GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

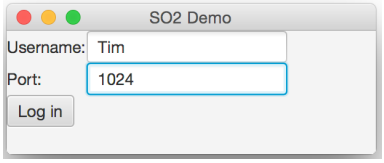
Lab 4 : GUIs

30 maart 2018
Stijn Verstichel
(e-mail: {voornaam.familienaam}@ugent.be)

1 Inleiding

In deze labsessie bouwen we een chattoepassing met een grafische front-end. Om de chatberichten tussen twee gebruikers te versturen, maken we gebruik van de genetwerkte `EventBroker` zoals we deze in de vorige labsessies gerealiseerd hebben. De grafische toepassing zal dus `Event`s genereren en ontvangen, die de chatboodschappen bevatten.
In deze labsessie wordt van de ontwikkelomgeving Eclipse gebruik gemaakt, samen met de SceneBuilder, de GUI-builder voor JavaFX. (die het mogelijk maakt om met relatief weinig inspanning snel een grafische toepassing te bouwen). We veronderstellen dat je bij aanvang van de labsessie de tutorial doorgenomen hebt, beschikbaar op `http://code.makery.ch/java/javafx-8-tutorial-part1/` (i.h.b. delen 1 en 2 van deze tutorial), zodat de tijd in de labsessie zelf efficiënt benut kan worden (of dat de kennis erin vervat via een alternatieve weg verworven werd). Zorg dat je een project aanmaakt van het type `JavaFX Project`, zodat automatisch de juiste grafische bibliotheken ingeladen worden in het project.

2 Login venster



Figuur 1: Login venster.

Opgave 1

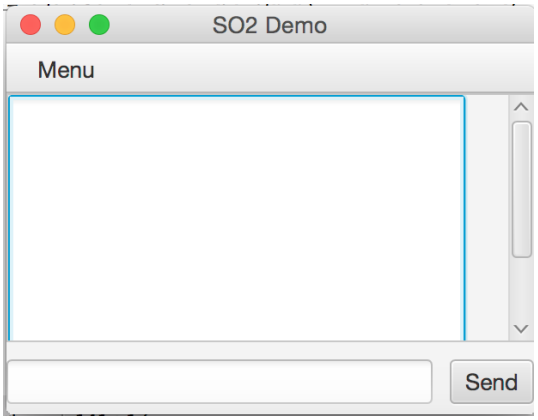
Maak zelf een nieuwe main klasse `main.Main`, en programmeer een loginvenster zoals Figuur 1 weergeeft. Hierbij wordt de naam van de gebruiker opgevraagd, samen met het poortnummer waarlangs de chattoepassing over het netwerk zal communiceren. Wanneer de “Log in”-knop aangeklikt wordt, wordt de invoer nagekeken op geldigheid. Is deze ongeldig, dan worden de tekstvelden gewist, en wacht de toepassing op een nieuwe klik op de knop “Log in”. Is de inhoud geldig, dan vervolgt de chattoepassing (zie Opgave 4). De invoer is ongeldig indien

- de lengte van de usernaam 0 karakters bedraagt
- het poortnummer geen geheel getal voorstelt
- het poortnummer wel een geheel getal voorstelt, maar dit getal niet in het bereik $0..2^{16}-1$ ligt

Bouw dit loginvenster ZONDER gebruik te maken van de GUI-builder. Kies een geschikte rootcontainer om dezelfde layout als in Figuur 1 te bekomen (b.v. `GridPane`). Maak best gebruik van een aparte `Scene` om de login widgets te groeperen. Zorg er ook voor dat de GUI correct geïnitialiseerd wordt (d.w.z. op de JavaFX application thread).

3 Chat panel

Eens ingelogd ziet de gebruiker een venster zoals weergegeven in Figuur 2. De root-container van dit venster is een `BorderPane`, waarin in het bovenste deel (`top`) een menubalk toegevoegd werd, en in het onderste deel (`bottom`) een `ChatPanel`. De overige delen (d.w.z. `left`, `right` en `center`) zijn voorlopig niet ingevuld (dit komt aan bod in de volgende labsessie). Het `ChatPanel` bestaat uit een venster dat de chat weergeeft, een tekst veld om zelf een bericht te typen en een knop om een bericht te versturen. In de volgende opgaven zullen we stapsgewijs hiernaartoe werken.



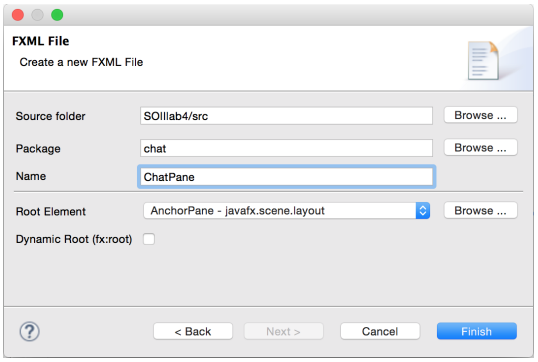
Figuur 2: Venster na inloggen, bestaande uit een menubalk en een chat panel.

Opgave 2

JavaFX maakt het mogelijk om de GUI-componenten, en de eventhandling van de widgets vast te leggen in een XML-formulier, met extensie `.fxml`. In plaats van de componenten programmatisch aan te maken, en de relatie tussen handlers en widgets via expliciete registratie van eventlisteners te realiseren, wordt het XML-formulier door een `FXMLLoader` ingeladen. Het resultaat van deze operatie is een container die alle widget elementen bevat. Daarnaast worden ook automatisch Controller-objecten aangemaakt, die de handling code bevatten. Onderstaand code-snippet laadt het formulier in, en maakt het resulterend `AnchorPane` aan.

```
FXMLLoader loader = new FXMLLoader();
loader.setLocation(HuidigeKlasse.class.getResource("NaamFormulier.fxml"));
chatAnchor = (AnchorPane) loader.load();
```

Omdat het editeren van XML-documenten dikwijls tot fouten aanleiding geeft, wordt deze taak uitbesteed aan de SceneBuilder. Deze editor maakt het mogelijk om componenten via drag-and-drop aan containers toe te voegen, deze te configureren, en indien nodig de handling code te specificeren. Maak daarom binnen de package `chat` een `fxml`-document aan via "**File** >**New** >**Other ...**" en kies "New FXML Document" binnen de categorie "Java FX". Kies als naam "ChatPane.fxml". Vervolgens klik je met de rechtermuisknop op dit document, en selecteer je "Open with SceneBuilder".



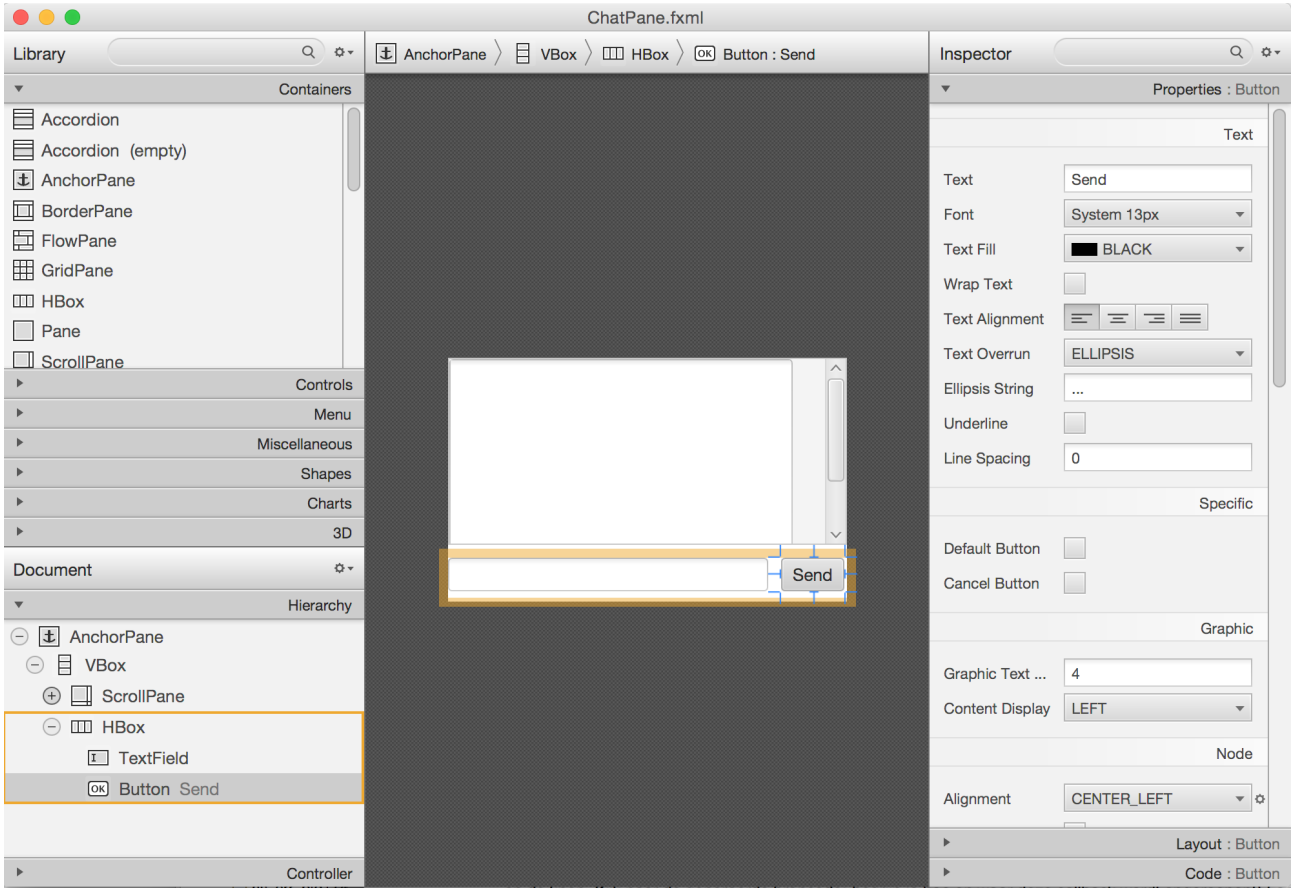
Figuur 3: Aanmaken van een nieuw FXML-document.

Bouw nu via de SceneBuilder zo goed mogelijk de configuratie na die getoond werd in figuur 2 (en beperk je hierbij tot het onderste deel).

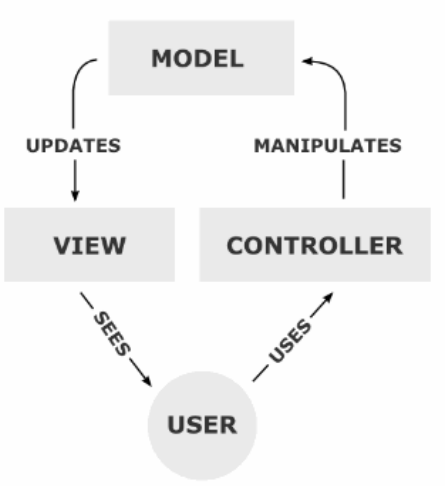
Ontwerp het chat panel zoals weergegeven in Figuur 4. Let op: de SceneBuilder en Eclipse zijn enkel gekoppeld via het opgeslagen `.fxml`-formulier. Wanneer je code start in Eclipse is het dan ook aan te raden om het project te refreshen, zeker als er aanpassingen gebeurden in het formulier via de SceneBuilder.

4 Model-View-Controller

Het Model-View-Controller patroon is een veelgebruikt patroon met als doel de GUI (= View) te scheiden van de data die deze weergeeft (= Model), en de achterliggende logica (= Controller). De werking ervan wordt weergegeven in Figuur 5. De gebruiker kijkt naar de View, en triggert acties van de Controller. De Controller past indien nodig het Model aan, en het Model notificeert de View dat deze gewijzigd is, en dat de GUI up to date moet worden gebracht.



Figuur 4: Opbouw van het chat panel via de SceneBuilder: bibliotheek van componenten (linksboven), component hiërarchie (linksonder), visuele voorstelling (centraal), configuratie van componenten (rechts).



Figuur 5: Model-view-controller.

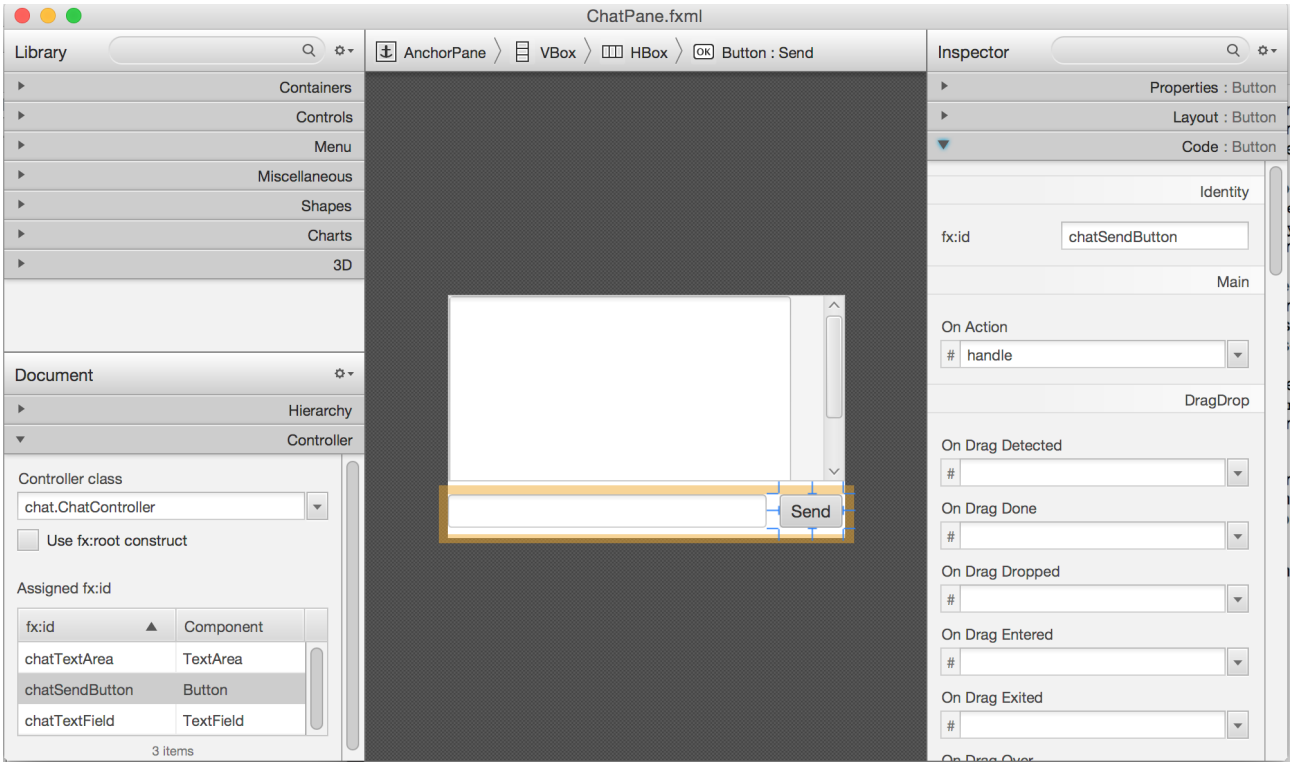
Opgave 3

In deze opgave bouwen we de chattoepassing zelf via het MVC-patroon (Figuur 5), waarbij de View-component het resultaat is van opgave 2. Het `chat.ChatModel` houdt dus alle relevante gegevens bij van de chatsessie, namelijk:

- de naam van de gebruiker die de chatsessie startte (voorzie een getter/setter-paar)
- de berichten van de chatsessie

Een bericht van de chatsessie wordt voorgesteld als een instantie van de klasse `chat.ChatMessage`. Een object van die klasse houdt de zender (= zijn naam) en de inhoud van het bericht bij. Aangezien chatberichten ook over het netwerk dienen verstuurd met behulp van de `EventBroker` uit vorige lab sessies, erft deze klasse over van de klasse `eventbroker.Event`.
Telkens de toestand van het `chat.ChatModel` wijzigt, wordt dit via het Observer-patroon doorgegeven aan mogelijke luisteraars. Hiervoor kan het gebruik maken van de FX `Properties` die de interface `ObservableValue` implementeren. De bedoeling is dan om een dergelijke `Property` van het model te binden aan een `Property` van de view. Op die manier worden wijzigingen rechtstreeks vanuit het model doorgegeven aan de view. In deze labsessie zullen we vooral werken met de klassen `StringProperty` en `SimpleStringProperty`.
De `chat.ChatController` implementeert de logica van de chattoepassing, en is verantwoordelijk voor het toevoegen van chatberichten aan `chat.ChatModel`. Daartoe luistert deze naar berichten die verstuurd worden vanuit de GUI enerzijds, en berichten die binnenkomen van andere gebruikers over het netwerk anderzijds. Belangrijk is zich te realiseren dat hier twee eventsystemen actief zijn (namelijk het ingebouwde FX eventsysteem, en het zelf geprogrammeerde eventstysteem dat in staat is om events over een netwerk te versturen.
Om de koppeling tussen de `ChatController` en de view mogelijk te maken, moet met elk relevant widget uit de view een widgetreferentie in de `ChatController` geassocieerd worden. Dit gebeurt door in de `ChatController` een geschikte referentie te annoteren met `@FXML`, en vervolgens in de SceneBuilder de verbinding te maken tussen het widget en de referentie van de `ChatController`. In Figuur 6 wordt dit process geïllustreerd. Naast het verbinden van referenties in de controller met widgets in de view, kan je ook de handling code specificeren. Voor het voorbeeld getoond in Figuur 6 is de corresponderende code uit de `ChatController`

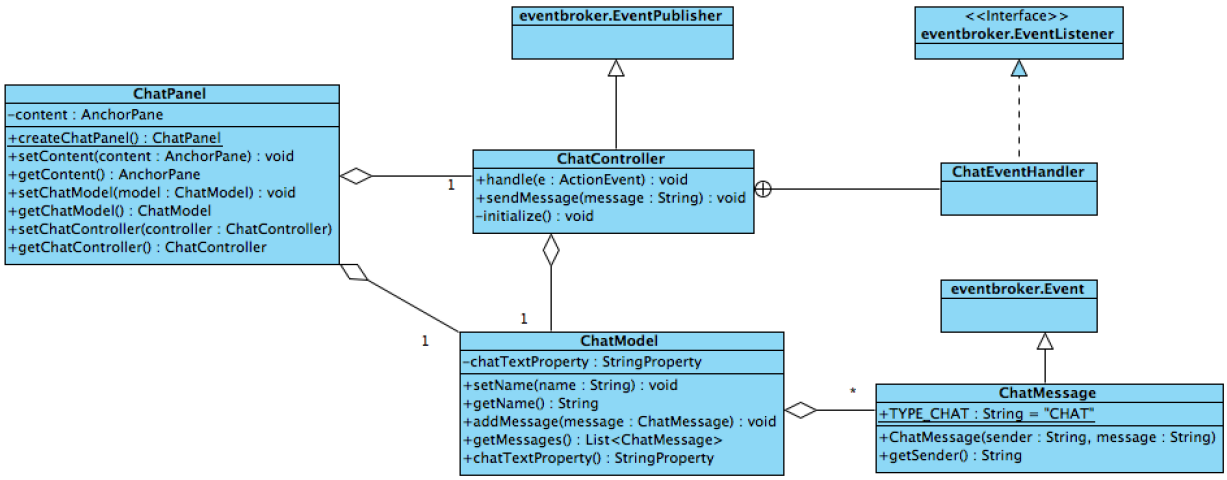
```
final public class ChatController extends EventPublisher {
        @FXML
        private Button chatSendButton;

        public void handle(ActionEvent e){
                // handling method
        }
}
```



Figuur 6: Verbinden van view en controller in de SceneBuilder: verbinden van referenties en aangeven van event handling code.

- Wanneer op de “send” knop gedrukt wordt op de GUI, zal deze via de callback het bericht doorgeven aan `chat.Controller` met behulp van de `sendMessage()` methode. De `Controller` publiceert dit bij de `EventBroker` als nieuw event, waardoor het over het netwerk verstuurd zal worden.
- De `ChatEventHandler` (inwendige klasse van `Controller`) luistert tevens naar `Events` van `EventBroker`, en indien er een chatbericht binnenkomt wordt dit toegevoegd aan het `ChatModel`.

Onderstaand UML-klassendiagram (Figuur 7) geeft een mogelijk ontwerp van de chattoepassing.



Figuur 7: Chattoepassing: Model-view-controller.

Implementeer de klassen `chat.ChatMessage`, `chat.Controller` en `chat.ChatModel`. Voeg ook de factory methode `public static ChatPanel createChatPanel(String name)` toe aan de klasse `chat.ChatPanel`, die een `ChatPanel`, `Controller` en `ChatModel` aanmaakt en de juiste luisteraars configureert. Deze methode laadt dus het aangemaakte FXML-formulier dat in opgave 2 aangemaakt werd. Bij het inladen zal ook automatisch een `ChatController`-object aangemaakt worden. Na constructie van dit object zal automatisch de methode

```
class ChatController {
      @FXML
      private void initialize(){
                // intialisatiecode
      }
}
```

opgeroepen worden. In deze methode kan je dus de nodige initialisatie-acties nemen. Een belangrijke actie in deze context is om de `StringProperty` van het het model te verbinden met de `StringProperty` van de view. Op die manier worden wijzigingen aan het model doorgegeven aan de view. Dit kan via de methode `bind()` van de klasse `Property`.

Let op: Zoals je weet is geen enkele GUI-bibliotheek (en in het bijzonder dus Java FX) thread safe. Dit betekent dat alle acties die de inhoud van de GUI veranderen, moeten gebeuren vanop de FX Application Thread. Om er zeker van te zijn dat een opdracht op deze thread gebeurt, kan je in JavaFX gebruik maken van de constructie:

```
Platform.runLater(new Runnable(){
     public void run(){
         // opdrachten uit te voeren op de FX Application Thread
     }
});
```

Om het chat panel te testen dienen we dit te tonen na inloggen, wat we behandelen in Opgave 4.

Opgave 4

In deze opgave integreren we alle delen tot een werkende chatapplicatie. Gebruik hiervoor de multithreaded event broker (package `eventbroker`) uit lab sessie 2, en het netwerk subsysteem (package `network`) uit lab sessie 3.

Wanneer de gebruiker succesvol inlogt, dienen de volgende acties te gebeuren:

- De `EventBroker` wordt gestart (zie lab sessie 2 voor de multithreaded `EventBroker`).

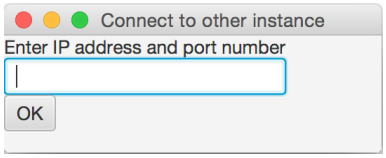
- Een nieuwe instantie van `Network` wordt aangemaakt, luisterend op de poort meegegeven bij het inloggen.
- Een `ChatPanel` (en bijhorende `Model` en `Controller`) wordt aangemaakt
- Een `BorderPane` wordt aangemaakt, waarin onderaan de inhoud (van het type `AchorPane`) van de `ChatPanel` wordt toegevoegd, en bovenaan een menubalk.
- Een nieuwe `Scene` wordt aangemaakt en getoond (namelijk de `Scene` uit figuur 2).

Implementeer deze stappen na succesvol inloggen. Om te testen of het chatten werkt, connecteer je twee locale instanties met behulp van de `connect()` methode van de klasse `Network`. Gebruik hierbij voorlopig vast gecodeerde IP-adressen (localhost) en poortnummers. In de volgende opgave voorzien we een menubalk die toelaat dit via de GUI te doen.

Merk op dat in dit geval beide instanties van de applicatie zowel de rol van client als server kunnen opnemen. Wanneer een connectie is opgezet, moeten beiden tegelijkertijd berichten kunnen versturen en ontvangen.

5 Menus

Het venster weergegeven in Figuur 2 heeft ook een menu bar, bestaande uit één drop-down menu “Menu”. Dit bevat één menu-item “Connect...” dat het onderstaande dialoogvenster toont (Figuur 8), hetgeen toelaat te connecteren naar een willekeurige andere client, gespecificeerd via IP-adres en poortnummer.



Figuur 8: Connect dialog.

Opgave 5

- Voeg aan het menu “Menu” het menu-item “Connect...” toe.
- Bij het klikken “Connect...”, wordt een dialoogvenster getoond zoals weergegeven in Figuur 8, waar de gebruiker een IP-adres en poortnummer kan ingeven om mee te connecteren, waarna een verbinding met een client op die locatie opgezet wordt.

6 Tot slot

Upload je bestanden naar Indianio. We verwachten volgende bestanden:

- package `main`
  - `Main.java`: het hoofdprogramma dat je toepassing aanstuurt.
- package `chat`
  - `ChatMessage.java`
  - `ChatPanel.java`
  - `ChatPane.fxml`
  - `ChatModel.java`
  - `ChatController.java`
- packages `eventbroker` en `network`

Mogelijke uitbreidingen:

- Geef de gebruiker foutboodschappen wanneer hij ongeldige parameters ingeeft bij het inloggen of connecteren.
- Zorg dat de gebruiker ook een bericht kan versturen bij het drukken op de “enter” toets.