Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative
Istituto sistemi informativi e networking

SUPSI

Software Engineering and Development I

Semester Project

G.Corti, M.Besenzoni

Introduction

- **Objective**
  - **Create a software product!**

- Organization
  - The current laboratory teams will continue to work on this project
  - Mostly during the 4 hours laboratory

- Educational goals
  - Practice the topics learned in the course
  - Begin experimenting iterative/incremental work approaches in a team
    - Collaboration platform: `https://scm.ti-edu.ch/projects/labingsw0120xx20xxnn`
    - Git repository: `https://scm.ti-edu.ch/repogit/labingsw0120xx20xxnn`
    - `20xx20xx` is the academic year
    - `nn` is the group number

Evaluation

- As anticipated during class
  - This project will be evaluated as 30% of the final course evaluation

- We have access to tools to automatically monitor how you work in the team
  - Weekly reports will be published

- **All team members will receive the same final evaluation!**
  - It is up to the team members to equally split the workload and ensure all deadlines are timely met

Groups

Group01:
A.Taglialatela, R.Renna, G.Mendonça

Group02:
M.Nolli, G.Visconti, L.Di Bello

Group03:
I.Fontantini, D.Fusco, M.Berchtold

Group04:
B.Beffa, P.Catania, M.Toscanelli

Group05:
D.Ibrahim, S.Finiletti, M.Cadoni

Group06:
L.Babbucci, M.Dell'Oca, S.Giamboni

Group07:
A.Falce, A.Fetta, A.Sarak

Group08:
A.Di Nicola, D.Zappa, E.Manassero

Group09:
S.Schuemperli, A.Bracelli, M.Byketa

Group10:
L.Rausa, A.Garatti, M.Grgic Figueiredo

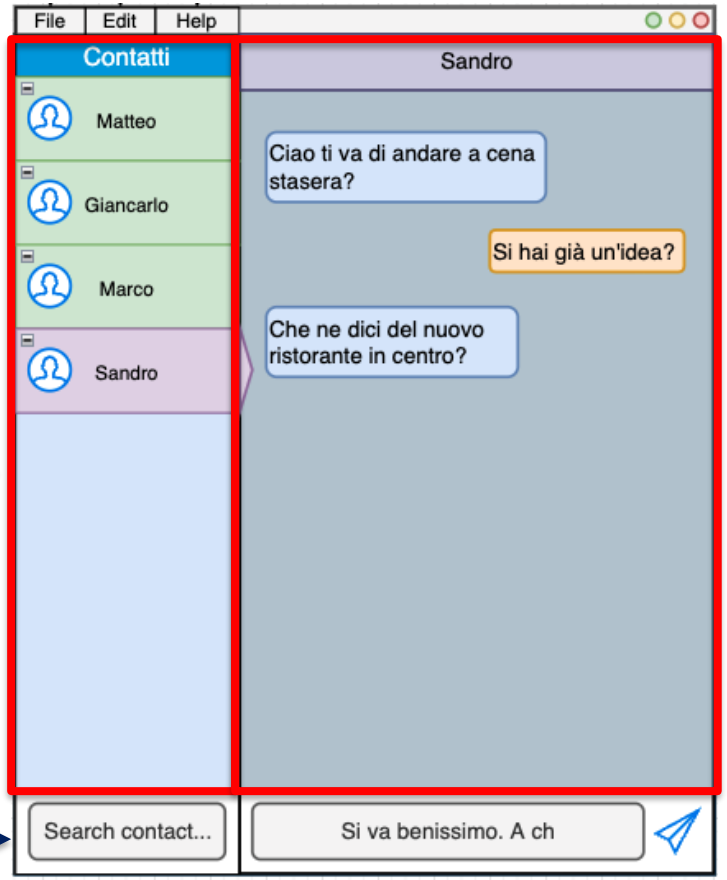
Group11:
M.Peluso, L.Di Folco

Product: “Local Chat” – Introduction

- In todays world everyone needs to quickly share information. Several messaging applications already exist but why not create our own?
- We want to create a flexible and simple tool that allows local users to send each other messages.
- The chats history and contacts must be maintained between sessions so that users can interrupt a conversation and continue it at a later time with any of the saved contacts.
- We want a client GUI stand-alone (in JavaFX) that follows the OS conventions (Linux, Window, MacOS)

Product: “Local Chat” – GUI

- The GUI will have two main stages displayed in the same frame: the **Login** page and the **Chat**
- The **Login** page allows to either chose an existing user or to create a new one. No password required and no duplicates.
- The **Chat** displays a contact list and on the side the chat history with the currently selected contact.
- New contacts can be added by searching the contact name in the search box

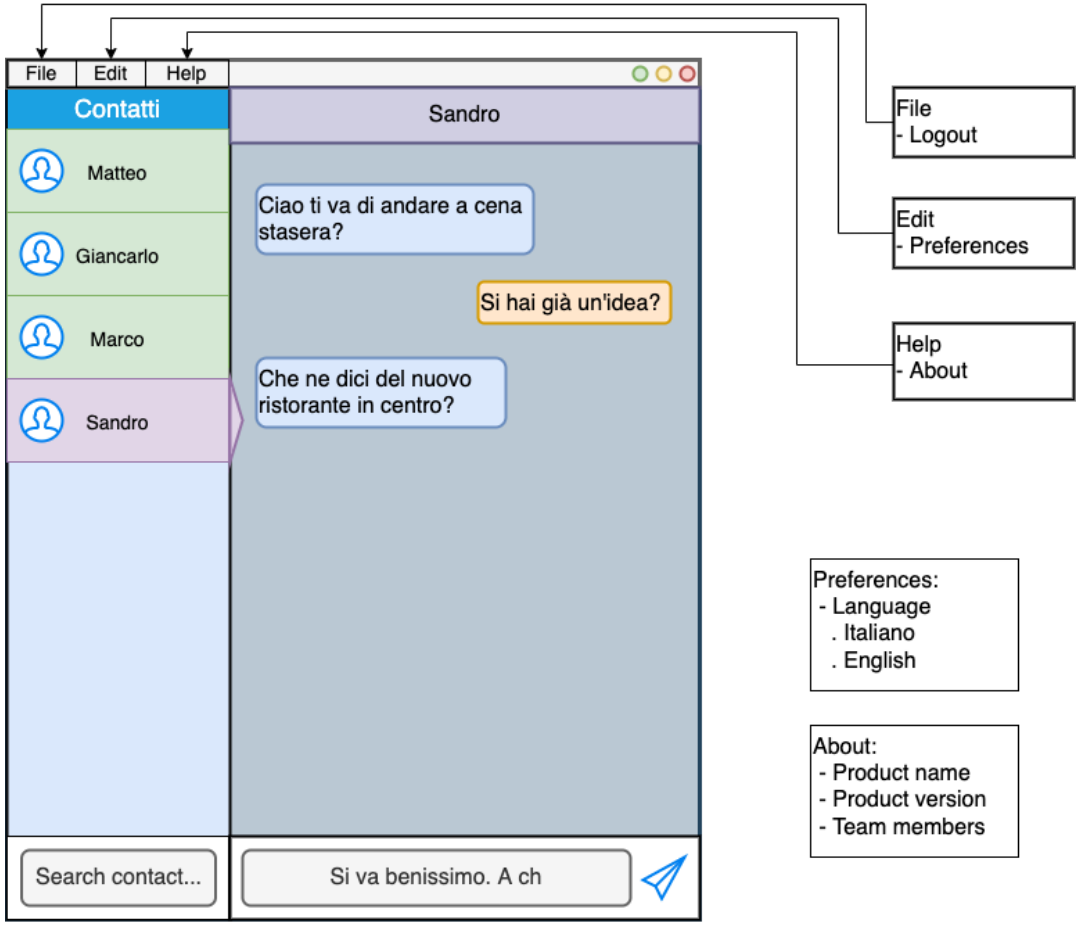


Product: “Local Chat” – Chat

- The frame must have a menu with the following items:
  - **File**
    - Logout
  - **Edit**
    - Preferences
  - **Help**
    - About

- **Preferences** will open a popup menu to change the application language (Italian – English)

- **About** will show information about the application (Product name, Product version, Team members)



Product: “Local Chat” – Entities (1)

- Users are defined by a unique name and a set of chats with other users
- Chats are the link between two users and represent the conversation between them
  - “sent” and “received” messages must be differentiated and displayed in the GUI as such depending on the current active user
- Each user has a set of contacts and chats. A user can be added to the contact list only if already created
- Messages do not allow formatting (bold, italic, …). Each message also saves the time it was sent formatted as hh:mm in the 24h format

Product: “Local Chat” – GUI Interactions

- Upon login, each user should be presented with the list of her/his saved contacts and the chats with the conversation history
- Messages can be sent by clicking the “send” button or by pressing the “enter” button
- New users can only be created in the **Login** page
- By clicking a contact in the contact list, the displayed chat will change and the contact should be highlighted in the list
- Messages bubbles should be displayed differently to reflect which are “sent” and which are “received”

Product: “Local Chat” – Persistence

- Chats history and users/contacts must be persisted
- This should be done in the file system in a CSV or JSON file (chose one)
- The name and location of the file must be defined as a variable in the user preferences read by the application at start-up
- The user preferences file is created automatically at start-up if it doesn’t exist (following the OS conventions)
  - GNU/Linux usually uses a hidden directory in the user home
  - An example could be: `~/.chat/user.prefs`
  - The default location of the persistence file is the same as the user preferences

Product: “Local Chat” – Interaction example

- Messages will not go through an “online” server, the project backend will simulate it. The application does not require a network connection and is a simplification of a normal messaging app.

- For example, a typical interaction can be tested with the following steps:

  1. Login as user **A**
  2. Add user **B** to the contacts list
  3. Send a message to user **B**
  4. Logout
  5. Login as user **B**
  6. Read the message received (and maybe reply)

- The above interaction is done on the same computer with the same application

Product: “Local Chat” – Notes

- **The main objective is to write good and clean code!**

- Although very appealing, do not waste hours of your time personalizing the GUI
  - Our suggestion is to first prepare a **reliable backend** and only later focus on making the GUI pretty

- The screenshots shown are guidelines to help you develop the interface, try to follow the simple examples and avoid creating different layouts

Engineering

- Remember to apply the Separation of Concerns (SoC) principle while designing the project structure and the various components

- More principles will be introduced in the upcoming weeks: information hiding, encapsulation, coupling, cohesion, … as well as SOLID principles, which you'll have to promptly introduce in this project.

- **Structure design (following SoC principles)**
  - Separate data management from their user/visualization (client) similarly to what has been done in the course/students laboratory exercise.
  - Data management should be split in at least two layers, data layer and service layer, so that the former can be replaced if and/or when necessary