ANEXA COD

/// <summary>

/// Callback for the click event on the "SignUp" button. Gets the input ///from the

/// textboxes and runs a series of tests before storing the employee in ///the database.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void buttonSignUp\_Click(object sender, EventArgs e)

{

string username = textBoxUsername.Text.Trim();

string password = textBoxPassword.Text.Trim();

string rePassword = textBoxRePassword.Text.Trim();

string firstName = textBoxFirstName.Text.Trim();

string lastName = textBoxLastName.Text.Trim();

string email = textBoxEmail.Text.Trim();

string phoneNr = textBoxPhoneNumber.Text.Trim();

try

{

// #0 check empty fields

if(username == "" || password == "" || rePassword == "" || firstName == "" || lastName == "" || email == "" || phoneNr == "")

{

throw new FieldsEmptyException("fields must not me empty");

}

// #1 check

// check f\_name, l\_name, phone\_nr to have correct format

bool isFNameValid = Regex.IsMatch(firstName, @"^[a-zA-Z]+$");

if (!isFNameValid)

{

throw new FirstNameException("first name not in a valid format");

}

bool isLNameValid = Regex.IsMatch(lastName, @"^[a-zA-Z]+$");

if (!isLNameValid)

{

throw new LastNameException("last name not in a valid format");

}

bool isPhoneNrValid = Regex.IsMatch(phoneNr, @"^[0-9]+$");

if (!isPhoneNrValid)

{

throw new PhoneNumberException("phone number not in a valid format");

}

// #2 check

// check email string with MailAddress class

// if it does not throw an error, we're good to go

var emailCheckAux = new MailAddress(email);

// #3 check

// check if password == re\_password

bool isPassValid = password == rePassword;

if(!isPassValid)

{

throw new PasswordMatchException("passwords don't match");

}

// #4 check

// check if username is unique in db

(bool isUsernameValid, Exception employeeValidEx) = ((LoginPage)this.TopLevelControl).EmployeeSRV.ValidateUsername(username);

if (employeeValidEx != null)

{

throw employeeValidEx;

}

if(!isUsernameValid) {

throw new UsernameTakenException("username already exists");

}

// if everything is right

// store user in db

// create a user model and save it to the form

(Employee employee, Exception registerEx) = ((LoginPage)this.TopLevelControl).EmployeeSRV.RegisterUser(username, password, firstName, lastName, email, phoneNr);

if(registerEx != null)

{

throw registerEx;

}

((LoginPage)this.TopLevelControl).Hide();

MainForm mainForm = new MainForm(employee);

mainForm.ShowDialog();

((LoginPage)this.TopLevelControl).Close();

} catch (Exception exception) when (

exception is FormatException ||

exception is ArgumentException ||

exception is ArgumentNullException)

{

MessageBox.Show("email not in the right format");

}

catch (Exception exception)

{

MessageBox.Show(exception.Message);

}

}

/// <summary>

/// Callback for the click event on the "LogIn" button. Checks the existence of the account based on username and password.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void buttonLogin\_Click(object sender, EventArgs e)

{

try

{

// get input from user

string username = textBoxUsername.Text;

string password = textBoxPassword.Text;

// check input textboxes

if (username == "" || password == "") { throw new FieldsEmptyException("please fill in the fields"); }

// check if username/password combo is good

(Employee employee, Exception ex) = ((LoginPage)this.TopLevelControl).EmployeeSRV.CheckEmployeeLogIn(username, password);

if(employee == null)

{

throw new Exception("user not found");

}

if(ex != null)

{

throw ex;

}

// login user if everything is right

((LoginPage)this.TopLevelControl).Hide();

MainForm mainForm = new MainForm(employee);

mainForm.ShowDialog();

((LoginPage)this.TopLevelControl).Close();

} catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Method to retrieve a subtask data based on its id.

/// </summary>

/// <param name="id">Subtask id.</param>

/// <returns>Returns the subtask data if it was found, otherwise null.

/// Also returns an exception in case an error happened while executing the statement.</returns>

public (Subtask, Exception) GetSubtaskById(int id)

{

string stmt = $"SELECT \* FROM subtasks WHERE subtaskid = '{id}'";

SQLiteCommand cmd = new SQLiteCommand(stmt, dbConnection);

using (cmd)

{

try

{

using (SQLiteDataReader dataReader = cmd.ExecuteReader())

{

Subtask subtask = null;

while (dataReader.Read())

{

int subtaskId = dataReader.GetInt32(0);

string subtaskTitle = dataReader.GetString(1);

string subtaskDescription = dataReader.GetString(2);

string subtaskStatus = dataReader.GetString(3);

int taskid = dataReader.GetInt32(4);

string employeeuuid = dataReader.GetString(5);

subtask = new Subtask(subtaskId, subtaskTitle, subtaskDescription, subtaskStatus, taskid, employeeuuid);

}

if (subtask == null) throw new Exception("subtask is null");

return (subtask, null);

}

}

catch (Exception ex)

{

return (null, new Exception(ex.Message));

}

}

}

/// <summary>

/// Method to retrieve all done tasks by an employee.

/// </summary>

/// <param name="empUUID"></param>

/// <returns></returns>

public (int, Exception) GetTasksDoneCount(string empUUID)

{

string stmt = $"SELECT COUNT(\*) FROM tasks WHERE employeeuuid = '{empUUID}' and taskstatus = 'done'";

using (SQLiteCommand command = new SQLiteCommand(stmt, dbConnection))

{

try

{

// Execute the query and get the count

int count = Convert.ToInt32(command.ExecuteScalar());

return (count, null);

} catch (Exception ex)

{

return (0, ex);

}

}

}

/// <summary>

/// Method to update status and progress of the task after a subtask has been modified.

/// </summary>

/// <param name="subtasks"></param>

/// <returns>Returns an exception if an error happened while executing the statement.</returns>

public Exception CheckSubtasksStatus(List<Subtask> subtasks)

{

if(subtasks == null)

{

return new NullSubtasksException("Subtask list is null");

}

if(subtasks.Count == 0)

{

return new SubtasksEmptyException("Subtask list is empty");

}

int taskId = subtasks[0].TaskId;

double doneSubtasks = 0;

double toDoSubtasks = 0;

foreach (Subtask subtask in subtasks)

{

if (subtask.Status == "toDo")

{

toDoSubtasks++;

}

if (subtask.Status == "inProgress")

{

Exception exc = UpdateTaskStatus(taskId, "inProgress");

if (exc != null)

{

return exc;

}

}

if (subtask.Status == "done")

{

doneSubtasks++;

}

}

if (toDoSubtasks == subtasks.Count)

{

Exception exc = UpdateTaskStatus(taskId, "toDo");

if (exc != null)

{

return exc;

}

}

if (doneSubtasks == subtasks.Count)

{

Exception exc = UpdateTaskStatus(taskId, "done");

if (exc != null)

{

return exc;

}

}

if (toDoSubtasks != subtasks.Count && doneSubtasks != subtasks.Count)

{

Exception exc = UpdateTaskStatus(taskId, "in-progress");

if (exc != null)

{

return exc;

}

}

int progress = (int)(doneSubtasks / subtasks.Count \* 100);

Exception ex = UpdateTaskProgress(taskId, progress);

if (ex != null)

{

return ex;

}

return null;

}

/// <summary>

/// Method to create an task row inside a data grid view.

/// </summary>

/// <param name="task"></param>

/// <returns>Returns the complete data grid view row with task data.</returns>

public DataGridViewRow makeTaskRow(Task task)

{

this.\_builder = new TaskBuilder();

TaskBuilder taskBuilder = (TaskBuilder)\_builder;

taskBuilder.Reset();

taskBuilder.SetID(task.ID.ToString());

taskBuilder.SetTitle(task.Name);

taskBuilder.SetDescription(task.Description);

taskBuilder.SetStatus(task.Status);

taskBuilder.SetProgress(task.Progress);

taskBuilder.SetDeadline(task.Deadline);

taskBuilder.SetGoToButton();

return taskBuilder.GetResult();

}

/// <summary>

/// Method to create a comment row inside a data grid view.

/// </summary>

/// <param name="comment"></param>

/// <returns>Returns the complete data grid view row with comment data.</returns>

public DataGridViewRow makeCommentRow(Comment comment)

{

this.\_builder = new CommentBuilder();

CommentBuilder commentBuilder = (CommentBuilder)\_builder;

commentBuilder.Reset();

commentBuilder.SetTitle(comment.Title);

commentBuilder.SetDescription(comment.Description);

commentBuilder.SetTimeReported(comment.TimeReported);

return commentBuilder.GetResult();

}