Dawid Waligórski, 264015

Michał Dziedziak, 263901

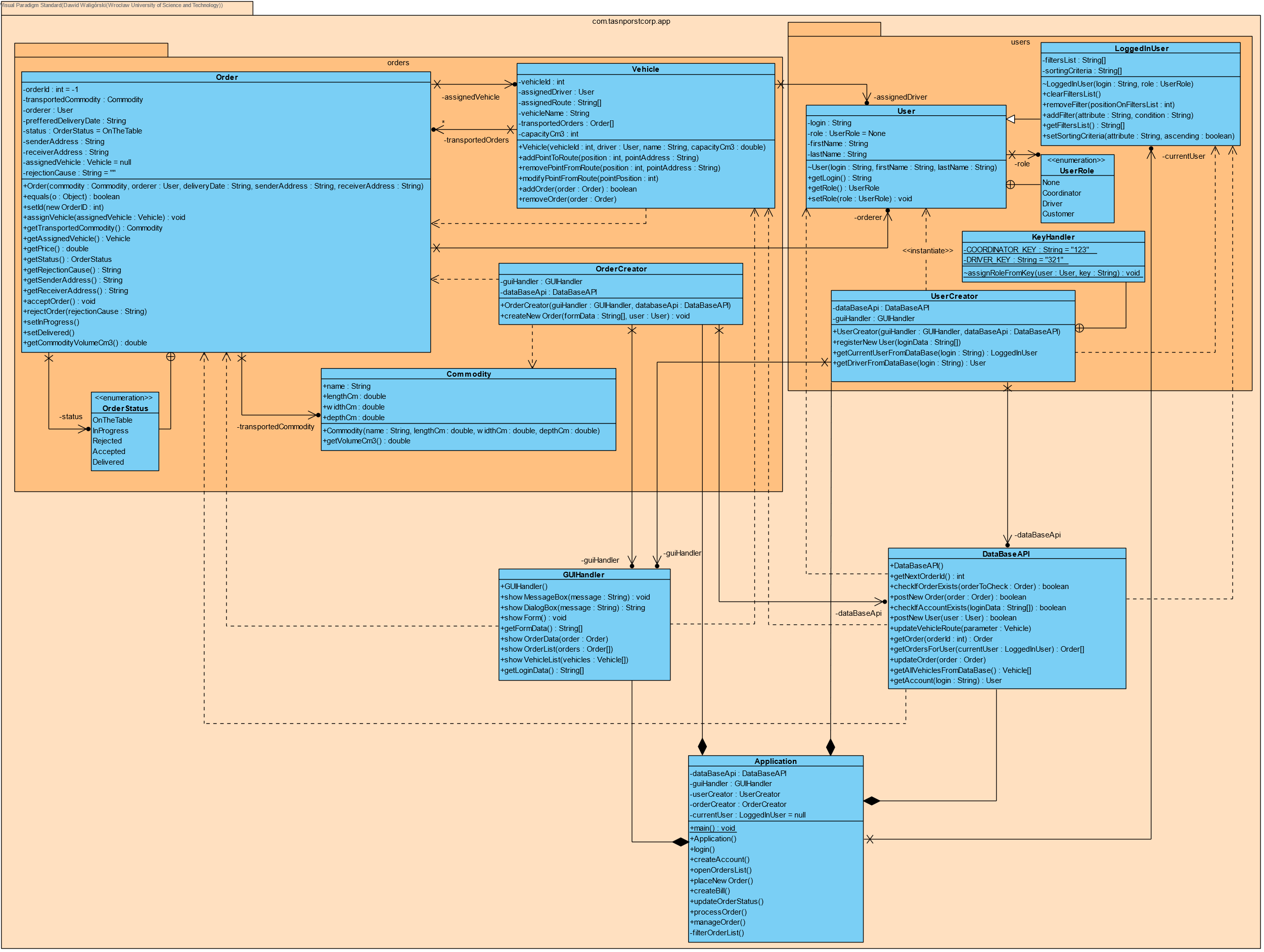
**Laboratorium 7, 8, 9, 10**

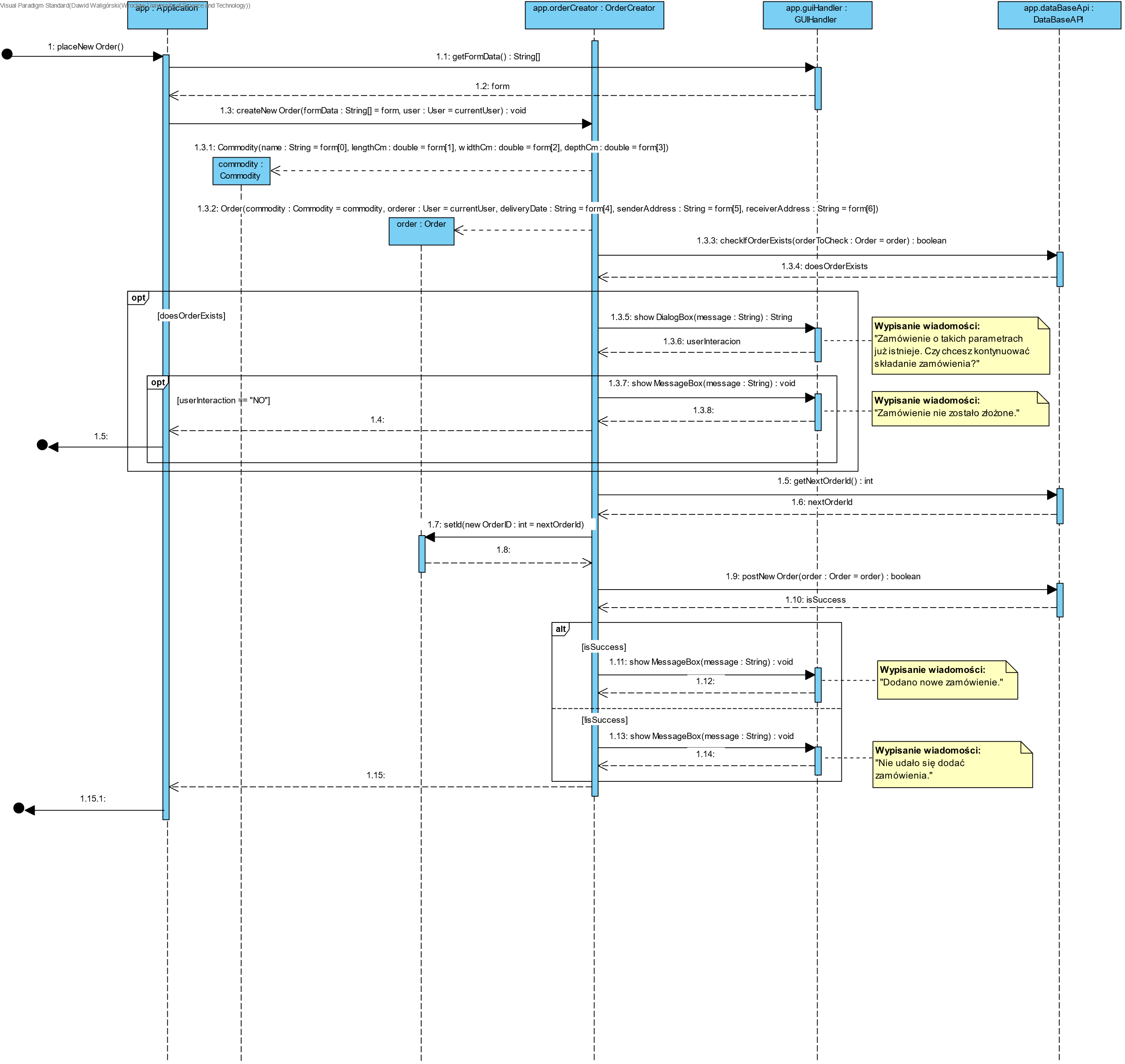
* Identyfikacja klas reprezentujących logikę biznesową projektowanego oprogramowania. Definicja atrybutów i operacji klas oraz związków między klasami na podstawie analizy scenariuszy przypadków użycia. Opracowanie diagramów klas i pakietów. Zastosowanie projektowych wzorców strukturalnych i wytwórczych.
* Opracowanie diagramów sekwencji dla wybranych przypadków użycia, reprezentujących usługi oprogramowania, wynikających również z wykonanych diagramów czynności. Definicja operacji klas na podstawie diagramów sekwencji w języku Java. Zastosowanie projektowych wzorców zachowania.
* Opracowanie diagramów sekwencji dla wybranych przypadków użycia reprezentujących usługi oprogramowania wynikających również z wykonanych diagramów czynności. Definicja operacji klas na podstawie diagramów sekwencji w języku Java. Zastosowanie projektowych wzorców zachowania.

**Temat projektu**

Program obsługujący zakład transportowy.

**Diagram klas:**



**Diagram sekwencji metody „placeOrder” (PU „Składanie zamówienia”):**

Kod metody placeNewOrder z klasy Application zamodelowanej przez diagram sekwencji:

**public** **void** placeNewOrder() {

String[] formData = guiHandler.getFormData();

orderCreator.createNewOrder(formData, currentUser);

}

Kod metody createNewOrder z klasy OrderCreator zamodelowanej przez diagram sekwencji:

**public** **void** createNewOrder(String[] formData, User user) {

var commodity = **new** Commodity(

formData[0],

Double.parseDouble(formData[1]),

Double.parseDouble(formData[2]),

Double.parseDouble(formData[3])

);

var order = **new** Order(

commodity,

user,

formData[4],

formData[5],

formData[6]

);

**boolean** doesOrderExists = dataBaseApi.checkIfOrderExists(order);

**if**(doesOrderExists)

{

String userInteraction = guiHandler.showDialogBox(

"Zamówienie o takich parametrach już istnieje. Czy chcesz

kontynuować składanie zamówienia?"

);

**if**(userInteraction.contains("NO"))

{

**return**;

}

}

**int** nextOrderId = dataBaseApi.getNextOrderId();

order.setId(nextOrderId);

**boolean** isSuccess = dataBaseApi.postNewOrder(order);

**if**(isSuccess)

{

guiHandler.showMessageBox("Dodano nowe zamówienie");

}

**else**

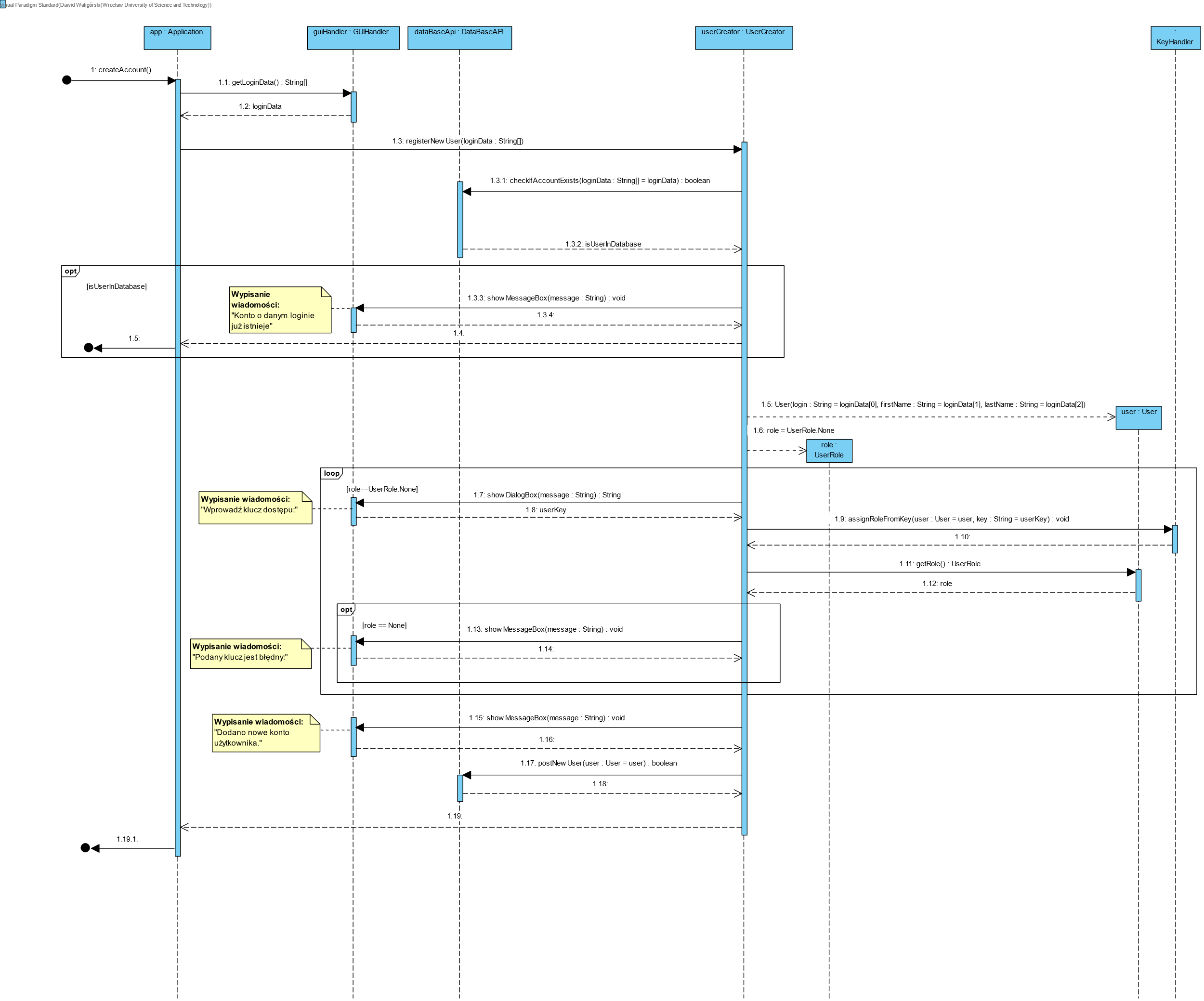
{

guiHandler.showMessageBox("Nie udało się dodać zamówienia");

}

}

**Diagram sekwencji metody „createAccount” (PU „Załozenie konta”):**



Kod metody createAccount z klasy Application zamodelowanej przez diagram sekwencji:

**public** **void** createAccount() {

String[] loginData = guiHandler.getLoginData();

userCreator.registerNewUser(loginData);

}

Kod metody registerNewUser z klasy UserCreator zamodelowanej przez diagram sekwencji:

**public** **void** registerNewUser(String[] loginData) {

**boolean** isUserInDatabase = dataBaseApi.checkIfAccountExists(loginData);

**if**(isUserInDatabase) {

guiHandler.showDialogBox("Konto o danym loginie już istnieje");

**return**;

}

User user = **new** User(loginData[0], loginData[1], loginData[2]);

UserRole role = UserRole.None;

**while** (role == UserRole.None) {

String userKey = guiHandler.showDialogBox("Wprowadź klucz dostępu.");

KeyHandler.assignRoleFromKey(user, userKey);

role = user.getRole();

**if**(role == UserRole.None)

guiHandler.showMessageBox("Podany klucz jest błędny.");

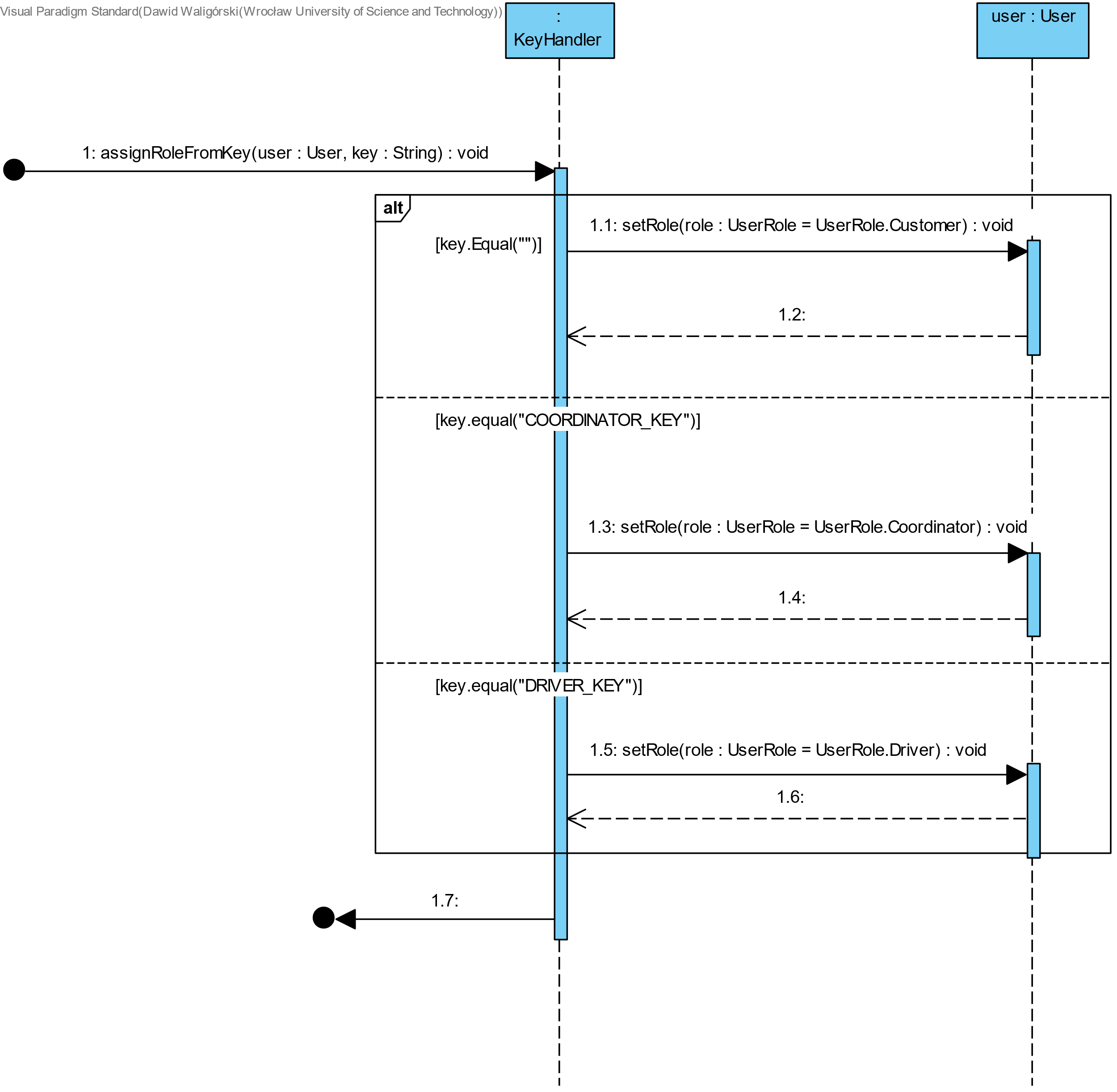
}

guiHandler.showMessageBox("Dodano nowe konto użytkownika!");

dataBaseApi.postNewUser(user);

}

Dla czytelności z diagramu modelującego metodę createAccount wydzielono do osobnego diagramu sekwencji metodę assignRoleFromKey z klasy KeyHandler.



Kod metody assignRoleFromKey z klasy KeyHandler zamodelowanej przez diagram sekwencji:

**static** **void** assignRoleFromKey(User user, String key) {

**if**(key.equals("")){

user.setRole(UserRole.Customer);

}

**else** **if**(key.equals(COORDINATOR\_KEY)){

user.setRole(UserRole.Coordinator);

}

**else** **if**(key.equals(DRIVER\_KEY)){

user.setRole(UserRole.Driver);

}

**return**;

}