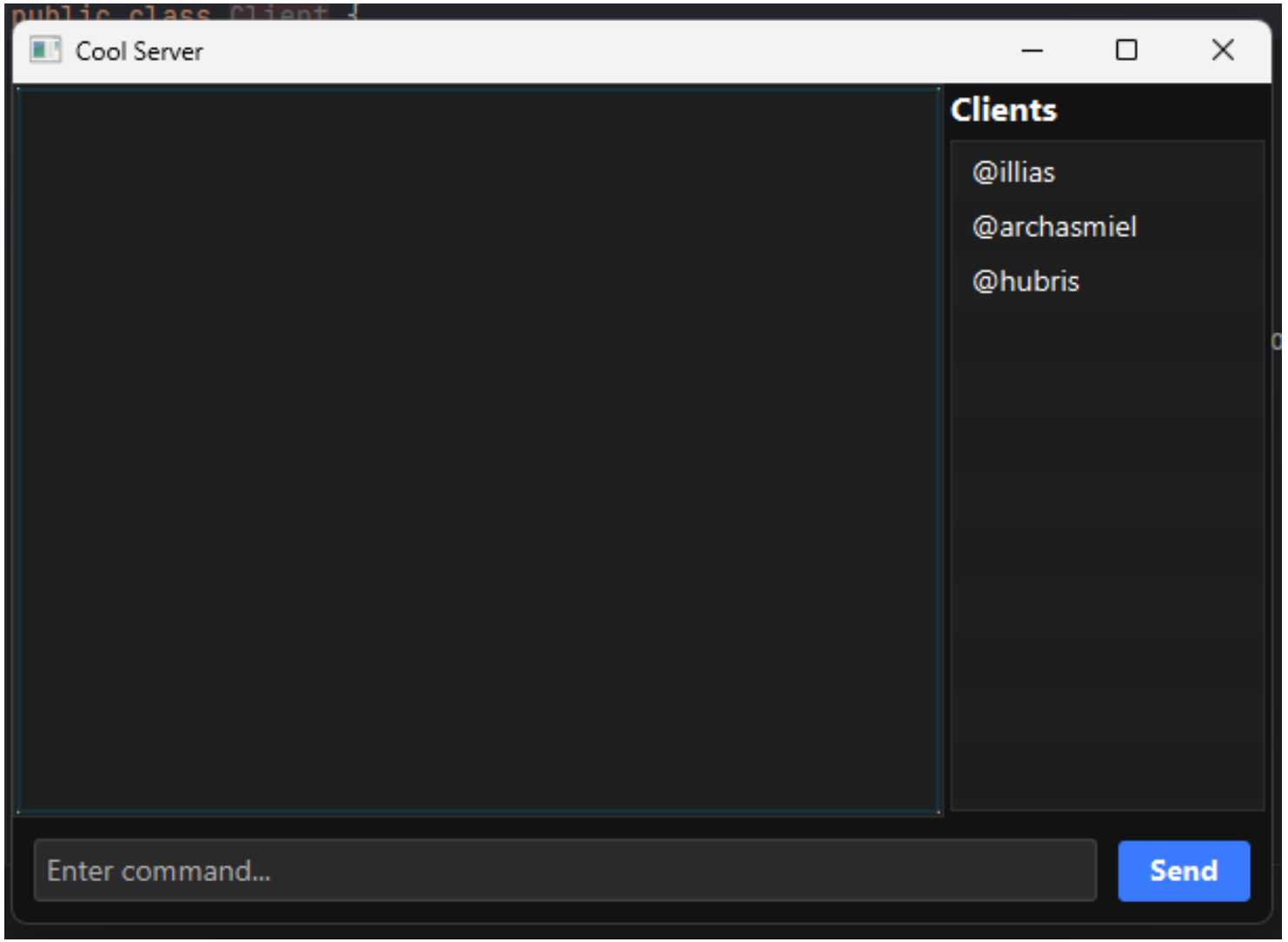
Мій Telegram

№4

Сервер та просте зберігання даних

План

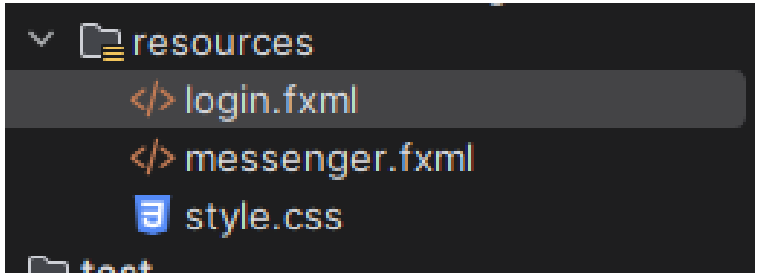
- Наш сервер зараз виглядає як пустий інтерфейс. Будемо додавати до нього наступне:
- можливість підключитися для клієнта
- показ на сервері всього що відбувається (підключення користувачів, додавання в друзі, виконання команд)
- зберігання даних (поки що Json)



Вхід

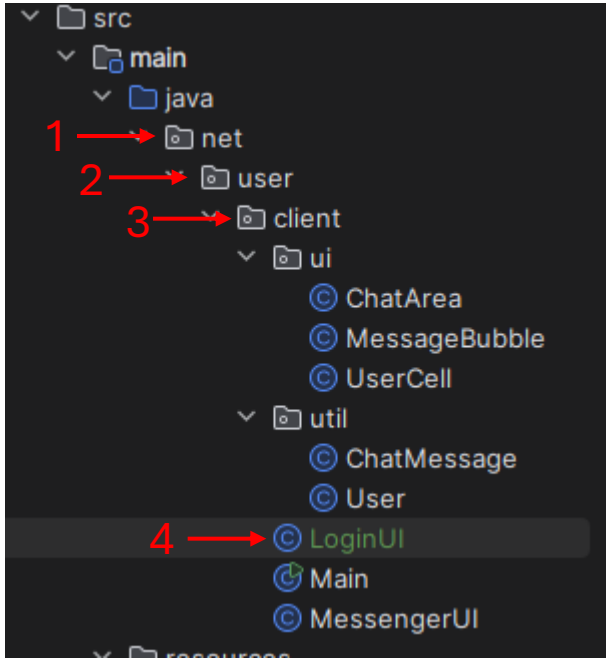
- Додамо можливість підключатися до сервера. Для цього зробимо додаткове просте меню, де буде:
- Адреса
- Порт
- Нікнейм
- Пароль

- **Створюємо у проекті client!**



Готовий файл до вікна в роздаточній папці.
Розмістіть його, далі права частина слайду

```
<AnchorPane xmlns="http://javafx.co
  xmlns:fx="http://javafx.com/fxml"
  fx:controller="LoginUI"
  prefHeight="400.0" prefWidth="600
```




```
<AnchorPane xmlns="http://javafx.com/javafx"
  xmlns:fx="http://javafx.com/fxml"
  fx:controller="net.user.client.LoginUI"
  prefHeight="400.0" prefWidth="600.0">

</AnchorPane>
```

Оцю частину треба написати обов’язково так, як іде шлях до класу

```java
public class LoginUI {

    public static final String MAIN_UI = "/login.fxml";

    @FXML private TextField ipAddressField;
    @FXML private TextField portField;
    @FXML private TextField nicknameField;
    @FXML private PasswordField passwordField;
    @FXML private Button connectButton;

    public void show(Stage stage) {
        FXMLLoader loader = new FXMLLoader(getClass().getResource(MAIN_UI));
        loader.setController(this);
        Parent root;
        try {
            root = loader.load();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        // Scene and stage setup
        Scene scene = new Scene(root, 600, 400);
        stage.setTitle("Login...");
        stage.setScene(scene);
        stage.setMinWidth(400);
        stage.setMinHeight(300);
        stage.show();
    }

    @FXML
    public void initialize() {
        connectButton.setDefaultButton(true);   // Зробити основною кнопку
    }
```

- Тепер до самого класу – розписуємо його. Дуже багато схожого з ти що було на MessengerUI/ServerUI
- Міняємо в Main завантаження інтерфейсу на Login, на нижній картинці

```java
public class Main extends Application {

    private static Stage PRIMARY_STAGE;

    @Override
    public void start(Stage primaryStage) {
        PRIMARY_STAGE = primaryStage;
        loadUI(stage -> new LoginUI().show(stage));
    }

    public static void loadUI(Consumer<Stage> stageLoader) {
        stageLoader.accept(PRIMARY_STAGE);
    }

    public static void main(String[] args) { launch(args); }
}
```

Тепер до змін у головному класі.

Зеленим – додали наш Stage як статичну змінну. Його можна уявити як головне вікно гри, а всі Scene, які створюємо у контролерах – це лише завантажена карта.

Тобто ми можемо **перезавантажувати** карти, а головне вікно лишиться завжди тим самим.

Звичайно, що розмір **Stage** будемо міняти в залежності від Scene (допустимо як в Сапері).

Статична змінна *не прив'язана до конкретного створеного об'єкту* а до всіх них і змінюється одночасно для всіх (простіше – закріплена до класу).

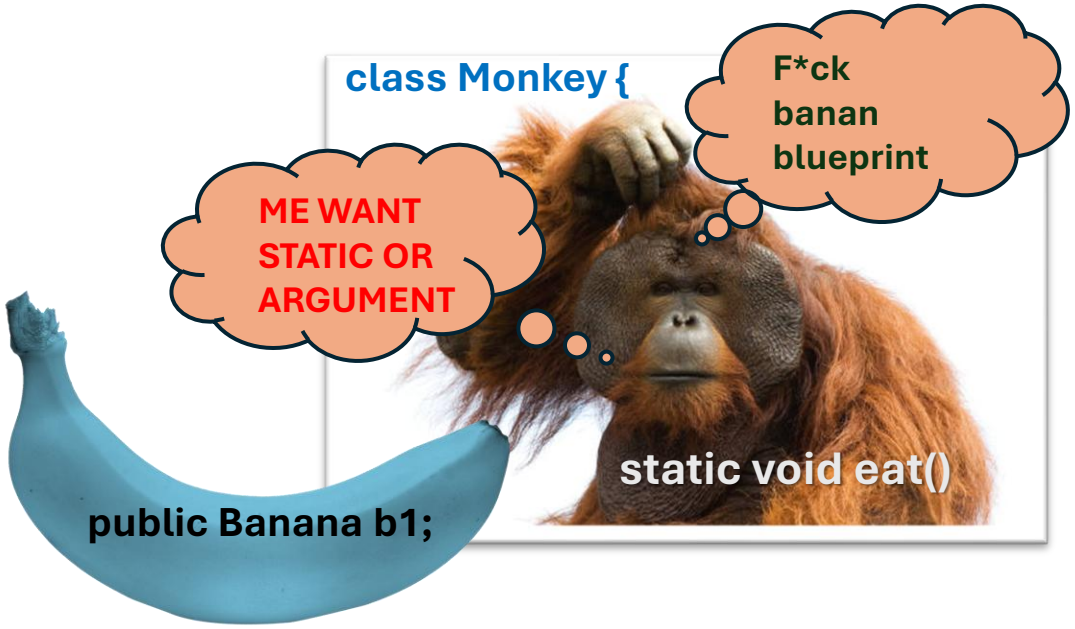
```
public class Main extends Application {

    private static Stage PRIMARY_STAGE;

    @Override
    public void start(Stage primaryStage) {
        PRIMARY_STAGE = primaryStage;
        loadUI(stage -> new LoginUI().show(stage));
    }

    public static void loadUI(Consumer<Stage> stageLoader) {
        stageLoader.accept(PRIMARY_STAGE);
    }

    public static void main(String[] args) { launch(args); }
}
```

Оранжевим – створили функцію для завантаження карти, зауважте що статичну (механіки схожі).

Статична функція МОЖЕ працювати з non-static (не-статичними) змінними, **якщо вони передані їй належним чином**. Всередині класу працювати з не-статичними *не зможе* (до якого конкретного об'єкту ви звертаєтеся, отримуючи non-static змінну, якщо static функція належить до всіх об'єктів?).

Нестатична функція МОЖЕ працювати зі всіма статичними змінними (вона належить до всіх об'єків і всюди однакова, тому логічно, що можна працювати з нею всюди).

Функції в Java бувають двох типів, крім статичних і нестатичних:
1) Звичайні, прив'язані до класу або об'єкту (static / non-static), їх вже давно бачили
2) Анонімні, які не належать ні до якого класу, АБО взяті з класу (так само і статичні і нестатичні). ***Продовження на наступному слайді.***




```java
public class Main extends Application {

    private static Stage PRIMARY_STAGE;

    @Override
    public void start(Stage primaryStage) {
        PRIMARY_STAGE = primaryStage;
        loadUI(stage -> new LoginUI().show(stage));
    }

    public static void loadUI(Consumer<Stage> stageLoader) {
        stageLoader.accept(PRIMARY_STAGE);
    }

    public static void main(String[] args) { launch(args); }
}
```

- ***Далі пропоную вам глянути ролик з поясненням лямбда виразів та інтерфейсів.***

- ***Для першого***
- ***https://www.youtube.com/watch?v=ToPZUJjAi6E***
- ***11 хвилин***

- ***Для другого https://www.youtube.com/watch?v=PnG5VtVhFT8***
- ***20 хвилин***

- ***Після цього скажіть, чи можемо ми використати різні функції show(Stage stage)?***
- ***Чи можемо створити Consumer<> без аргументів?***

```
public class Main extends Application {

    private static Stage PRIMARY_STAGE;

    @Override
    public void start(Stage primaryStage) {
        PRIMARY_STAGE = primaryStage;
        loadUI(stage -> new LoginUI().show(stage));
    }

    public static void loadUI(Consumer<Stage> stageLoader) {
        stageLoader.accept(PRIMARY_STAGE);
    }

    public static void main(String[] args) { launch(args); }
}
```

Стилізація

Спробуйте самі написати функціональний інтерфейс IntToString

додайте анотацію з відео

додайте єдину функцію
String apply(int number)

Додайте дві функції, які просто повертають ваше ім'я та вік

```
default String name() {
  // розписати
}

default int age() {
  // розписати
}
```

Далі саме цікаве, спробуйте створити змінну

```
IntToString fun1 = (num) -> Integer.toString(num);
IntToString fun2 = (num) -> Integer.toString(num+1);
IntToString fun3 = (num) -> Integer.toString(num*2);
```

Отримати результат

```
String res1 = fun1.apply(2);
String res2 = fun2.apply(2);
String res3 = fun3.apply(2);
```

Вивести в консоль

```
System.out.println(res1);……
```

Ми створили звичайний клас?
Чи можна міняти поведінку нашого інтерфейсу?
Як можна використати Supplier<> з відео?

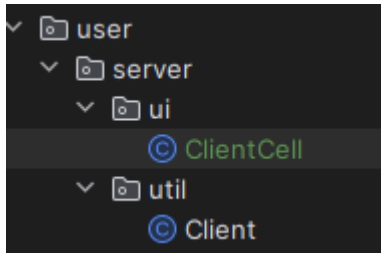
Підключення

Тепер переходимо до проекту server!

Далі будемо на сервері отримувати запит на приєднання і додавати до списку людей.

Зробимо список кращим – додамо айпі адреси, а під ними нікнейми.

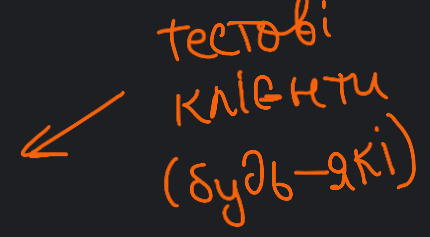
Всі зміни будуть командами – видалення, зміна паролю, очищення чатів користувачів.



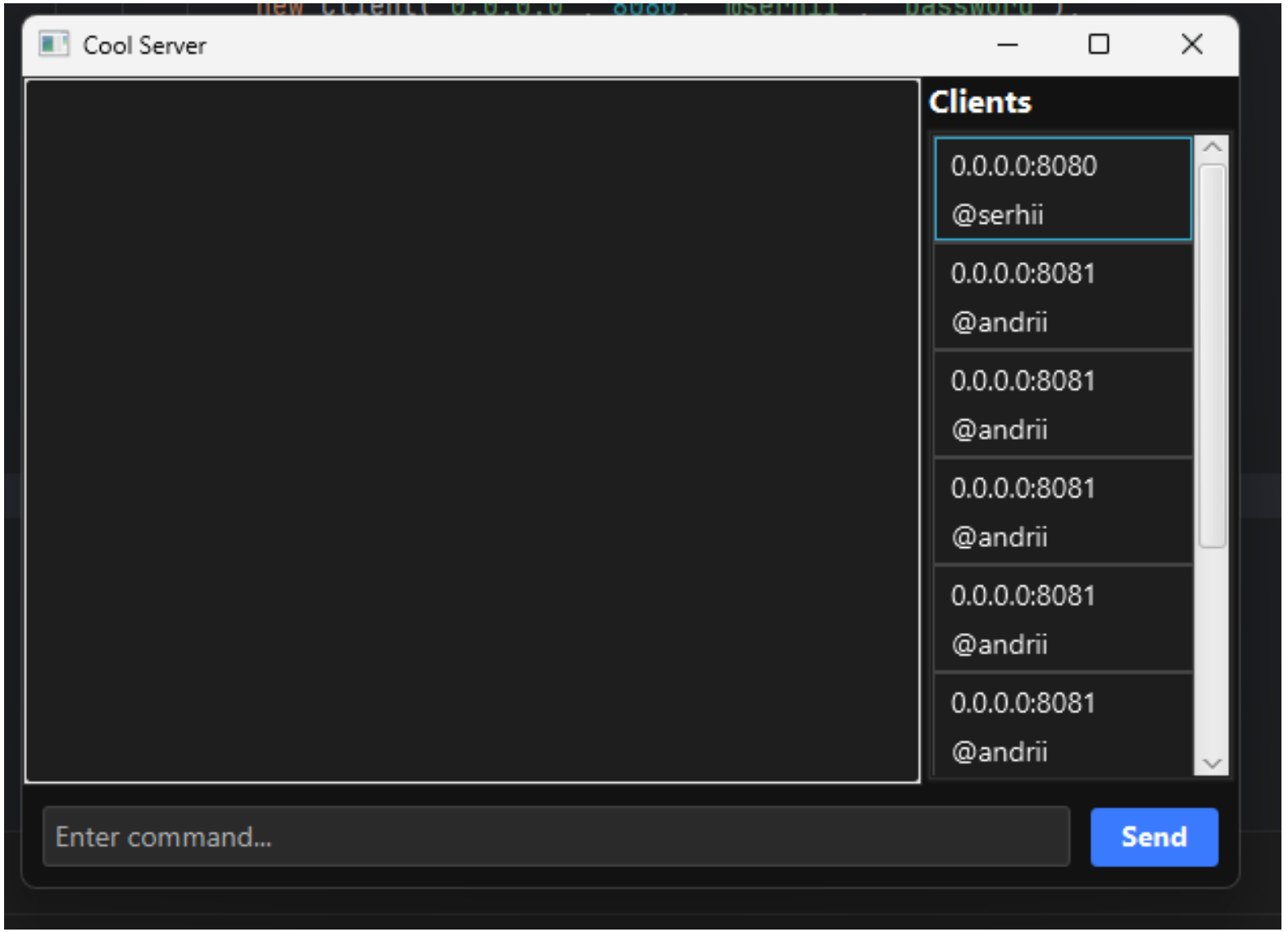

```java
public class ClientCell extends ListCell<Client> {

  private final VBox container;
  private final Text ipPortText;
  private final Text nicknameText;

  public ClientCell() {
    ipPortText = new Text();
    ipPortText.setFill(Color.WHITE);
    nicknameText = new Text();
    nicknameText.setFill(Color.WHITE);
    container = new VBox(5, ipPortText, nicknameText);
    this.setStyle("-fx-border-color: #444; -fx-border-width: 1 1 1 1;");
  }

  @Override
  protected void updateItem(Client client, boolean empty) {
    super.updateItem(client, empty);
    if (empty || client == null) {
      setGraphic(null);
    } else {
      ipPortText.setText(client.getIpAddress() + ":" + client.getPort());
      nicknameText.setText(client.getNickname());
      setGraphic(container);
    }
  }
}
```

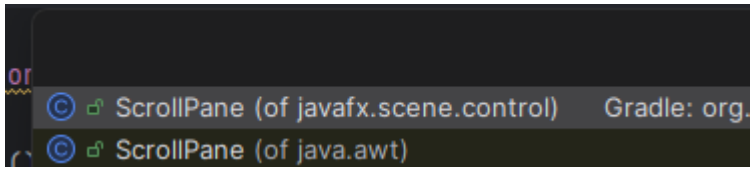
```java
public class ServerUI {

    public static final String MAIN_UI = "/server

    @FXML public TextArea consoleArea;
    @FXML public TextField commandField;
    @FXML public Button sendButton;
    @FXML public ListView<Client> clientList;

```

```java
@FXML
public void initialize() {
    clientList.setSelectionModel(null);  // не можна обирати клієнта
    clientList.setCellFactory(info -> new ClientCell());   // завод ячейок для клієнтів
    clientList.getItems().addAll(
        new Client("0.0.0.0", 8080, "@serhii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8082, "@taras", "password")
    );
}
```



Операції з юзерами






```
public class ServerConsole extends ScrollPane {

  public ServerConsole() {

  }

}
```

```
public class MessageBubble extends VBox {

  public MessageBubble() {

  }

}
```

```
@FXML public TextArea consoleArea;
```

Робимо аналогічний клієнту чат, але простіше. Видаляємо змінну TextArea. Заміняти її буде ServerConsole, який є нащадком ScrollPane (панель яку можна прокручувати). Всередині будуть MessageBubble.


```
<!-- Console -->
<center>
    <TextArea fx:id="consoleArea"
              editable="false"
              wrapText="true"
              BorderPane.alignment="CENTER"
              styleClass="console-area" />
</center>
```

MenuBar в програмі

```
<BorderPane xmlns="http://javafx.com/javafx/23.0.1"
            xmlns:fx="http://javafx.com/fxml/1"
            fx:controller="net.user.server.ServerUI"
            fx:id="root"
            stylesheets="@style.css">
```

```
@FXML public TextField commandField;
@FXML public Button sendButton;
@FXML public ListView<Client> clientList;
public ServerConsole console;
@FXML public BorderPane root;
```

```
public class ServerConsole extends ScrollPane {

    private final VBox container;

    public ServerConsole() {
        container = new VBox(5);   // 5 - піксельні відступи між елементами
        container.setPadding(new Insets(10));   // внутрішні відступи (всі по 10 px)
        this.setContent(container);   // встановити основною панелькою VBox
        this.setFitToWidth(true);   // розтягувати на всю ширину
        this.setVbarPolicy(ScrollBarPolicy.ALWAYS);   // завжди є смуга прокручення
    }

    public void addMessage(MessageBubble bubble) {
        container.getChildren().add(bubble);   // додати в список повідомлення
        this.layout();   // змусити оновитися панельку
        this.setVvalue(1.0);   // прокрутити в самий низ
    }

}
```

Додавання нікнейму

```java
public class MessageBubble extends VBox {

    public MessageBubble(String type, String msg) {
        // Відправник
        Text typeText = new Text(type + "\n");
        typeText.setStyle(
                "-fx-fill: red;" +
                        "-fx-font-size: 12;" +
                        "-fx-font-weight: bold;");

        // Відступ від відправника
        Text spacer = new Text("\n");
        spacer.setStyle("-fx-font-size: 4;");

        // Повідомлення
        Text messageText = new Text(msg);
        messageText.setStyle(
                "-fx-fill: white; -fx-font-size: 14;"
        );

        // Зшивач текстів
        TextFlow flow = new TextFlow(typeText, spacer, messageText);
        flow.setMaxWidth(300);   // Для того щоб повідомлення переносилися на нові рядки
        flow.setTextAlignment(TextAlignment.LEFT);
        flow.setStyle(
                "-fx-padding: 10;" +
```

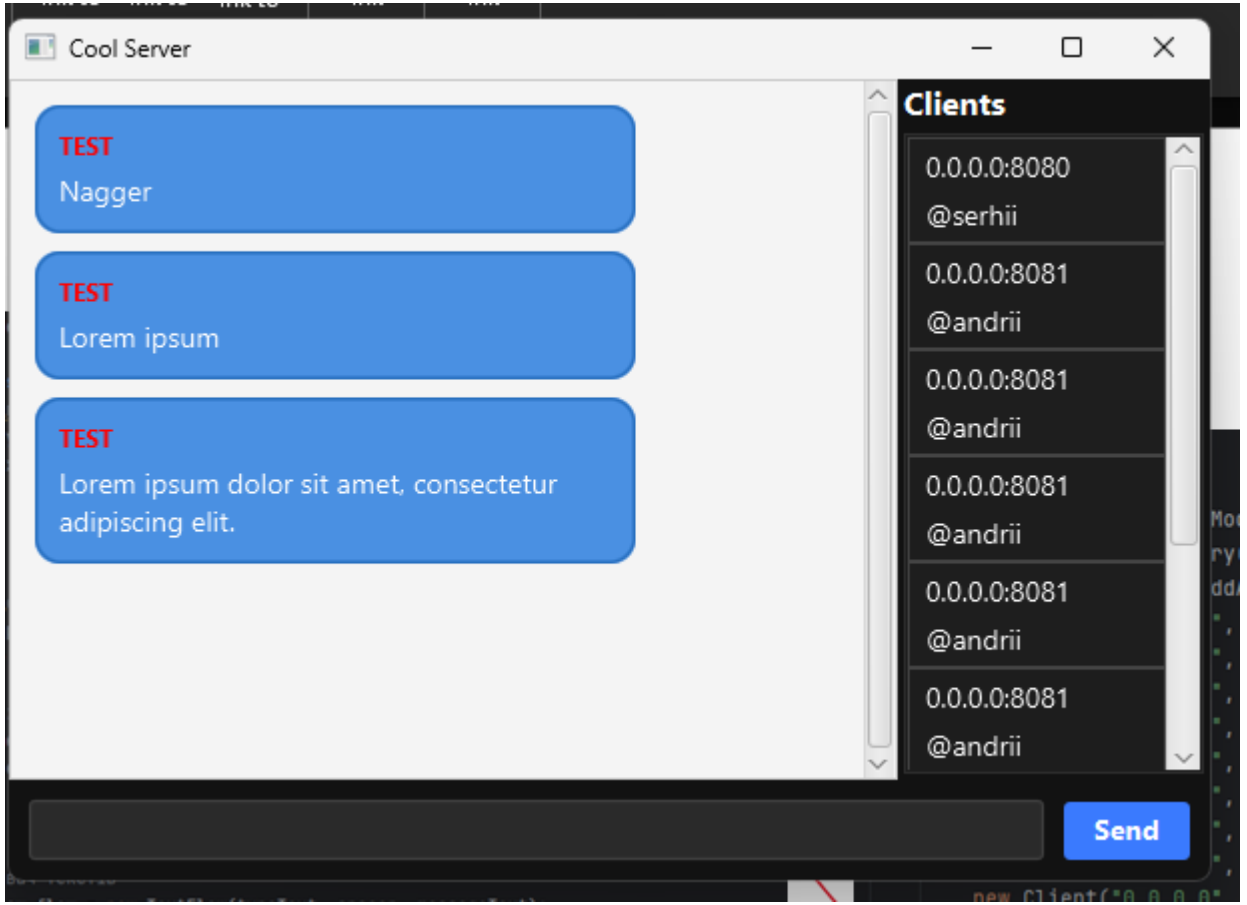
Показати все, велика програма

```java
@FXML
public void initialize() {
    clientList.setSelectionModel(null);  // не можна обирати клієнта
    clientList.setCellFactory(info -> new ClientCell());   // завод ячейок дл
    clientList.getItems().addAll(
        new Client("0.0.0.0", 8080, "@serhii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8081, "@andrii", "password"),
        new Client("0.0.0.0", 8082, "@taras", "password")
    );

    console = new ServerConsole();
    root.setCenter(console);
    console.addMessage(new MessageBubble("TEST", "Nagger"));
    console.addMessage(new MessageBubble("TEST", "Lorem ipsum"));
    console.addMessage(new MessageBubble("TEST", "Lorem ipsum dolor sit " +
            "amet, consectetur adipiscing elit."));
}
```

тежі тестові

Результат

- Тепер у нас є готовий інтерфейс сервера.
- Можемо далі робити програмну частину в наступний раз.
- Якщо є бажання і час, можете стилізувати кольори як хочете.
- **Тільки майте на увазі, що деякі стилі в .css, а деякі в коді – обираються за умов.**



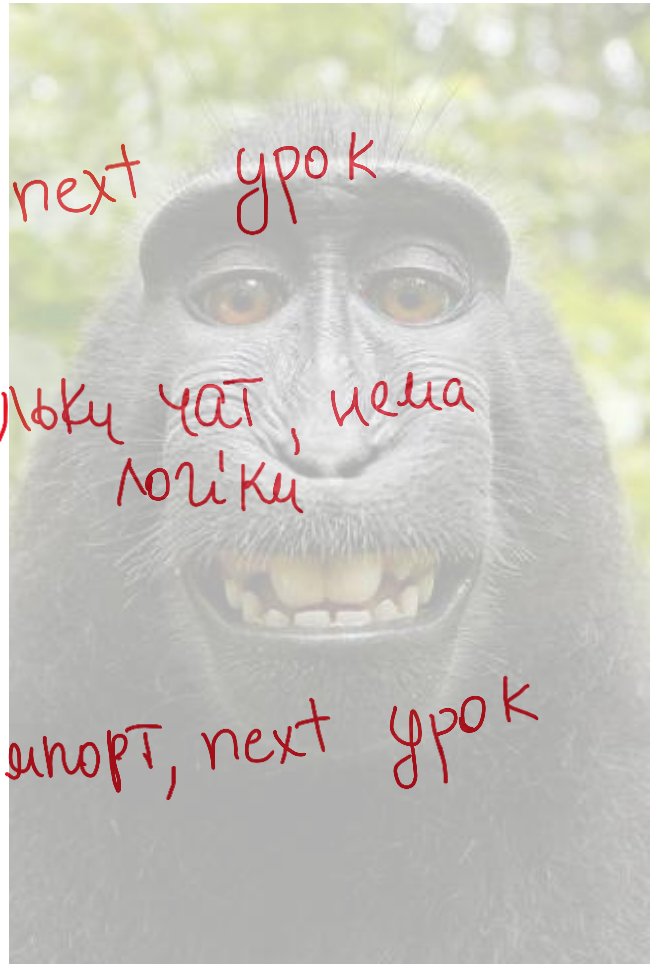

```
public static void loadUI(Consumer<Stage> stageLoader) {
    stageLoader.accept(PRIMARY_STAGE);
}
```

Спойлер – дана функція буде завантажувати в клієнта інтерфейси які побажаємо.

```
18
19  dependencies {
20      implementation group: 'org.slf4j', name: 'slf4j-api', version: '2.0.16'
21      implementation 'org.java-websocket:Java-WebSocket:1.5.3' // WebSocket бібліотека
22      implementation 'com.google.code.gson:gson:2.10.1' // Google JSON library
23
24      testImplementation platform('org.junit:junit-bom:5.10.0')
25      testImplementation 'org.junit.jupiter:junit-jupiter'
```

До **обох** проектів додаємо рядок з бібліотекою Json.
Жмемо на слона.

- можливість підключитися для клієнта
- показ на сервері всього що відбувається (підключення користувачів, додавання в друзі, виконання команд)
- зберігання даних (поки що Json)



next урок

тільки чат, нема логіки

імпорт, next урок

+ інтерфейс для входу