```

```

718     };
719     db.add_todo_item(&item).await.unwrap();
720
721     let item = ItemTodo {
722         id: 0,
723         user_name: name.to_string(),
724         title: "二件目 (work 無し)".to_string(),
725         work: None,
726         update_date: None,
727         start_date: Some(Local::now().date_naive()),
728         end_date: Some(Local::now().date_naive() + Days::new(3)),
729         done: false,
730     };
731     db.add_todo_item(&item).await.unwrap();
732
733     let item = ItemTodo {
734         id: 0,
735         user_name: name.to_string(),
736         title: "三件目 (work 無し)".to_string(),
737         work: None,
738         update_date: None,
739         start_date: Some(Local::now().date_naive()),
740         end_date: Some(Local::now().date_naive() + Days::new(3)),
741         done: false,
742     };
743     db.add_todo_item(&item).await.unwrap();
744 }
745 }

```

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
13 export const routes = createBrowserRouter(
14   createRoutesFromElements(
15     <>
16       <Route element={ <BasePage/> }>
17         <Route path="/" element={ <Init/> }/>
18         <Route path="/login" element={ <Login/> }/>
19         <Route path="/regist_user" element={ <RegistUser/> }/>
20         <Route path="/todo" element={ <TodoList/> }/>
21         <Route path="/addtodo" element={ <AddTodo/> }/>
22         <Route path="/edittodo/:id" element={ <EditTodo/> }/>
23       </Route>
24     </>
25   ));
26
27 function App() {
28   return (
29     <RouterProvider router={routes}/>
30   );
31 }
32 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 {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 } from "@yamada-ui/react";
6  import { BsWrenchAdjustable } from "react-icons/bs";
7  import { BsAlarm } from "react-icons/bs";
8  import { BsEmojiGrin } from "react-icons/bs";
9  import { BiPencil } from "react-icons/bi";
10
11 export default function TodoItem({item}) {
12     const navi = useNavigate();
13     const queryClient = useQueryClient();
14     const {mutate} = useMutation({
15         mutationFn: () => {
16             return invoke("update_done", {id: item.id, done: !item.done});
17         },
18         onSuccess: () => {
19             queryClient.invalidateQueries({ queryKey: ["todo_list"]});
20         }
21     });
22
23     const onEditClick = () => {
24         navi("/edittodo/"+item.id);
25     }
26
27     const onDoneClick = () => {
28         console.log(item.id + " : " + item.title);
29         mutate();
30     }
31
32     // 日付の表示内容生成
33     let end_date = new Date(item.end_date);
34     if (item.end_date === "9999-12-31") {
35         end_date = null;
36     }
37     const start_date = new Date(item.start_date);
38     const update_date = new Date(item.update_date);
39
40     // 完了ボタンのアイコン選択
41     let done_icon;
42     if (item.done) {
43         done_icon = <BsEmojiGrin/>;
```

```

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

```

```
89         return day1 >= day2;
90     } else {
91         return month1 > month2;
92     }
93 } else {
94     return year1 > year2;
95 }
96 }
97
```

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 }
```

2.13 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

```

2.14 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ー
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
```