Esercizio 4
Verbalizzazione esami

Tosetti Luca

Web Technologies

Traccia 1/2

Un’applicazione permette di verbalizzare gli esiti degli esami di un appello.

Il docente accede tramite login e seleziona nella HOME page un corso da una lista dei propri corsi ordinata in modo alfabetico decrescente e poi una data d’appello del corso scelto selezionata da un elenco ordinato per data decrescente. Ogni corso ha un solo docente.

La selezione dell’appello porta a una pagina ISCRITTI, che mostra una tabella con tutti gli iscritti all’appello. La tabella riporta i seguenti dati: matricola, cognome e nome, email, corso di laurea, voto e stato di valutazione. Il voto può non essere ancora definito. Lo stato di valutazione dello studente rispetto all’appello può assumere i valori: non inserito, inserito, pubblicato, rifiutato e verbalizzato.

Selezionando un’etichetta nell’intestazione della tabella, l’utente ordina le righe in base al valore di tale etichetta (ad esempio, selezionando “cognome” la tabella è riordinata in base al cognome). Successive selezioni della stessa etichetta invertono l’ordinamento: si parte con l’ordinamento crescente. Il valore del voto viene considerato ordinato nel modo seguente: <vuoto>, assente, rimandato, riprovato, 18, 19, …, 30, 30 e lode.

Nella tabella della pagina ISCRITTI ad ogni riga corrisponde un bottone “MODIFICA”. Premendo il bottone compare una pagina con una form che mostra tutti i dati dello studente selezionato e un campo di input in cui è possibile scegliere il voto. L’invio della form provoca la modifica o l’inserimento del voto. Inizialmente le righe sono nello stato di valutazione “non inserito”. L’inserimento e le successive eventuali modifiche portano la riga nello stato di valutazione “inserito”.

Traccia 2/2

Alla tabella della pagina ISCRITTI è associato un bottone PUBBLICA che comporta la pubblicazione delle righe con lo stato di valutazione INSERITO. La pubblicazione rende il voto non più modificabile dal docente e visibile allo studente e cambia lo stato di valutazione della riga dello studente a “pubblicato”.

Lo studente accede tramite login e seleziona nella HOME page un corso tra quelli a cui è iscritto mediante una lista ordinata in modo alfabetico decrescente e poi una data d’appello del corso scelto selezionata da un elenco ordinato per data decrescente.

Uno studente può essere iscritto a più appelli dello stesso corso. La selezione della data d’appello porta a una pagina ESITO che mostra il messaggio “Voto non ancora definito” se il docente non ha ancora pubblicato il risultato per quello studente in quell’appello. Altrimenti, la pagina mostra i dati dello studente, del corso, dell’appello e il voto assegnato.

Se il voto è tra 18 e 30 e lode compare un bottone RIFIUTA. Premendo tale bottone la pagina mostra gli stessi dati con la dizione aggiunta “Il voto è stato rifiutato” e senza il bottone RIFIUTA. Il rifiuto del voto cambia lo stato di valutazione a “rifiutato” della riga dello studente per quell’appello nella pagina ISCRITTI del docente.

Nella pagina ISCRITTI del docente la tabella degli iscritti è associata anche a un bottone VERBALIZZA. La pressione del bottone provoca il cambio di stato a “verbalizzato” per le righe nello stato “pubblicato” o "rifiutato" e comporta anche la creazione di un verbale e la disabilitazione della possibilità di rifiutare il voto.

Il rifiuto implica la verbalizzazione di “rimandato” come voto. Un verbale ha un codice generato dal sistema, una data e ora di creazione ed è associato all’appello del corso a cui si riferisce e agli studenti (con nome, cognome, matricola e voto) che passano allo stato “verbalizzato”. A seguito della pressione del bottone VERBALIZZA compare una pagina VERBALE che mostra i dati completi del verbale creato.

Analisi requisiti DB 1/2

Un'applicazione permette di verbalizzare gli esiti degli esami di un appello.

Il docente accede tramite login e seleziona nella HOME page un corso da una lista dei propri corsi ordinata in modo alfabetico decrescente e poi una data d'appello del corso scelto selezionata da un elenco ordinato per data decrescente. Ogni corso ha un solo docente.

La selezione dell'appello porta a una pagina ISCRITTI, che mostra una tabella con tutti gli iscritti all'appello. La tabella riporta i seguenti dati: matricola, cognome e nome, email, corso di laurea, voto e stato di valutazione. Il voto può non essere ancora definito. Lo stato di valutazione dello studente rispetto all'appello può assumere i valori: non inserito, inserito, pubblicato, rifiutato e verbalizzato.

Selezionando un'etichetta nell'intestazione della tabella, l'utente ordina le righe in base al valore di tale etichetta (ad esempio, selezionando "cognome" la tabella è riordinata in base al cognome). Successive selezioni della stessa etichetta invertono l'ordinamento: si parte con l'ordinamento crescente. Il valore del voto viene considerato ordinato nel modo seguente: <vuoto>, assente, rimandato, riprovato, 18, 19, …, 30, 30 e lode.

Nella tabella della pagina ISCRITTI ad ogni riga corrisponde un bottone "MODIFICA". Premendo il bottone compare una pagina con una form che mostra tutti i dati dello studente selezionato e un campo di input in cui è possibile scegliere il voto. L'invio della form provoca la modifica o l'inserimento del voto. Inizialmente le righe sono nello stato di valutazione "non inserito". L'inserimento e le successive eventuali modifiche portano la riga nello stato di valutazione "inserito ".

Entità, Attributi, Relazioni

Analisi requisiti DB 2/2

Alla tabella della pagina ISCRITTI è associato un bottone PUBBLICA che comporta la pubblicazione delle righe con lo stato di valutazione INSERITO. La pubblicazione rende il voto non più modificabile dal docente e visibile allo studente e cambia lo stato di valutazione della riga dello studente a “pubblicato”.

Lo studente accede tramite login e seleziona nella HOME page un corso tra quelli a cui è iscritto mediante una lista ordinata in modo alfabetico decrescente e poi una data d’appello del corso scelto selezionata da un elenco ordinato per data decrescente.

Uno studente può essere iscritto a più appelli dello stesso corso. La selezione della data d’appello porta a una pagina ESITO che mostra il messaggio “Voto non ancora definito” se il docente non ha ancora pubblicato il risultato per quello studente in quell’appello. Altrimenti, la pagina mostra i dati dello studente, del corso, dell’appello e il voto assegnato.

Se il voto è tra 18 e 30 e lode compare un bottone RIFIUTA. Premendo tale bottone la pagina mostra gli stessi dati con la dizione aggiunta “Il voto è stato rifiutato” e senza il bottone RIFIUTA. Il rifiuto del voto cambia lo stato di valutazione a “rifiutato” della riga dello studente per quell’appello nella pagina ISCRITTI del docente.

Nella pagina ISCRITTI del docente la tabella degli iscritti è associata anche a un bottone VERBALIZZA. La pressione del bottone provoca il cambio di stato a “verbalizzato” per le righe nello stato “pubblicato” o "rifiutato" e comporta anche la creazione di un verbale e la disabilitazione della possibilità di rifiutare il voto.

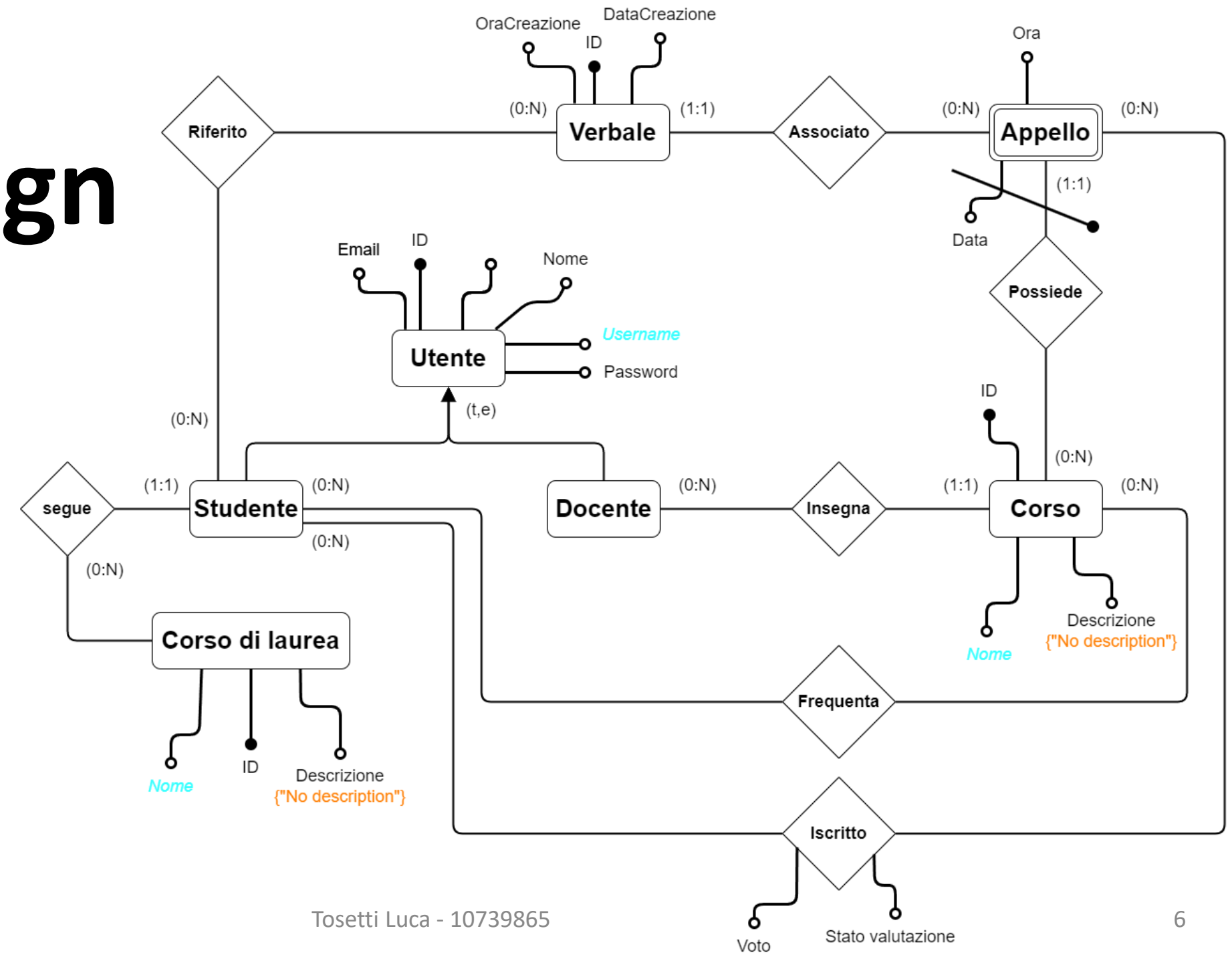
Il rifiuto implica la verbalizzazione di “rimandato” come voto. Un verbale ha un codice generato dal sistema, una data e ora di creazione ed è associato all’appello del corso a cui si riferisce e agli studenti (con nome, cognome, matricola e voto) che passano allo stato “verbalizzato”. A seguito della pressione del bottone VERBALIZZA compare una pagina VERBALE che mostra i dati completi del verbale creato.

Entità, Attributi, Relazioni

DB design

Default value

Unique value



Schema DB 1/5

Commento(ITA)

Per l'implementazione della tabella Appelli, ho valutato 2 possibili alternative:

- Mantenere la relazione di entità debole, e quindi realizzare la tabella in modo che la chiave primaria sia costituita dall'attributo OraData (Unione dei due attributi "Ora" e "Data") e da ID_Corso (chiave primaria della tabella Corso), questo permetterebbe di associare più appelli ad uno stesso corso, a patto che siano in momenti diversi (OraData diversi). Il lato negativo di questa implementazione è che non sarebbe possibile specificare che più appelli, relativi allo stesso corso, possano essere tenuti alla stessa data e ora.
- (Scelta adottata) Trasformare la relazione di entità debole in una normale relazione 1:N introducendo un attributo "ID", all'interno della tabella Appelli. In questo modo a differenza del caso precedente, posso anche andare a specificare la concomitanza di due appelli relativi allo stesso corso ed inoltre posso tenere separati gli attributi "Ora" e "Data" per maggiore flessibilità nelle query.

Tables(ENG)

For the Calls table I evaluated 2 possible alternatives:

- Maintain the weak relationship, and then create the table so that the primary key consists of the TimeDate attribute (Union of the "Time" and "Date" attributes) and ID_Course (primary key of the Course table). This allows to specify that multiple calls can be associated with a course, provided that they are at different times (TimeDate must be different). The negative part of this implementation is that it's not possible to specify that multiple calls, related to the same course, can be held on the same date and time.
- (Choice adopted) Transform the weak relationship into a normal 1:N relationship by introducing an "ID" attribute, in the Calls table. In this way, unlike the precedent case, I can specify the concurrence of two calls related to the same course and I can also keep the "Time" and "Date" attributes separated for more flexibility in the Querys

Schema DB 2/5

Chiavi esterne(ITA)

- corsi.ID_Docente -> utenti.ID
- studenti_verbali.ID_Studente -> utenti.ID
- studenti_verbali.ID_Verbale -> verbali.ID
- verbali.ID_Appello -> appelli.ID
- appelli.ID_Corso -> corsi.ID
- IscrizioniAppelli.ID_Studente -> utenti.ID
- IscrizioniAppelli.ID_Appello -> appelli.ID
- iscrizioniCorsi.ID_Studente -> utenti.ID
- iscrizioniCorsi.ID_Appello -> appelli.ID
- utenti.ID_CorsoDiLaurea -> CorsiDiLaurea.ID

Foreign keys(ENG)

- calls.ID_Course -> courses.ID
- courses.ID_Lecturer -> lecturers.ID
- Registrations_calls.ID_Student -> students.ID
- Registrations_calls.ID_Call -> calls.ID
- Registrations_courses.ID_Student -> students.ID
- Registrations_courses.ID_Call -> calls.ID
- users.ID_DegreeCourse -> degree_courses.ID
- students_verbals.ID_Student -> students.ID
- students_verbals.ID_Verbal -> verbals.ID
- verbals.ID_Call -> calls.ID

Schema DB

Calls table

```
CREATE TABLE `calls` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Date` date NOT NULL,
  `Time` time NOT NULL,
  `ID_Course` int NOT NULL,
  PRIMARY KEY (`ID`), KEY `calls->courses_idx` (`ID_Course`), CONSTRAINT `calls->courses` FOREIGN KEY (`ID_Course`) REFERENCES `courses` (`ID`) ON UPDATE CASCADE
);
```

Courses table

```
CREATE TABLE `courses` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Name` varchar(64) NOT NULL,
  `Description` varchar(1024) NOT NULL DEFAULT 'No description',
  `ID_Lecturer` int NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `Name_UNIQUE` (`Name`),
  KEY `Courses->Lecturers_idx` (`ID_Lecturer`), CONSTRAINT `Courses->Lecturers` FOREIGN KEY (`ID_Lecturer`) REFERENCES `users` (`ID`) ON UPDATE CASCADE
);
```

Degree courses table

```
CREATE TABLE `degree_courses` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Name` varchar(64) NOT NULL,
  `Description` varchar(1024) NOT NULL DEFAULT 'No description',
  PRIMARY KEY (`ID`),
  UNIQUE KEY `Nome_UNIQUE` (`Name`)
);
```

Schema DB 4/5

Registrations_calls table

```
CREATE TABLE `registrations_calls` (
  `ID_Student` int NOT NULL,
  `ID_Call` int NOT NULL,
  `Mark` enum('','Assente','Rimandato','Riprovato','18','19','20','21','22','23','24','25','26','27','28','29','30','30L') NOT NULL,
  `EvaluationStatus` enum('Non inserito','Inserito','Pubblicato','Rifiutato','Verbalizzato') NOT NULL,
  PRIMARY KEY (`ID_Student`,`ID_Call`),
  KEY `Registrations_calls->Students_idx` (`ID_Student`),
  KEY `Registrations_calls->Calls_idx` (`ID_Call`),
  CONSTRAINT `Registrations_calls->Calls` FOREIGN KEY (`ID_Call`) REFERENCES `calls` (`ID`) ON UPDATE CASCADE,
  CONSTRAINT `Registrations_calls->Students` FOREIGN KEY (`ID_Student`) REFERENCES `users` (`ID`) ON UPDATE CASCADE
);
```

Registrations_courses table

```
CREATE TABLE `registrations_courses` (
  `ID_Student` int NOT NULL,
  `ID_Course` int NOT NULL,
  PRIMARY KEY (`ID_Student`,`ID_Course`),
  KEY `Registrations_courses->Students_idx` (`ID_Student`),
  KEY `Registrations_courses->Courses_idx` (`ID_Course`),
  CONSTRAINT `Registrations_courses->Courses` FOREIGN KEY (`ID_Course`) REFERENCES `courses` (`ID`) ON UPDATE CASCADE,
  CONSTRAINT `Registrations_courses->Students` FOREIGN KEY (`ID_Student`) REFERENCES `users` (`ID`) ON UPDATE CASCADE
);
```

Schema DB

Students_verbals table

```
CREATE TABLE `students_verbals` (
  `ID_Student` int NOT NULL,
  `ID_Verbal` int NOT NULL,
  PRIMARY KEY (`ID_Student`,`ID_Verbal`),
  KEY `students_verbals->Students_idx` (`ID_Student`),
  KEY `students_verbals->Verbals_idx` (`ID_Verbal`),
  CONSTRAINT `students_verbals->Students` FOREIGN KEY (`ID_Student`) REFERENCES `users` (`ID`) ON UPDATE CASCADE,
  CONSTRAINT `students_verbals->verbals` FOREIGN KEY (`ID_Verbal`) REFERENCES `verbals` (`ID`) ON UPDATE CASCADE
);
```

Verbals table

```
CREATE TABLE `verbals` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `CreationDate` date NOT NULL,
  `CreationTime` time NOT NULL,
  `ID_Call` int NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `Verbals->Calls_idx` (`ID_Call`),
  CONSTRAINT `Verbals->Calls` FOREIGN KEY (`ID_Call`) REFERENCES `calls` (`ID`) ON UPDATE CASCADE
);
```

Users table

```
CREATE TABLE `users` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Surname` varchar(64) NOT NULL,
  `Name` varchar(64) NOT NULL,
  `Email` varchar(64) NOT NULL,
  `Username` varchar(64) NOT NULL,
  `Password` varchar(64) NOT NULL,
  `Role` varchar(64) NOT NULL,
  `ID_DegreeCourse` int DEFAULT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `Username_UNIQUE` (`Username`),
  KEY `Students->DegreeCourses_idx` (`ID_DegreeCourse`),
  CONSTRAINT `users->degreeCourses` FOREIGN KEY (`ID_DegreeCourse`) REFERENCES `degree_courses` (`ID`) ON UPDATE CASCADE
);
```

VERSIONE HTML PURA

Completamento delle specifiche

- Aggiunta di un bottone di logout, per permettere all'utente (sia docente che studente) di poter uscire dal sito
- Possono essere creati più verbali relativi allo stesso appello, ciascuno contenente studenti diversi e mai duplicati/condivisi. Si è scelto questo tipo di implementazione per garantire una maggiore flessibilità al docente, il quale in questo modo non è costretto a inserire e pubblicare tutti i voti di un appello prima di poterli verbalizzare.
- Gli studenti ,come osservabile dallo schema ER, sono associati sia ai corsi, che agli appelli. Questo permette a ciascuno studente di poter vedere solo i corsi e gli appelli a cui è iscritto.
- Aggiunta possibilità da parte del docente di riportare un voto dallo stato «inserito» allo stato «non inserito» andando a modificare il singolo voto con il valore <vuoto>
- Aggiunto un bottone di «Go back» per ritornare alla pagina precedente.

Analisi requisiti applicazione 1/2

Un'applicazione permette di verbalizzare gli esiti degli esami di un appello.

Il docente accede tramite login e seleziona nella HOME page un corso da una lista dei propri corsi ordinata in modo alfabetico decrescente e poi una data d'appello del corso scelto selezionata da un elenco ordinato per data decrescente. Ogni corso ha un solo docente.

La selezione dell'appello porta a una pagina ISCRITTI, che mostra una tabella con tutti gli iscritti all'appello. La tabella riporta i seguenti dati: matricola, cognome e nome, email, corso di laurea, voto e stato di valutazione. Il voto può non essere ancora definito. Lo stato di valutazione dello studente rispetto all'appello può assumere i valori: non inserito, inserito, pubblicato, rifiutato e verbalizzato.

Selezionando un'etichetta nell'intestazione della tabella, l'utente ordina le righe in base al valore di tale etichetta (ad esempio, selezionando "cognome" la tabella è riordinata in base al cognome). Successive selezioni della stessa etichetta invertono l'ordinamento: si parte con l'ordinamento crescente. Il valore del voto viene considerato ordinato nel modo seguente: <vuoto>, assente, rimandato, riprovato, 18, 19, …, 30, 30 e lode.

Nella tabella della pagina ISCRITTI ad ogni riga corrisponde un bottone "MODIFICA". Premendo il bottone compare una pagina con una form che mostra tutti i dati dello studente selezionato e un campo di input in cui è possibile scegliere il voto. L'invio della form provoca la modifica o l'inserimento del voto. Inizialmente le righe sono nello stato di valutazione "non inserito". L'inserimento e le successive eventuali modifiche portano la riga nello stato di valutazione "inserito".

Pages (views), View components, Events, actions

Analisi requisiti applicazione 2/2

Alla tabella della pagina ISCRITTI è associato un bottone PUBBLICA che comporta la pubblicazione delle righe con lo stato di valutazione INSERITO. La pubblicazione rende il voto non più modificabile dal docente e visibile allo studente e cambia lo stato di valutazione della riga dello studente a “pubblicato”.

Lo studente accede tramite login e seleziona nella HOME page un corso tra quelli a cui è iscritto mediante una lista ordinata in modo alfabetico decrescente e poi una data d’appello del corso scelto selezionata da un elenco ordinato per data decrescente.

Uno studente può essere iscritto a più appelli dello stesso corso. La selezione della data d’appello porta a una pagina ESITO che mostra il messaggio “Voto non ancora definito” se il docente non ha ancora pubblicato il risultato per quello studente in quell’appello. Altrimenti, la pagina mostra i dati dello studente, del corso, dell’appello e il voto assegnato.

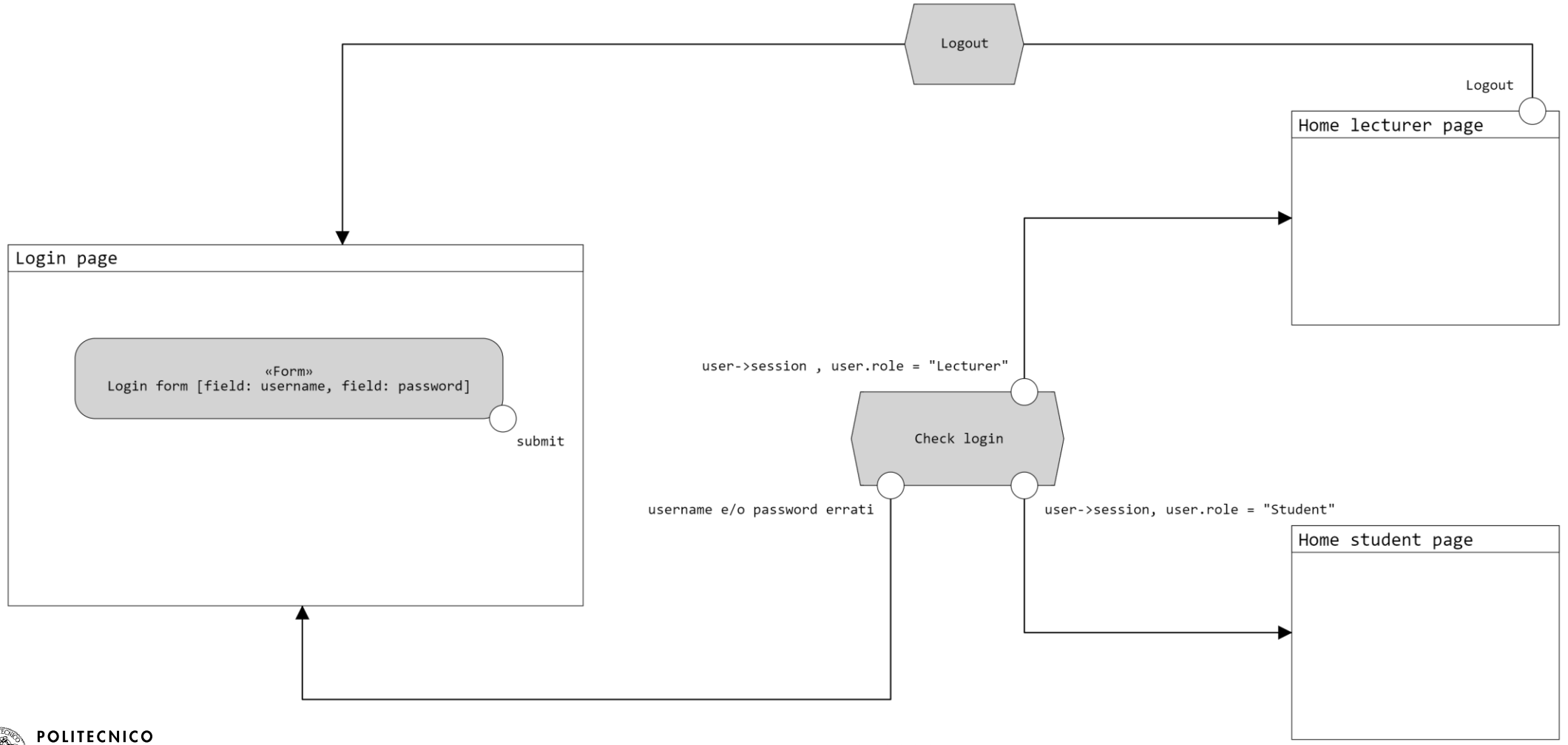
Se il voto è tra 18 e 30 e lode compare un bottone RIFIUTA. Premendo tale bottone la pagina mostra gli stessi dati con la dizione aggiunta “Il voto è stato rifiutato” e senza il bottone RIFIUTA. Il rifiuto del voto cambia lo stato di valutazione a “rifiutato” della riga dello studente per quell’appello nella pagina ISCRITTI del docente.

Nella pagina ISCRITTI del docente la tabella degli iscritti è associata anche a un bottone VERBALIZZA. La pressione del bottone provoca il cambio di stato a “verbalizzato” per le righe nello stato “pubblicato” o "rifiutato" e comporta anche la creazione di un verbale e la disabilitazione della possibilità di rifiutare il voto.

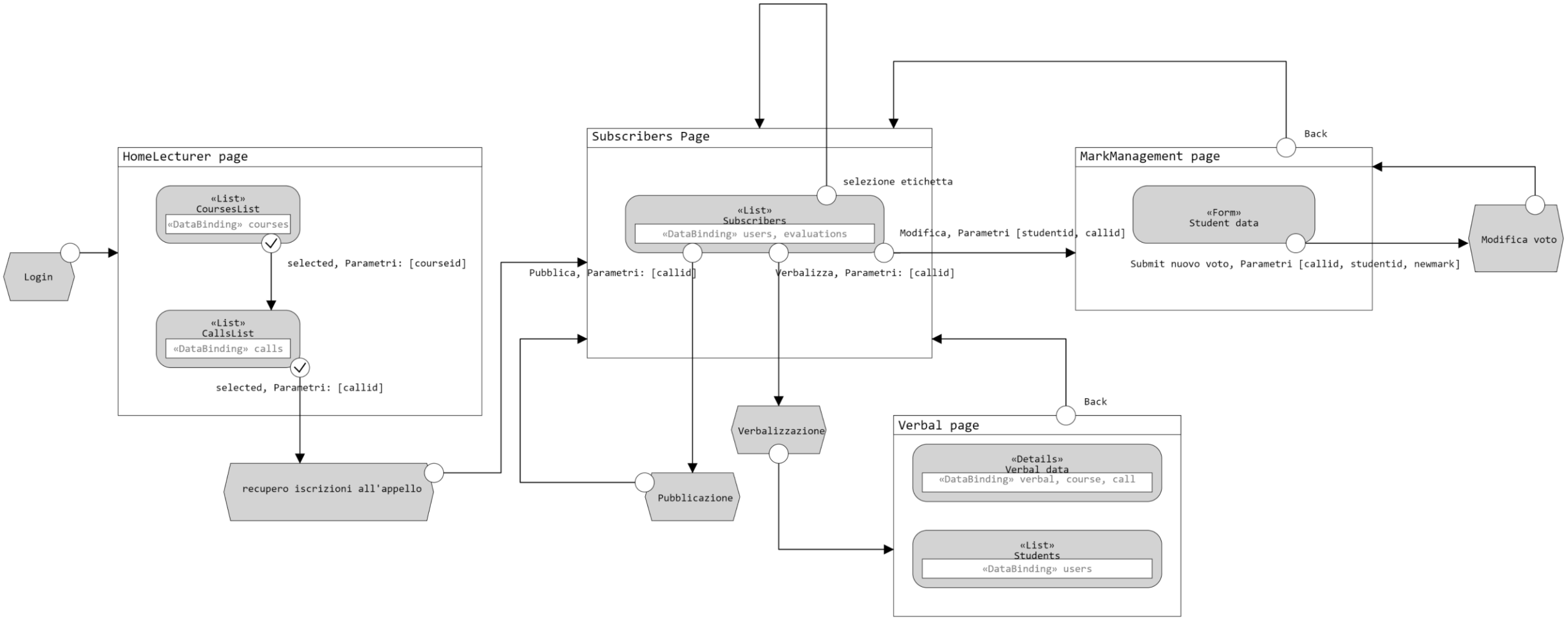
Il rifiuto implica la verbalizzazione di “rimandato” come voto. Un verbale ha un codice generato dal sistema, una data e ora di creazione ed è associato all’appello del corso a cui si riferisce e agli studenti (con nome, cognome, matricola e voto) che passano allo stato “verbalizzato”. A seguito della pressione del bottone VERBALIZZA compare una pagina VERBALE che mostra i dati completi del verbale creato.

Pages (views), View components, Events, actions

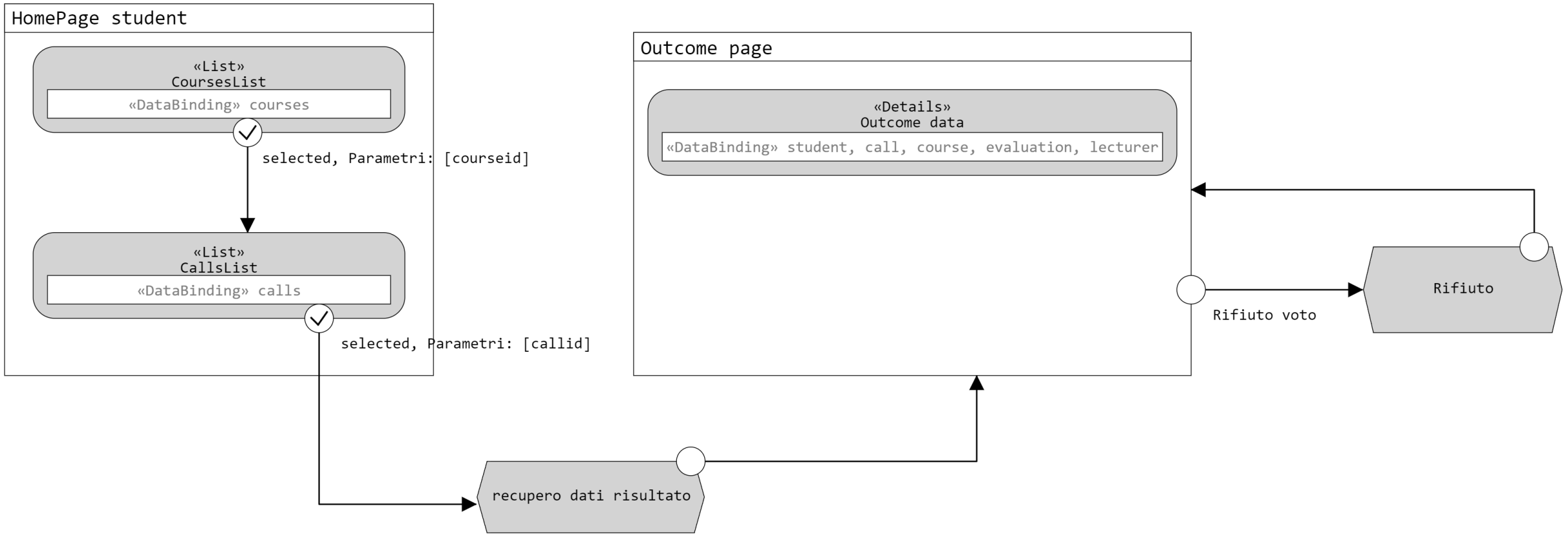
Design applicativo (IFML)



Design applicativo (IFML)



Design applicativo (IFML)



Componenti 1/3

- **Model objects (Beans)**
  - CallEvaluation
  - Course
  - DegreeCourse
  - GraduationCall
  - User
  - Verbal

- **Controllers (Servlet)**
  - CheckLogin
  - GetSubscriptionToCall
  - GoToHomeLecturer
  - GoToHomeStudent
  - GoToMarkManagement
  - GoToOutcome
  - GoToVerbalRecap
  - Logout
  - PublishStudentsMarks
  - RefuseMark
  - UpdateStudentMark
  - VerbalizeStudentsMarks

- **Data Access Object (Classes)**
  - ***CallEvaluationDAO***
    - findAllEvaluationByStudentId(int)
    - findAllEvaluationByCallId(int)
    - findEvaluationByCallAndStudentId(int, int)
    - updateEvaluationStateByStudentAndCallId(int, int, String)
    - verbalizeAllMarksByCallId(date, time, int)
    - publishAllMarksByCallId(int)
    - updateMarkByStudentAndCallId(int, int, String)
    - checkIfAnyMarkIsVerbalizable(int)
    - checkIfAnyMarkIsPublishable(int)
    - checkIfFormatIsCorrect(String)
    - getNumberOfVerbalizableMarks(int)
    - getNumberOfPublishableMarks(int)
    - checkIfStudentMarkIsRefusable(int, int)
    - checkIfStudentMarkIsUpdatable(int, int)
    - checkIfOutcomeCanBeRequested(int, int)

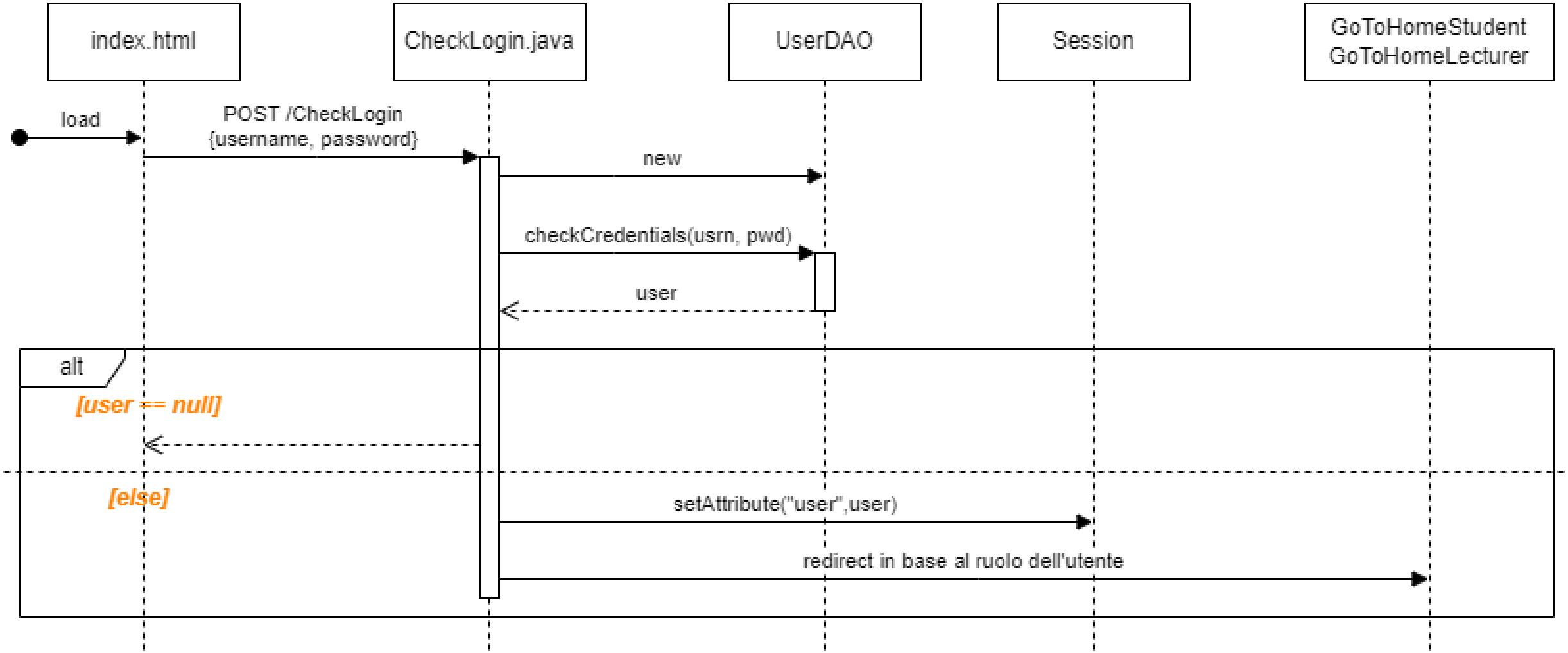
Componenti 2/3

- ***CourseDAO***
  - findAllCoursesByLecturer(int)
  - findAllCoursesByStudentId(int)
  - findCourseById(int)
  - checkIfCourseIsTaughtByLecturer(int, int)
- ***DegreeCourseDAO***
  - findAllDegreeCourses()
  - findDegreeCourseById(int)
- ***GraduationCallDAO***
  - findAllDegreeCallByCourseId(int)
  - findAllDegreeCallWhichStudentSubscribedToByCourseId(int, int)
  - findAllDegreeCallByDate(date)
  - createGraduationCall(date, time, int)
  - findGraduationCallById(int)
  - checkIfCourseOfCallIsTaughtByLecturer(int, int)
- ***LecturerDAO***
  - insertLecturer(String, String, String, String, String)
  - findLecturerById(int)
- ***StudentDAO***
  - findAllStudentsByDegreeCourse(int)
  - insertStudent(String, String, String, String, String, int)
  - findStudentsInVerbal(int)
  - findAllRegistrationsToTheCall(int)
  - findAllRegistrationsToTheCall(int, String, String)
  - findStudentById(int)
  - findAllRegistrationsAndEvaluationToCall(int)
  - findAllRegistrationsAndEvaluationToCallOrdered(int, String, String)
  - checkIfStudentIsSubscribedToCourse(int, int)
  - checkIfStudentIsSubscribedToCall(int, int)
  - findAllStudentsInVerbalById(int)
- ***UserDAO***
  - checkCredentials(String, String)

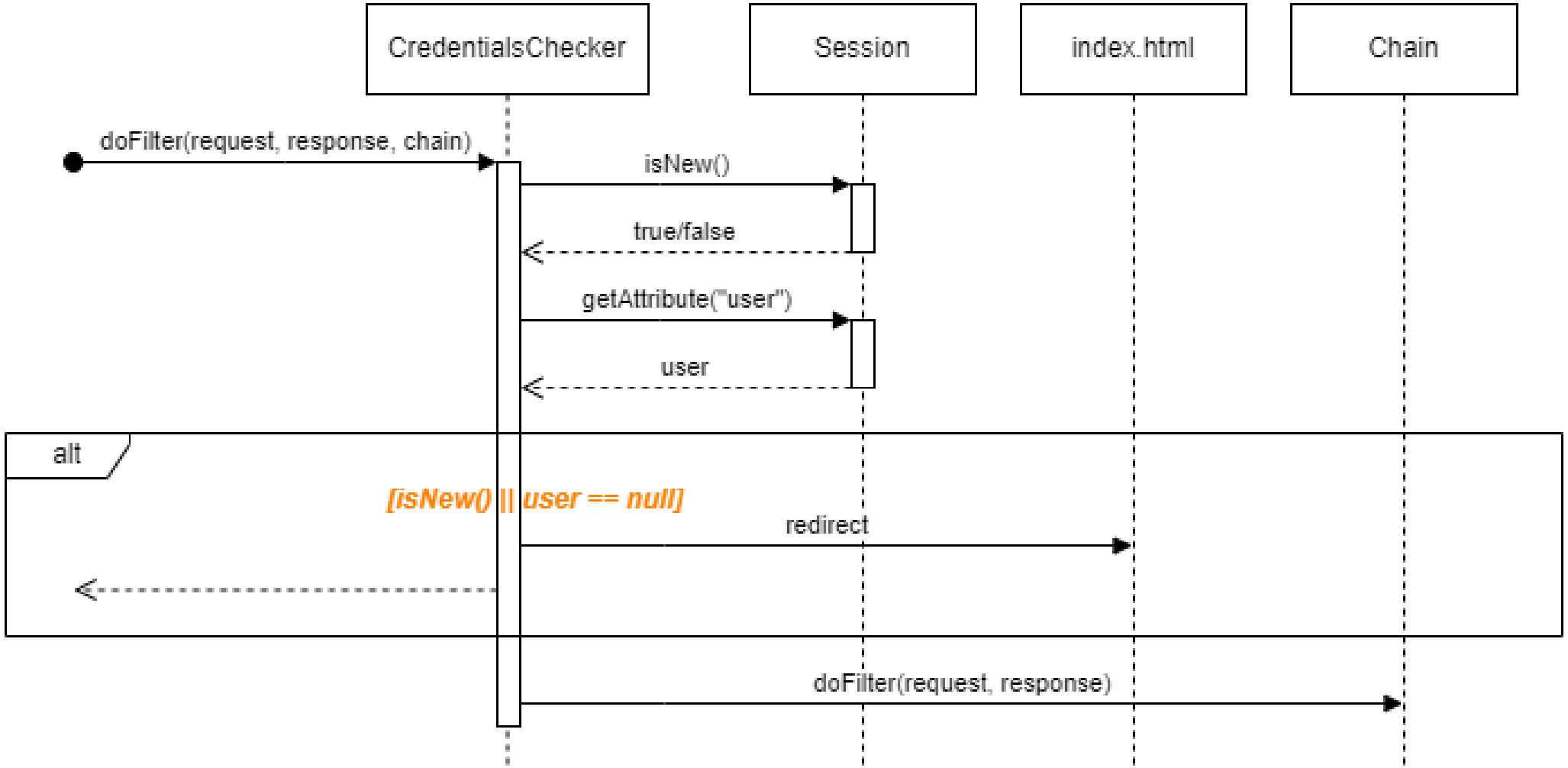
Componenti 3/3

- **Views (Template)**
  - Index.html
  - HomeLecturer.html
  - HomeStudent.html
  - MarkManagement.html
  - Outcome.html
  - Subscribers.html
  - Verbal.html
- **Exception (Classes)**
  - CallEvaluationDAOException
  - CourseDAOException
  - GraduationCallDAOException
  - StudentDAOException
- **Filters (Classes)**
  - CredentialsChecker
  - LecturerChcker
  - StudentChecker
  - NoCacher
- **Utils (Classes)**
  - ConnectionHandler
- **CSS**
  - *loginStyle.css* (Utilizzato per decorare la pagina di login
  - *mystyle.css* (Utilizzato per delle decorazioni specifiche differenti da quelle fornite da bootstrap)
  - *Bootstrap.min.css* (Utilizzato per decorare tutte le pagine html presenti nel progetto)

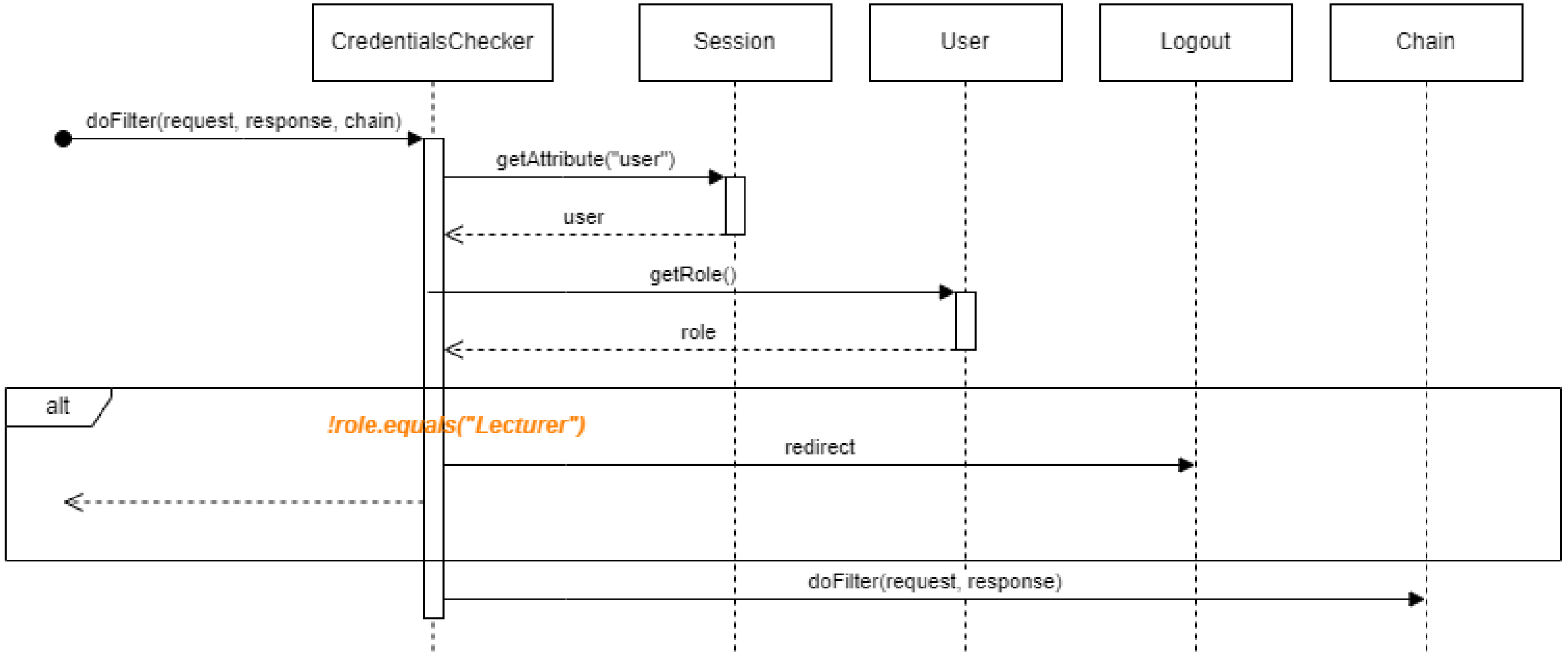
Sequence diagrams - Login



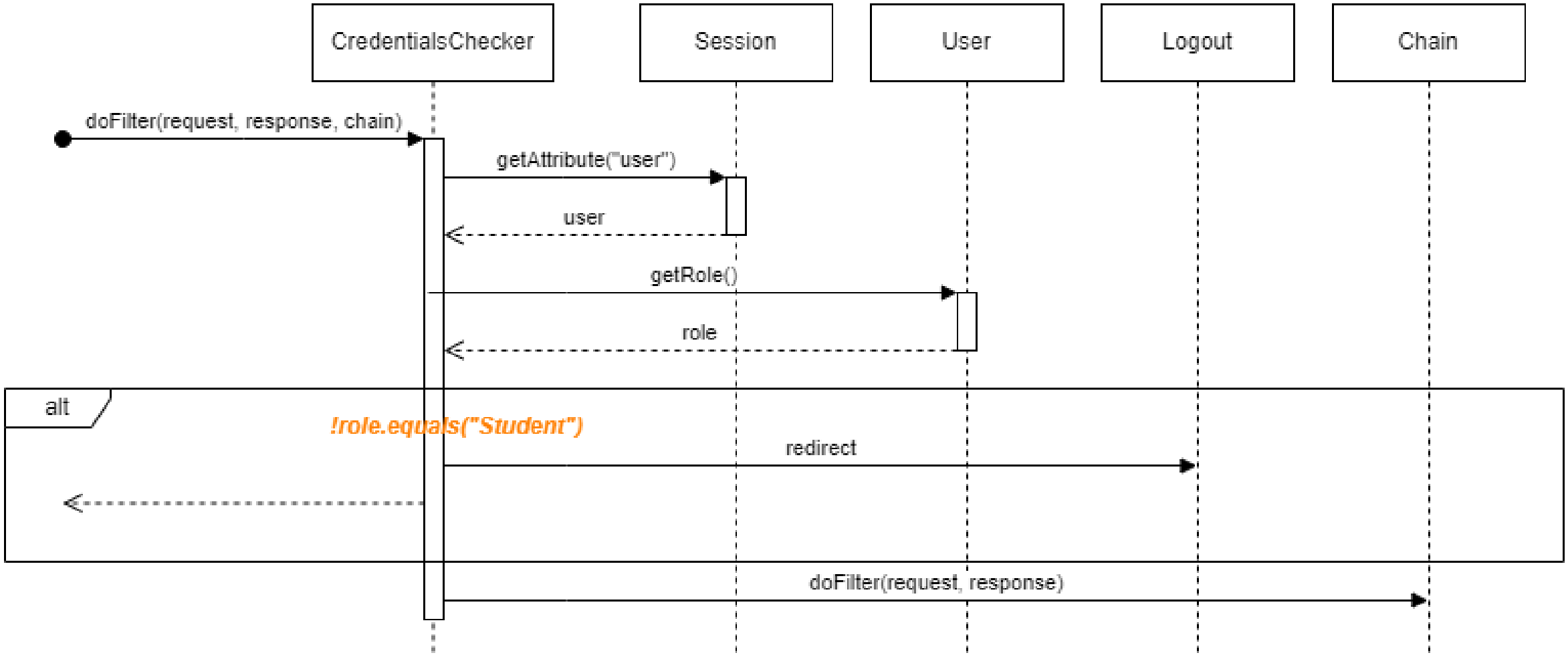
Sequence diagrams – Login filter



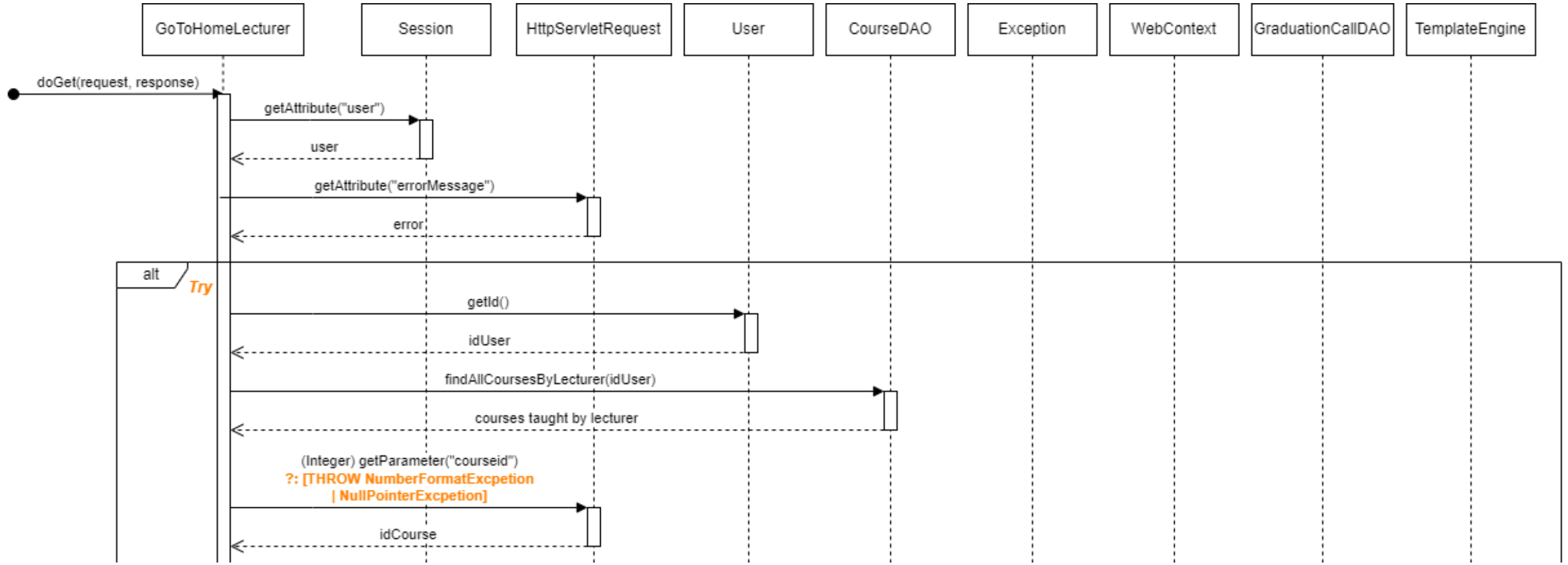
Sequence diagrams – Role lecturer filter



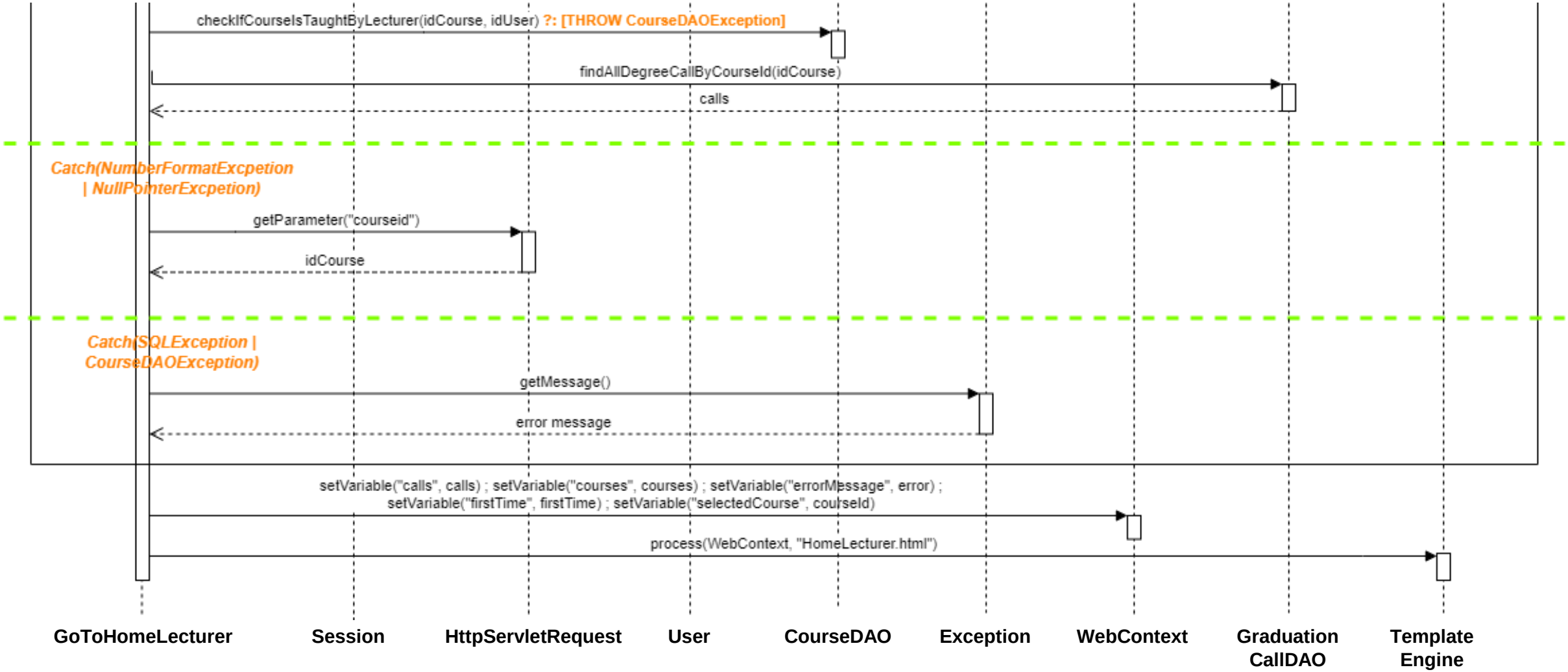
Sequence diagrams – Role student filter



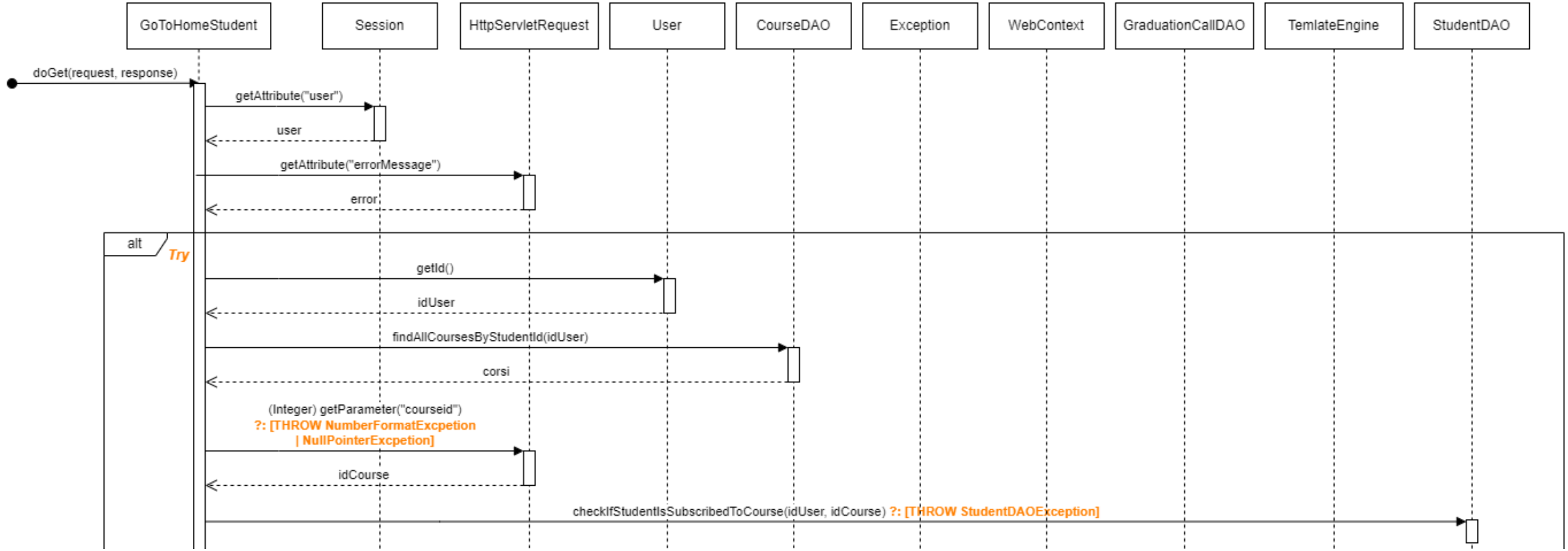
Sequence diagrams – GoToHomeLecturer



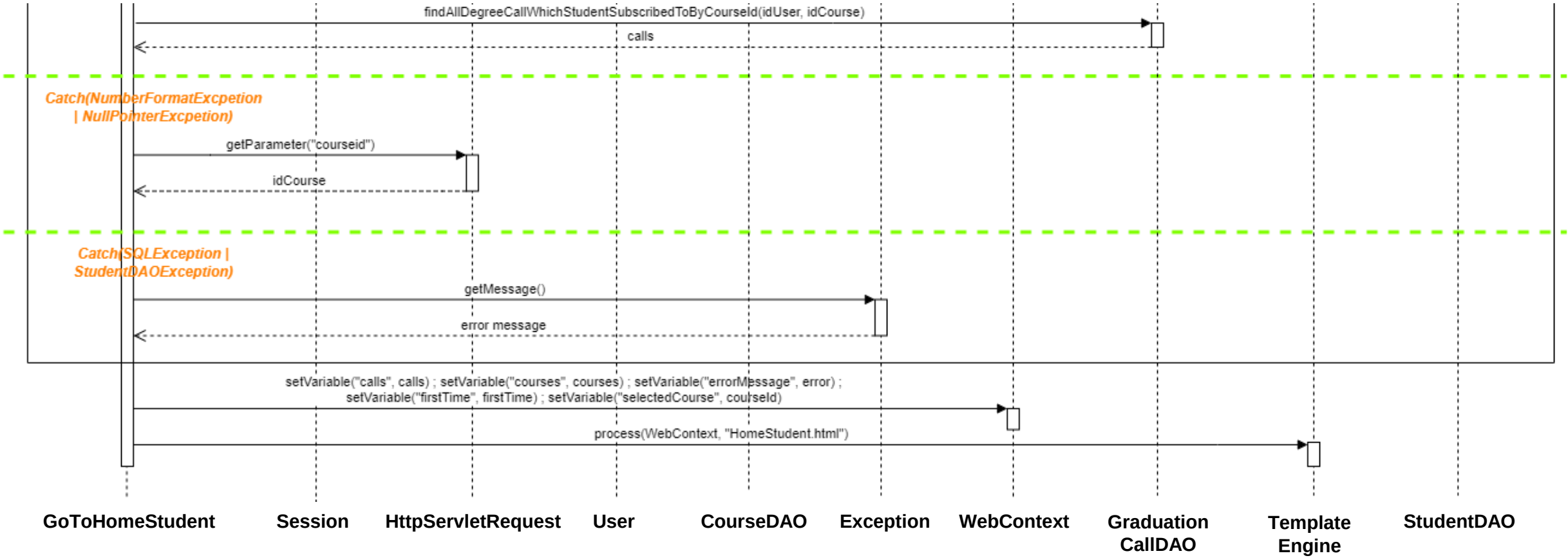
Sequence diagrams – GoToHomeLecturer



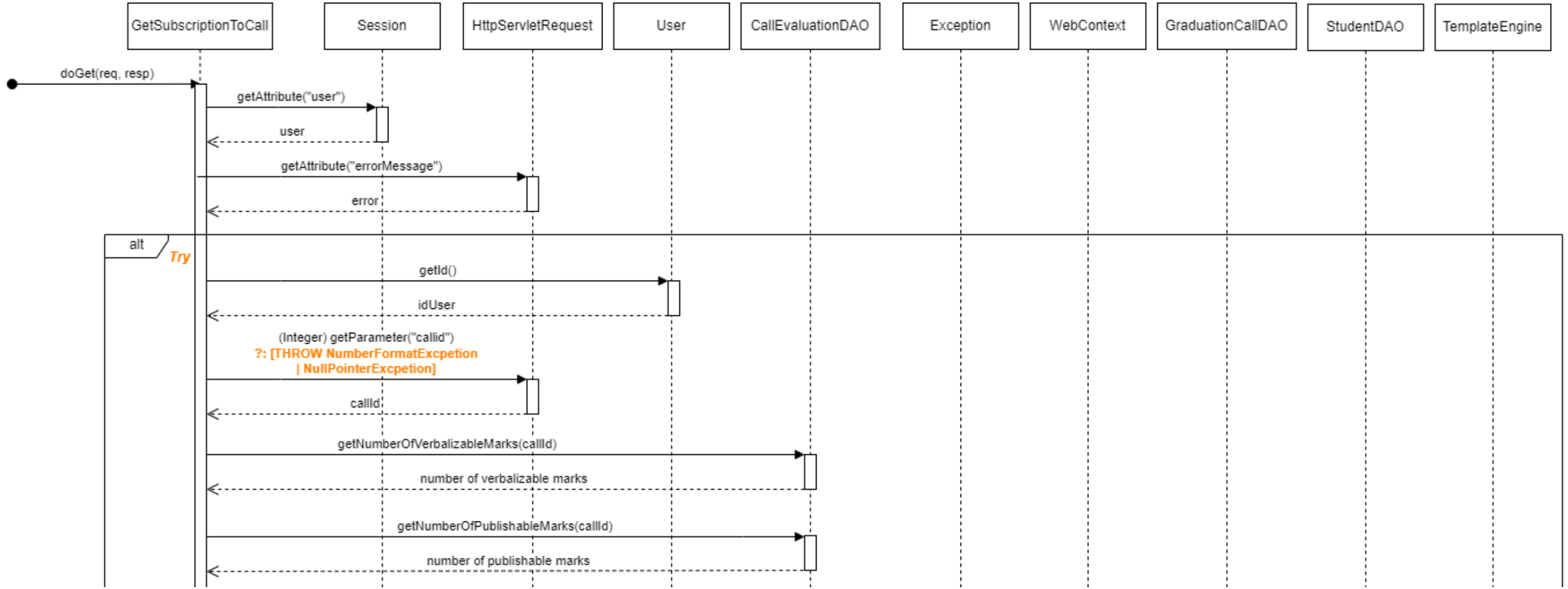
Sequence diagrams – GoToHomeStudent



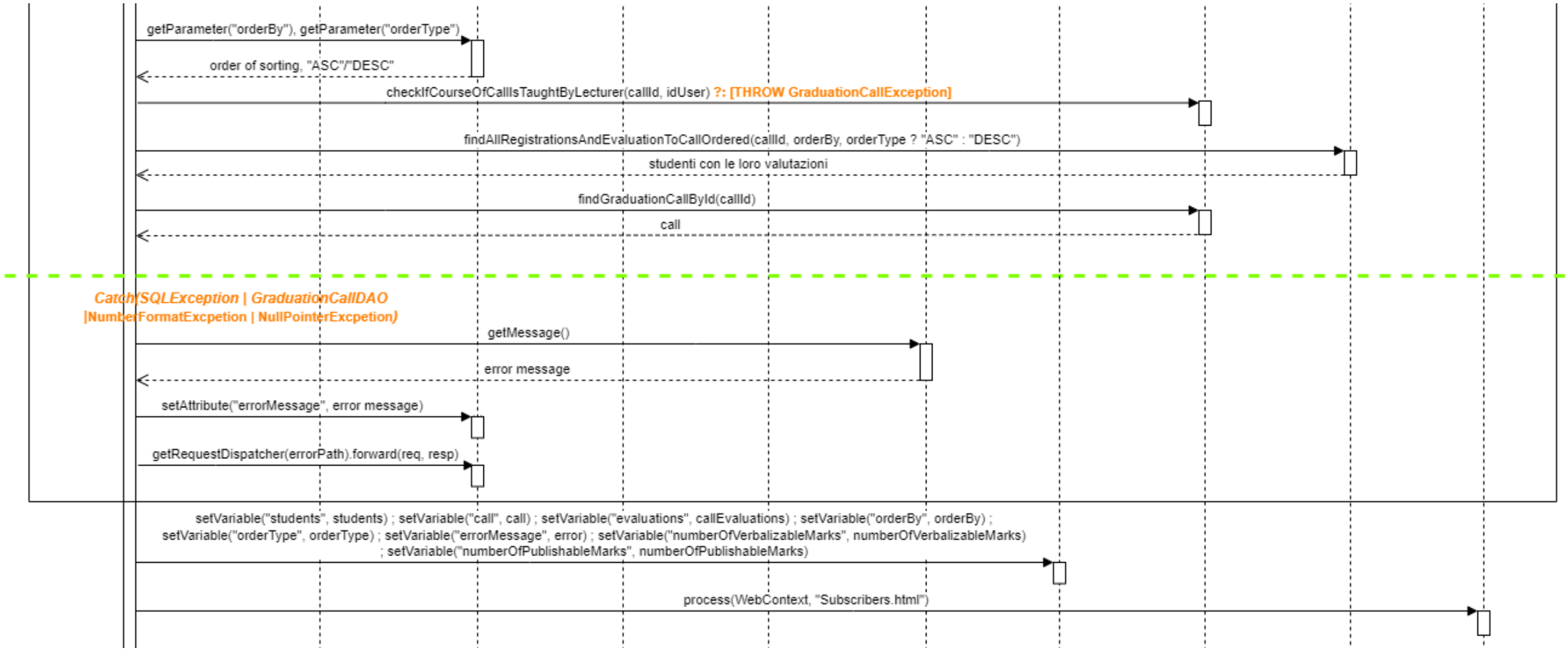
Sequence diagrams – GoToHomeStudent



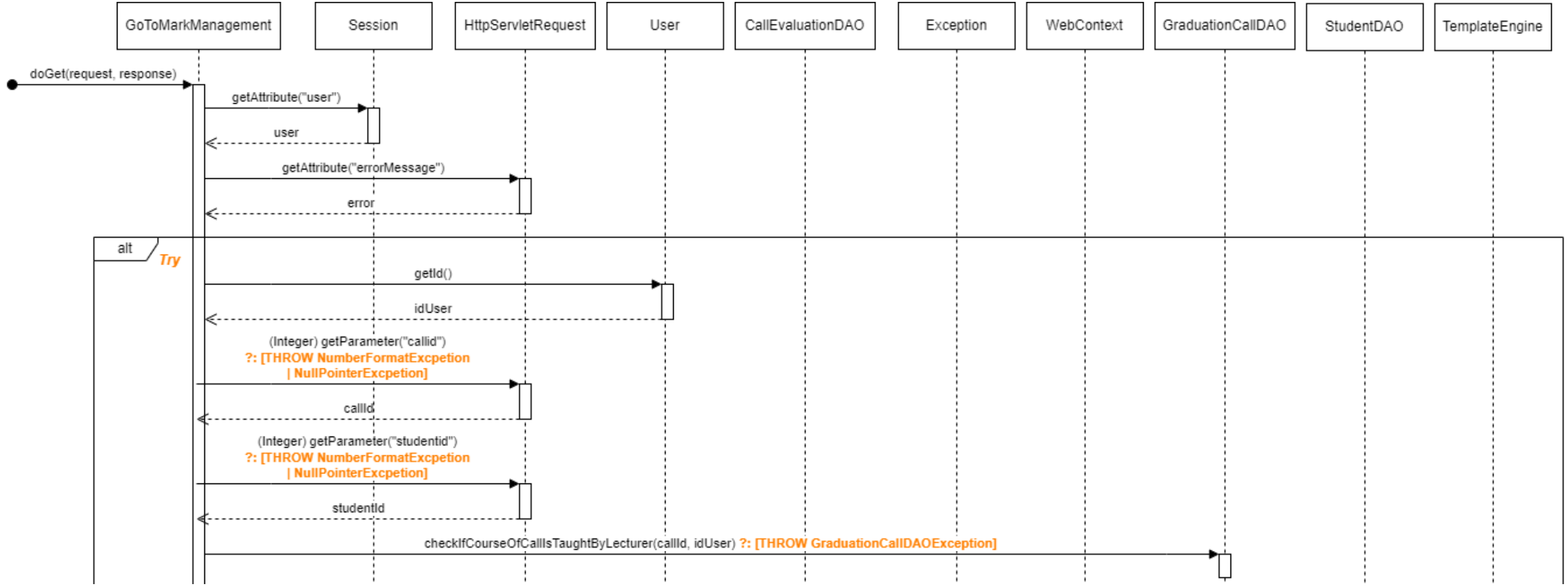
Sequence diagrams – GetSubscriptionToCall



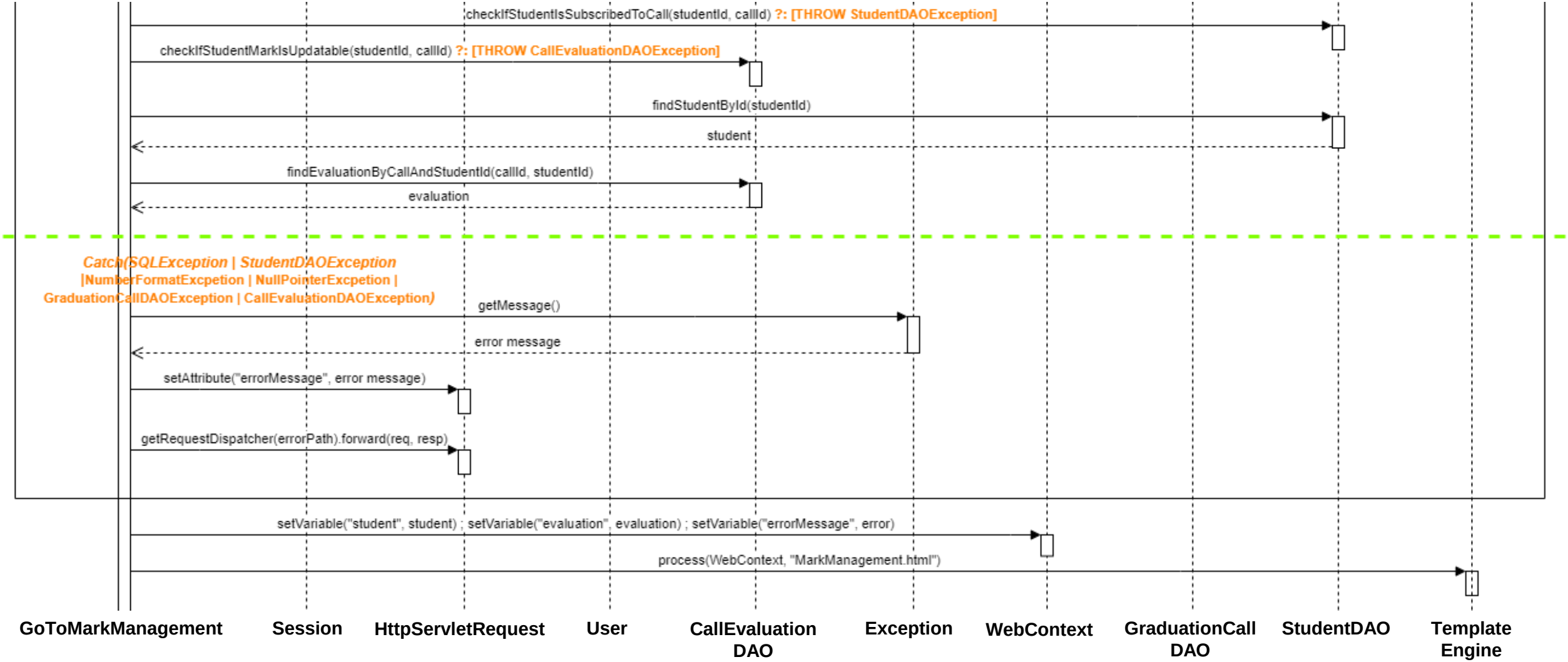
Sequence diagrams – GetSubscriptionToCall



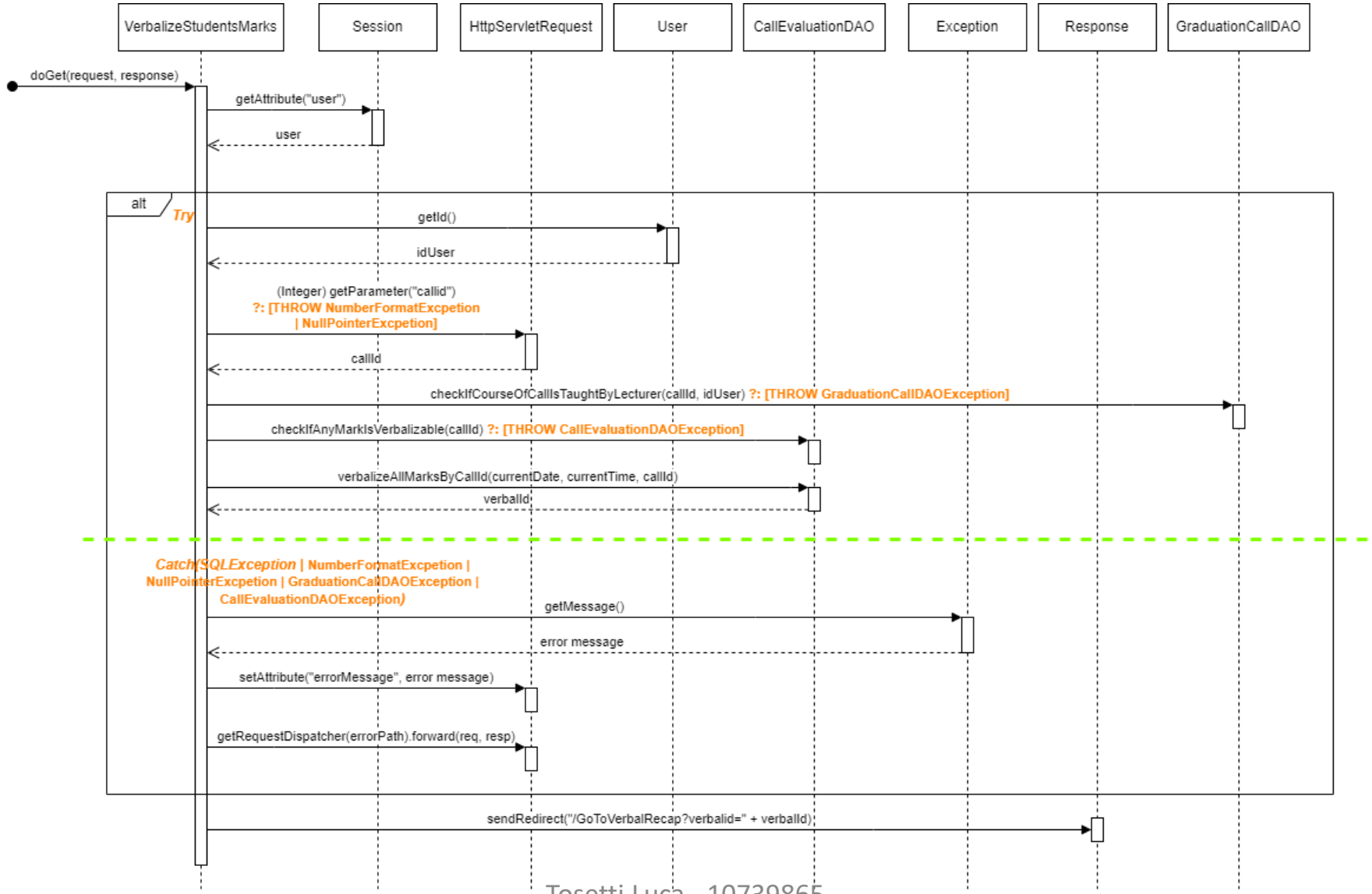
Sequence diagrams – GoToMarkManagement



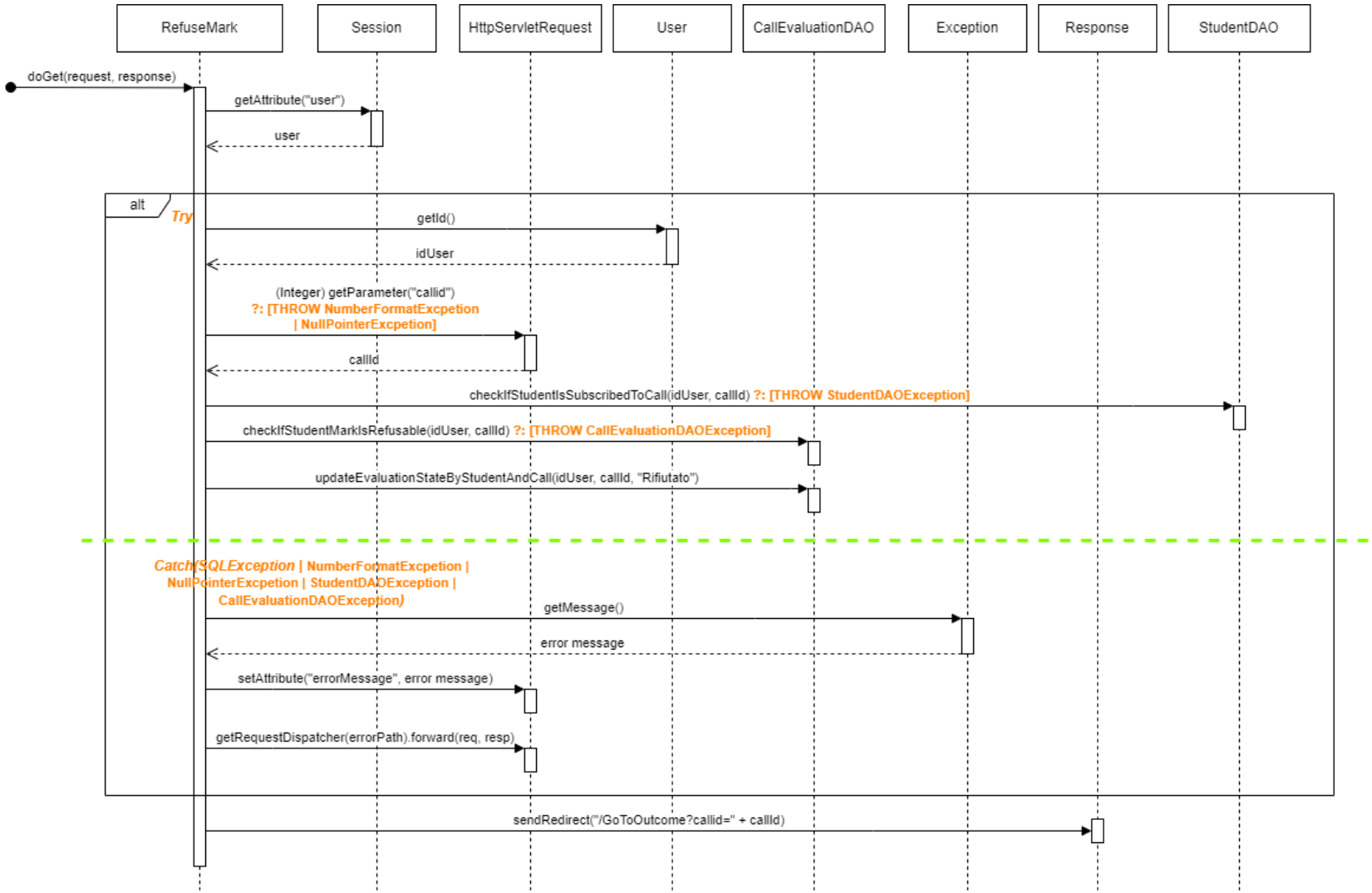
Sequence diagrams – GoToMarkManagement



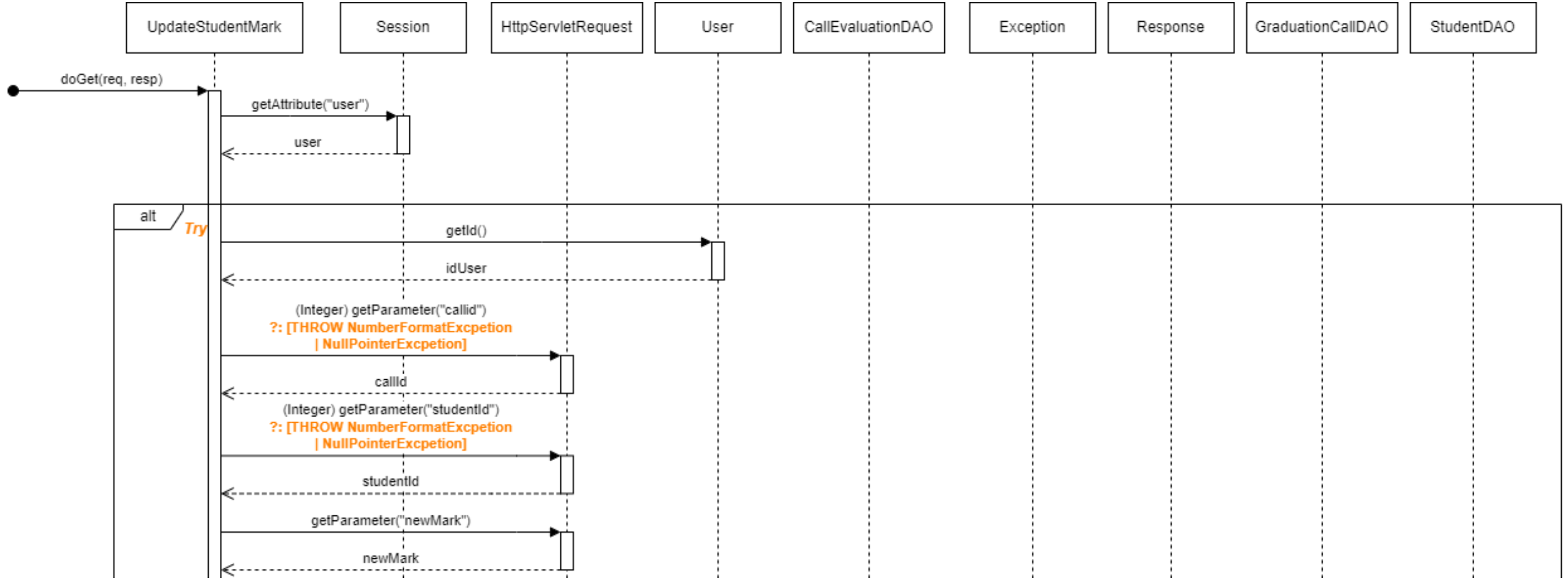
Sequence diagrams – VerbalizeStudentsMarks



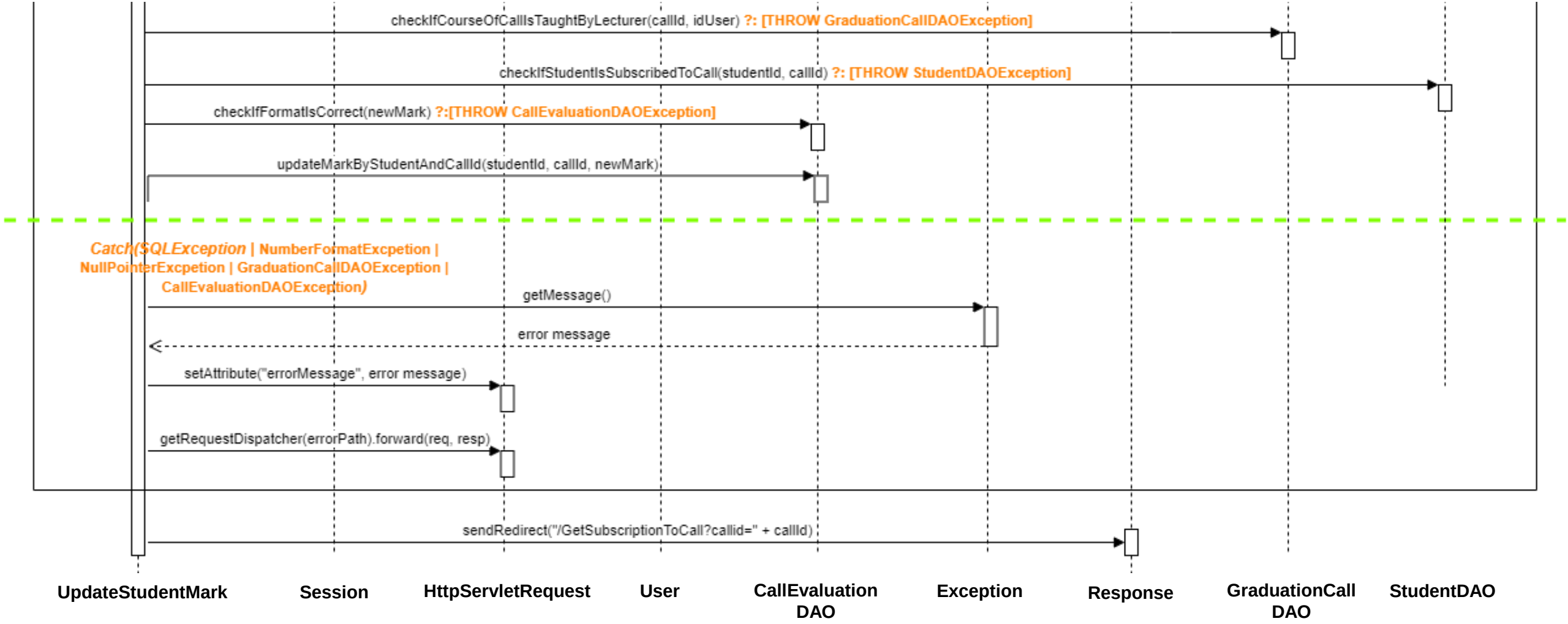
Sequence diagrams – RefuseMark



Sequence diagrams – UpdateStudentMark



Sequence diagrams – UpdateStudentMark



VERSIONE JAVASCRIPT

Traccia

Si realizzi un'applicazione client server web che modifica le specifiche precedenti come segue:

- Dopo il login dell'utente, l'intera applicazione è realizzata con un'unica pagina per il docente e un'unica pagina per lo studente.
- Ogni interazione dell'utente è gestita senza ricaricare completamente la pagina, ma produce l'invocazione asincrona del server e l'eventuale modifica del contenuto da aggiornare a seguito dell'evento.
- La funzione di riordino della tabella degli iscritti è realizzata a lato client.
- Alla tabella degli iscritti è associato un bottone INSERIMENTO MULTIPLO che provoca la comparsa di una pagina modale con tutte e sole le righe nello stato "non inserito" associate a un campo di input. Il docente può inserire un voto per un insieme delle righe e premere un bottone INVIA che comporta l'invio al server dei voti, il cambio di stato delle righe coinvolte, la chiusura della finestra modale e l'aggiornamento della tabella degli iscritti.

Completamento delle specifiche

- Stesse aggiunte specificate per la versione HTML pura (tranne il bottone di «Go back»)
- Implementata la visualizzazione della pagina di ricapitolazione dei dati di un verbale appena creato, attraverso una pagina modale

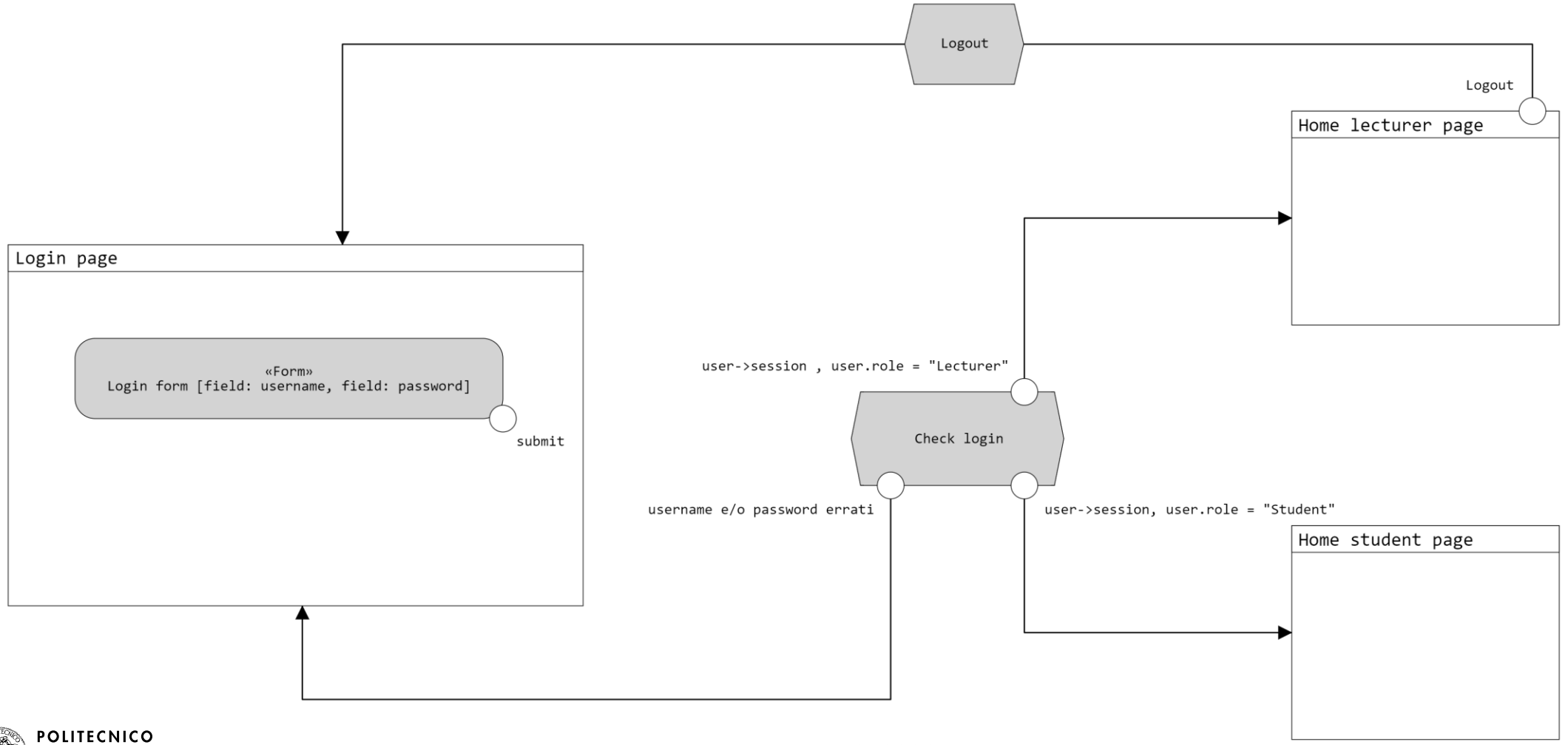
Analisi requisiti applicazione

Si realizzi un'applicazione client server web che modifica le specifiche precedenti come segue:

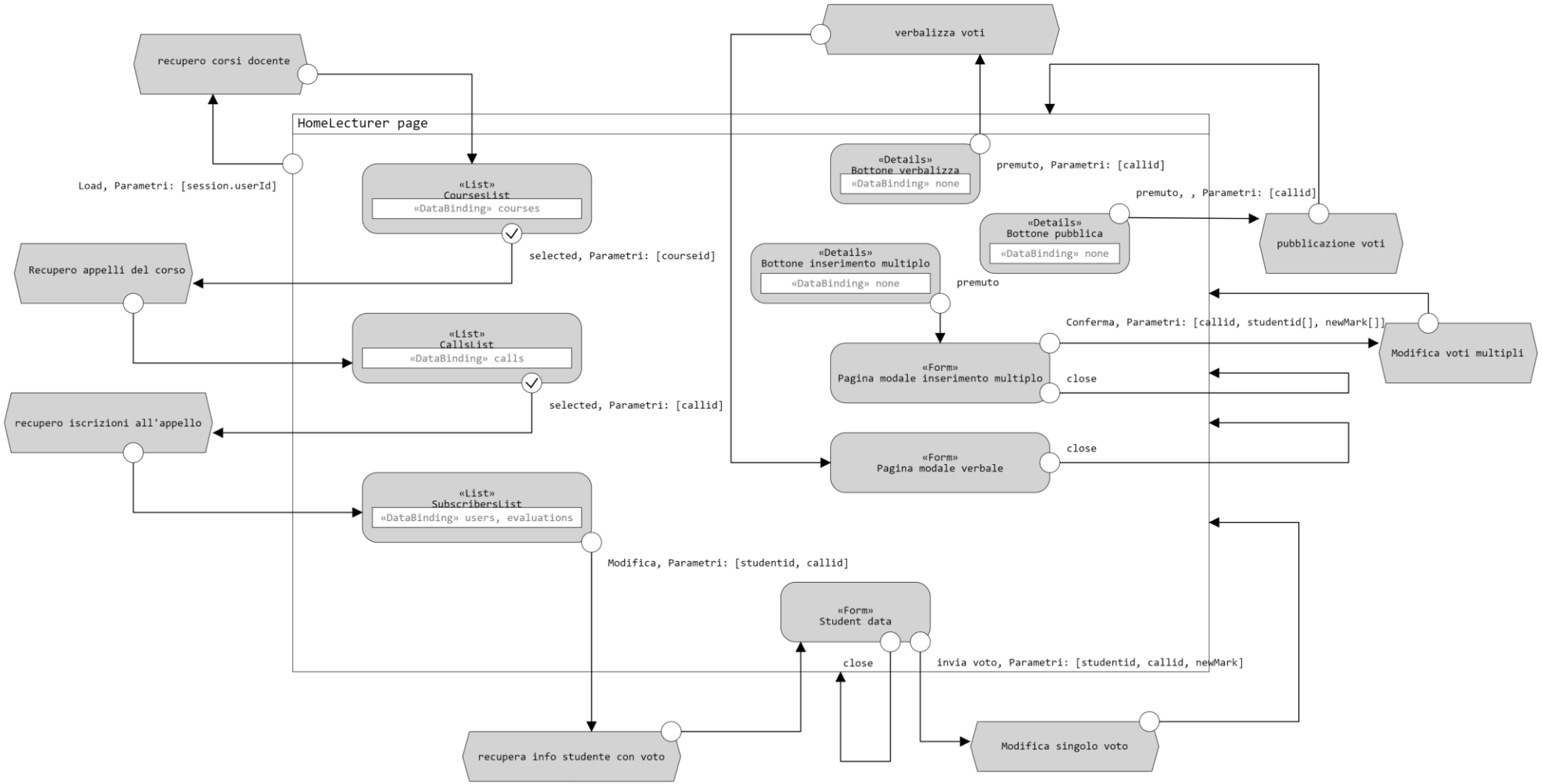
- Dopo il login dell'utente, l'intera applicazione è realizzata con un'unica pagina per il docente e un'unica pagina per lo studente.
- Ogni interazione dell'utente è gestita senza ricaricare completamente la pagina, ma produce l'invocazione asincrona del server e l'eventuale modifica del contenuto da aggiornare a seguito dell'evento.
- La funzione di riordino della tabella degli iscritti è realizzata a lato client.
- Alla tabella degli iscritti è associato un bottone INSERIMENTO MULTIPLO che provoca la comparsa di una pagina modale con tutte e sole le righe nello stato "non inserito" associate a un campo di input. Il docente può inserire un voto per un insieme delle righe e premere un bottone INVIA che comporta l'invio al server dei voti, il cambio di stato delle righe coinvolte, la chiusura della finestra modale e l'aggiornamento della tabella degli iscritti.

Pages (views), View components, Events, actions

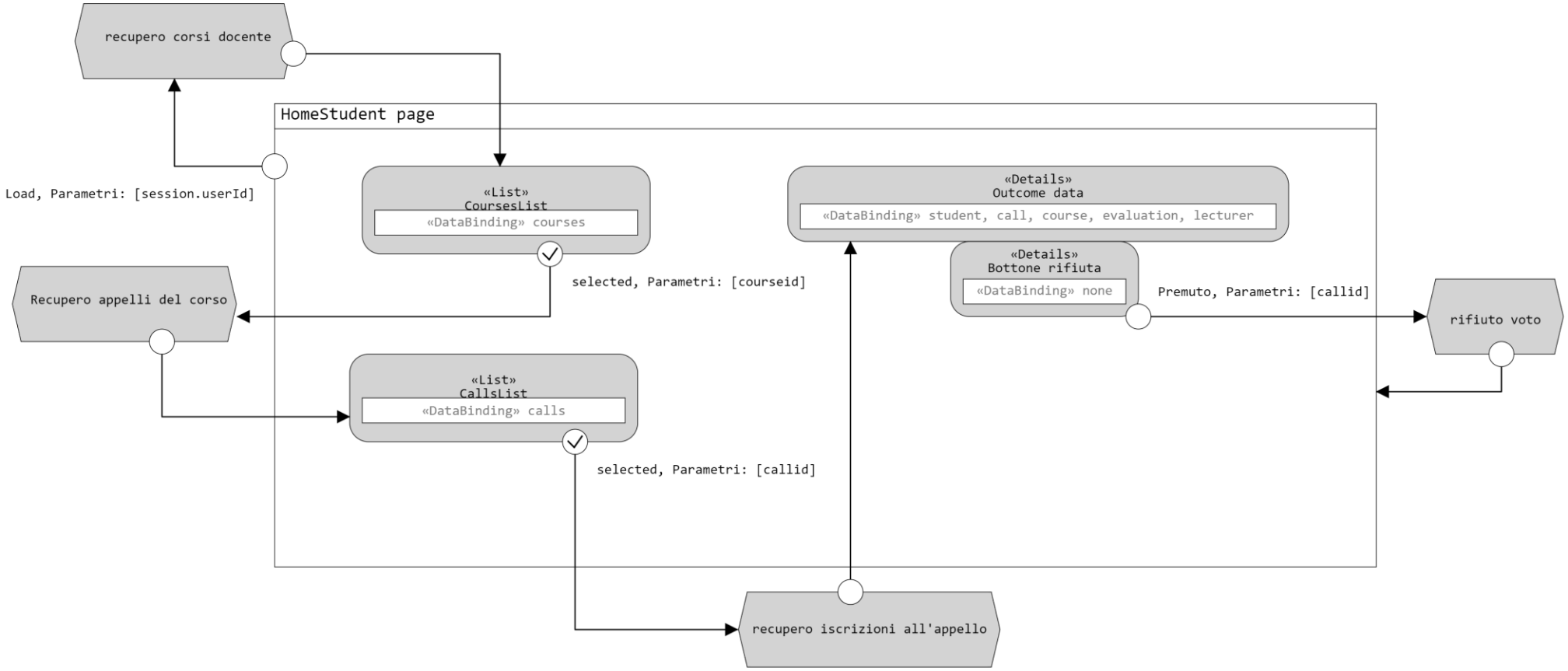
Design applicativo (IFML)



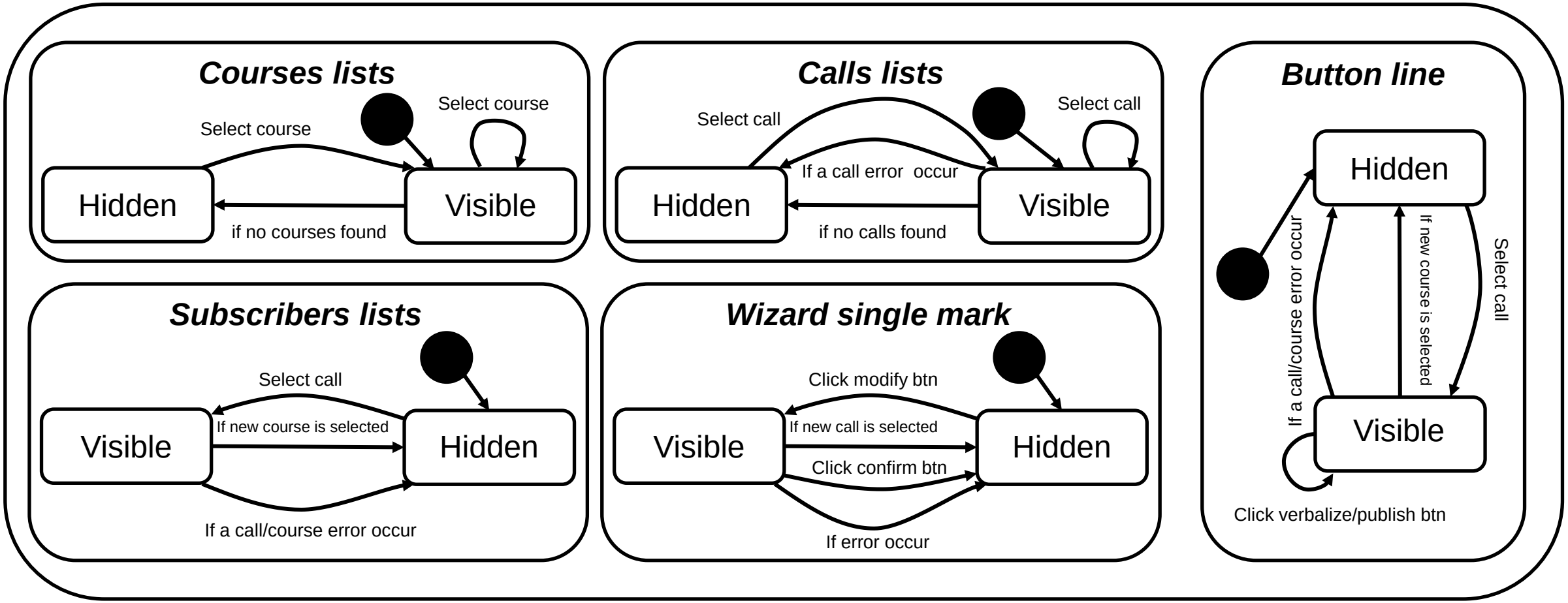
Design applicativo (IFML)



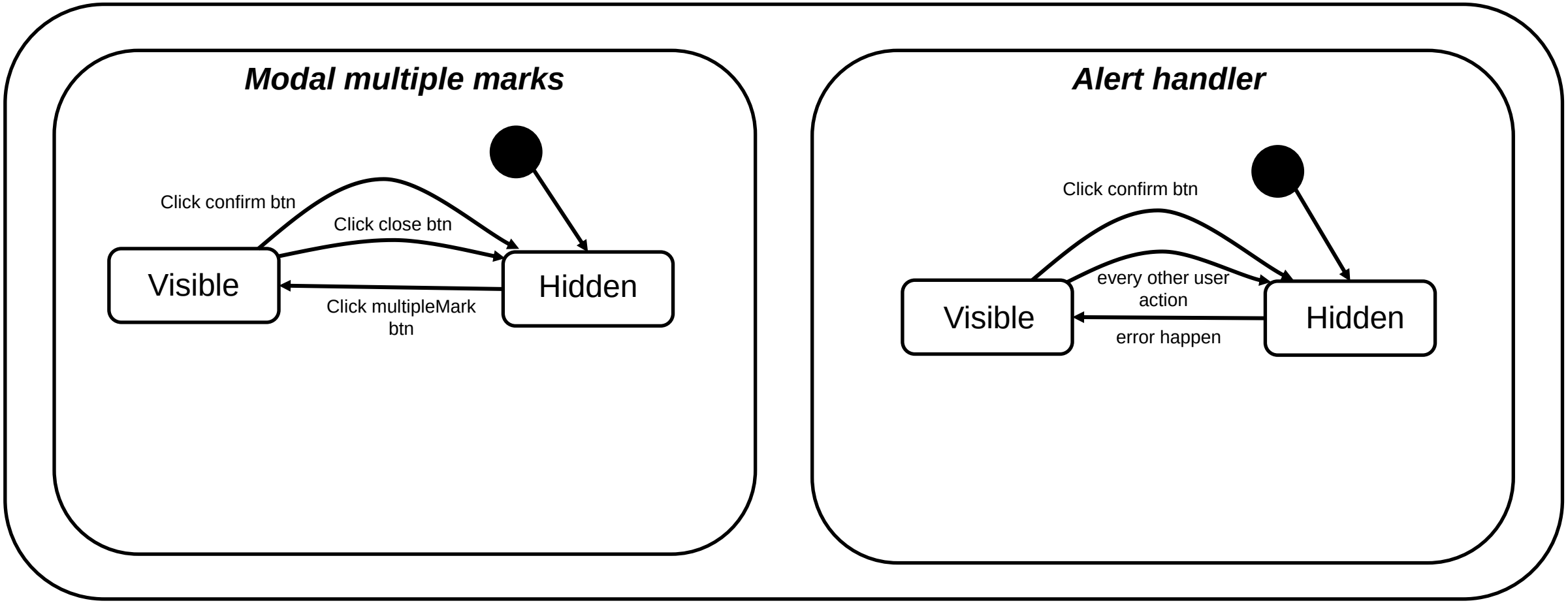
Design applicativo (IFML)



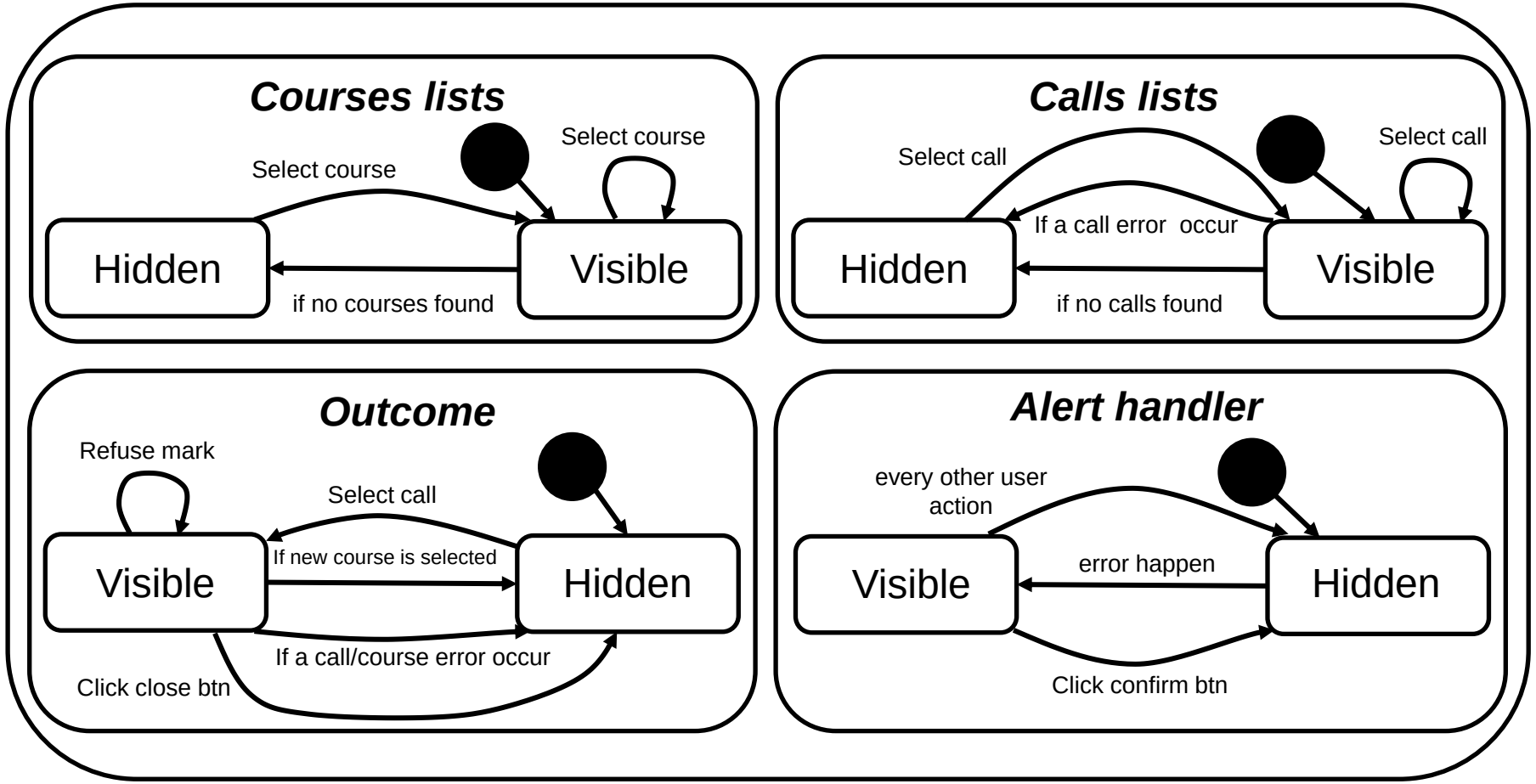
Stati - Docente 1/2



Stati - Docente 2/2



Stati - Studente



Componenti 1/2

- **Model objects (Beans)**
  - CallEvaluation
  - Course
  - DegreeCourse
  - GraduationCall
  - User
  - Verbal
- **Controllers (Servlet)**
  - CheckLogin
  - GetCoursesCalls
  - GetSubscriptionToCall
  - GetLecturersCourses
  - GetStudentCourses
  - GetMarkManagement
  - GetOutcome
  - GetVerbalData
  - Logout
  - PublishStudentsMarks
  - RefuseMark
  - UpdateMultipleMarks
  - UpdateStudentMark
  - VerbalizeStudentsMarks
- **Data Access Object (Classes)**
  - ***CallEvaluationDAO***
    - findAllEvaluationByStudentId(int)
    - findAllEvaluationByCallId(int)
    - findEvaluationByCallAndStudentId(int, int)
    - updateEvaluationStateByStudentAndCallId(int, int, String)
    - verbalizeAllMarksByCallId(date, time, int)
    - publishAllMarksByCallId(int)
    - updateMarkByStudentAndCallId(int, int, String)
    - checkIfAnyMarkIsVerbalizable(int)
    - checkIfAnyMarkIsPublishable(int)
    - checkIfFormatIsCorrect(String)
    - getNumberOfVerbalizableMarks(int)
    - getNumberOfPublishableMarks(int)
    - checkIfStudentMarkIsRefusable(int, int)
    - checkIfStudentMarkIsUpdatable(int, int)
    - checkIfOutcomeCanBeRequested(int, int)

Componenti 2/2

- **Views (Template)**
  - Index.html
  - HomeLecturer.html
  - HomeStudent.html
- **Filters (Classes)**
  - CredentialsChecker
  - LecturerChcker
  - StudentChecker
  - NoCacher
- **CSS**
  - *loginStyle.css* (Utilizzato per decorare la pagina di login
  - *mystyle.css* (Utilizzato per delle decorazioni specifiche differenti da quelle fornite da bootstrap)
  - *Bootstrap.min.css* (Utilizzato per decorare tutte le pagine html presenti nel progetto)
- **Exception (Classes)**
  - CallEvaluationDAOException
  - CourseDAOException
  - GraduationCallDAOException
  - StudentDAOException
- **Utils (Classes)**
  - ConnectionHandler
- **Javascript**
  - LecturerHomePageHandler.js
  - StudentHomePageHandler.js
  - Sort.js
  - LoginManagement.js
  - Utils.js

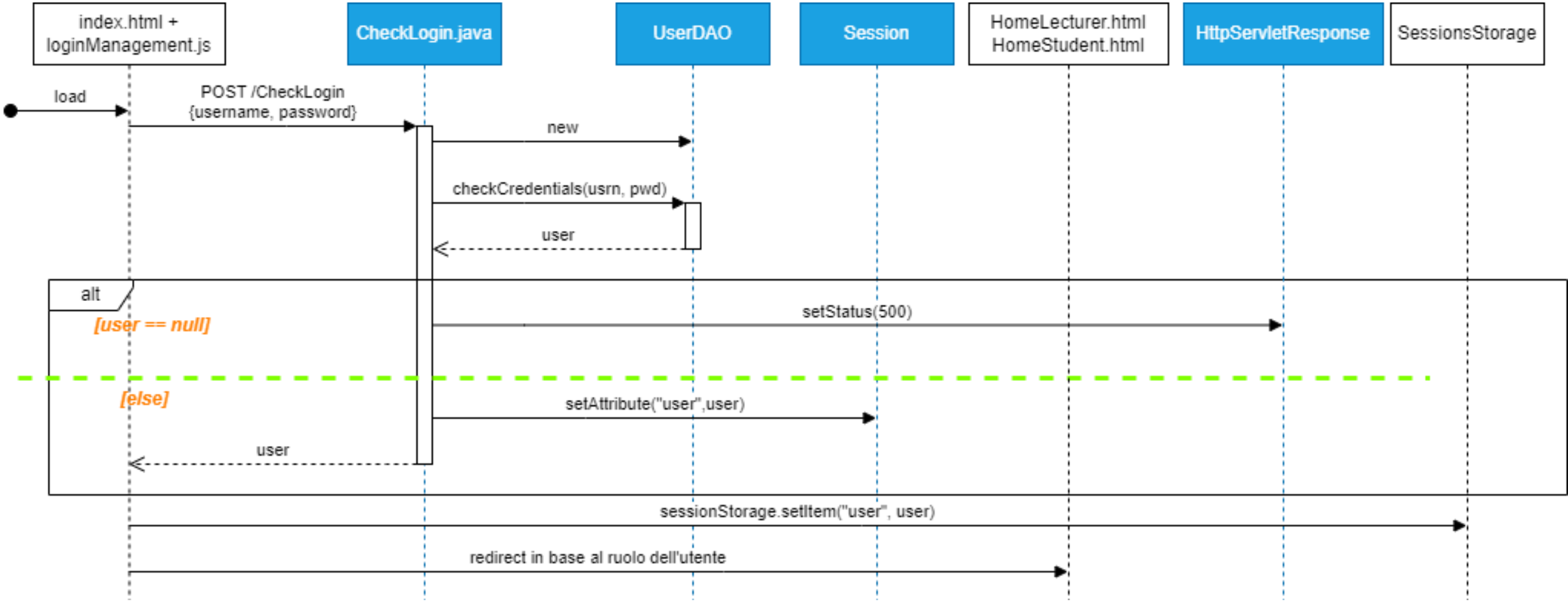
Event handler - Docente

Client side		Server side	
Evento	**Controllore**	**Evento**	**Controllore**
Index → login form → submit	Function makeCall	POST(username, password)	CheckLogin(servlet)
HomePage → load	Function PageOrchestrator…	GET(nessun parametro)	GetLecturerCourses(servlet)
HomePage → elenco corsi → selezione corso	Function CallsList.show(courseId, functionToShowDefault)	GET(id corso)	GetCoursesCalls(servlet)
HomePage → elenco appelli → selezione appello	Function SubscribersList.show(callId)	GET(id appello)	GetSubscriptionToCall(servlet)
HomePage → elenco iscritti → click link modifica	Function WizardSingleMark.show(studentId, callId)	GET(id studente, id appello)	GetMarkManagement(servlet)
HomePage → elenco iscritti → click bottone pubblica	Function MakeCall	POST(id appello)	PublishStudentsMarks(servlet)
HomePage → elenco iscritti → click bottone verbalizza	Function MakeCall	POST(id appello)	VerbalizeStudentsMarks(servlet)
HomePage → elenco iscritti → click bottone inserimento multiplo	Function ModalMultipleMark.show()	-	-
HomePage → elenco iscritti → wizard per voti singoli → click bottone modifica	Function MakeCall	POST(id studente, nuovo voto)	UpdateStudentMark(servlet)
HomePage → elenco iscritti → selezione appello	Function SubscribersList.show(callId) → ButtonLine.show(callId)	GET(id appello) x2	VerbalizeStudentsMarks(servlet) PublishStudentsMarks(servlet)
HomePage → ModalMultipleMarks → click bottone inserisci voti	Function MakeCall	POST(id studenti[], nuovi voti[], id appello)	UpdateMultipleMarks(servlet)
HomePage → elenco iscritti → click bottone verbalizza	Function MakeCall → ModalVerbalRecap.show(verbalId)	GET(id verbale)	GetVerbalData(servlet)
HomePage→elenco iscritti → click intestazione	Function sortTable(header.id)	-	-
Logout	-	GET	Logout(servlet)

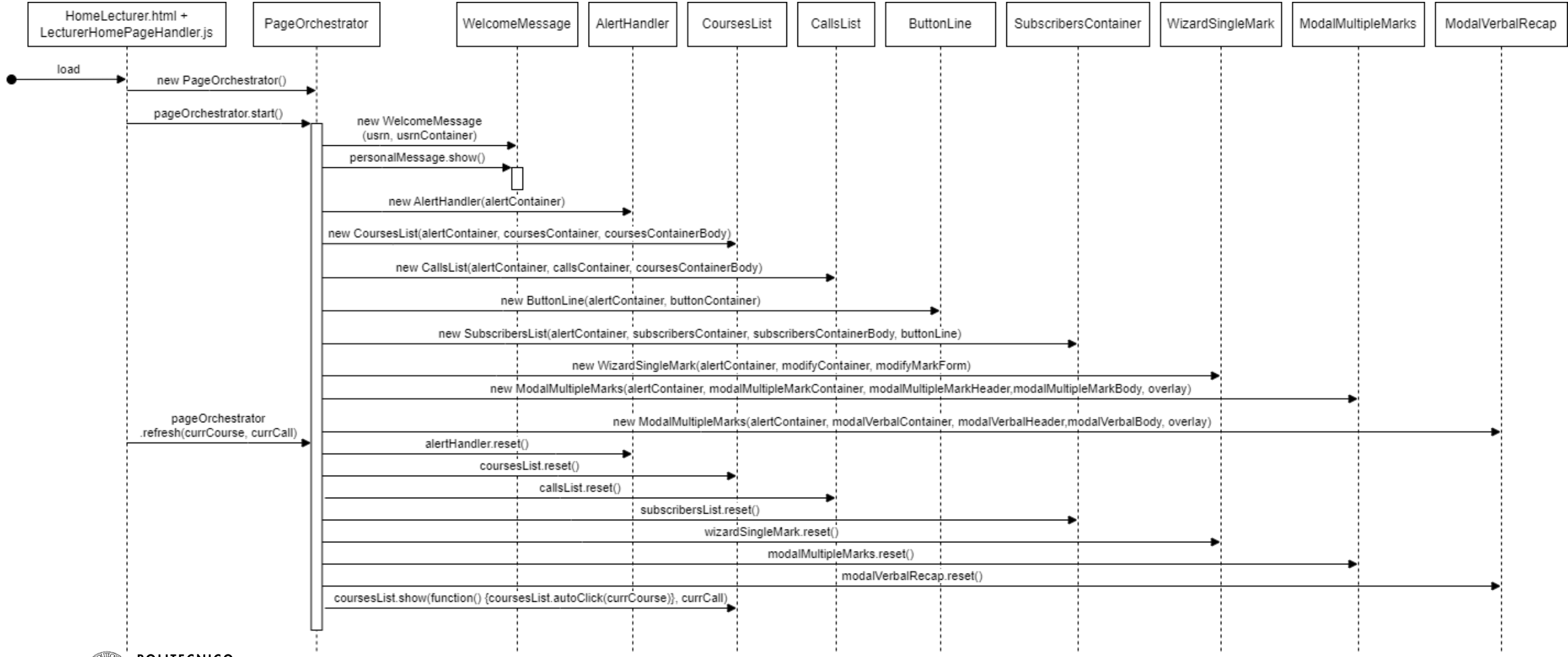
Event handler - Studente

Client side		Server side	
Evento	**Controllore**	**Evento**	**Controllore**
Index → login form → submit	Function makeCall	POST(username, password)	CheckLogin(servlet)
HomePage → load	Function PageOrchestrator…	GET(nessun parametro)	GetStudentCourses(servlet)
HomePage → elenco corsi → selezione corso	Function CallsList.show(courseId)	GET(id corso)	GetCoursesCalls(servlet)
HomePage → elenco appelli → selezione appello	Function Outcome.show(callId)	GET(id appello)	GetOutcome(servlet)
HomePage → risultato → click bottone rifiuto	Function MakeCall	POST(id appello)	RefuseMark(servlet)
Logout	-	GET	Logout(servlet)

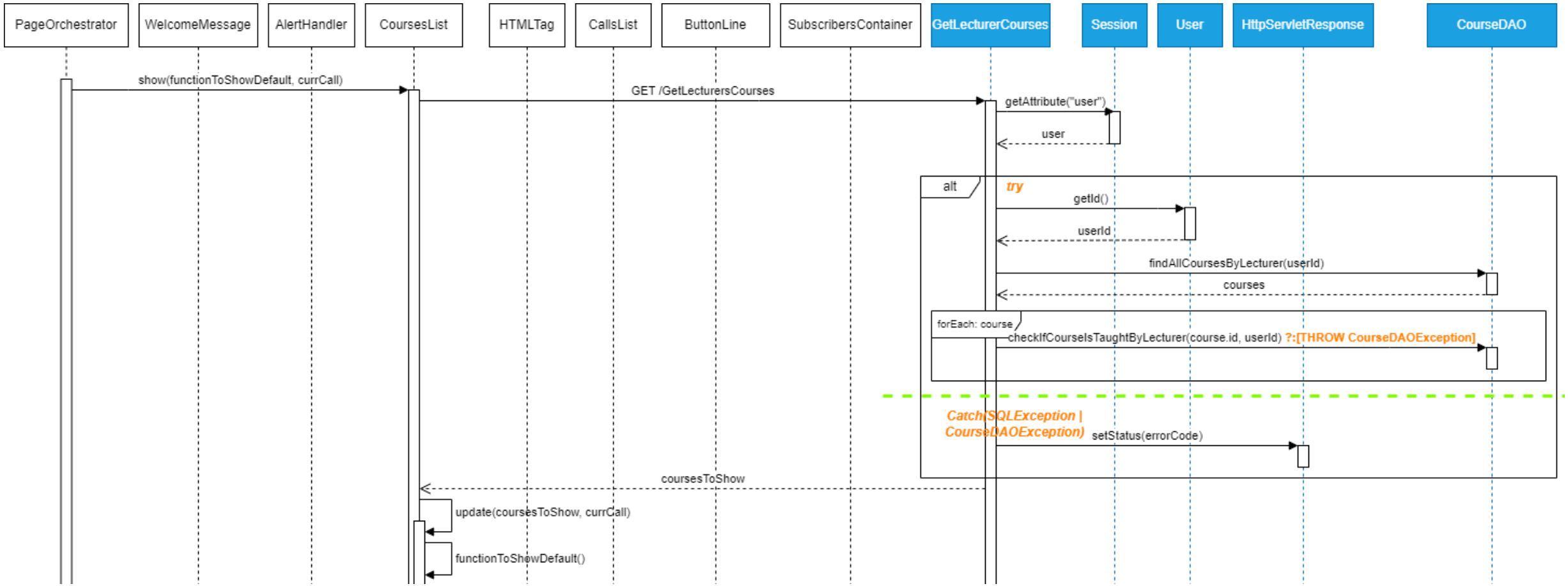
Sequence diagrams - Login



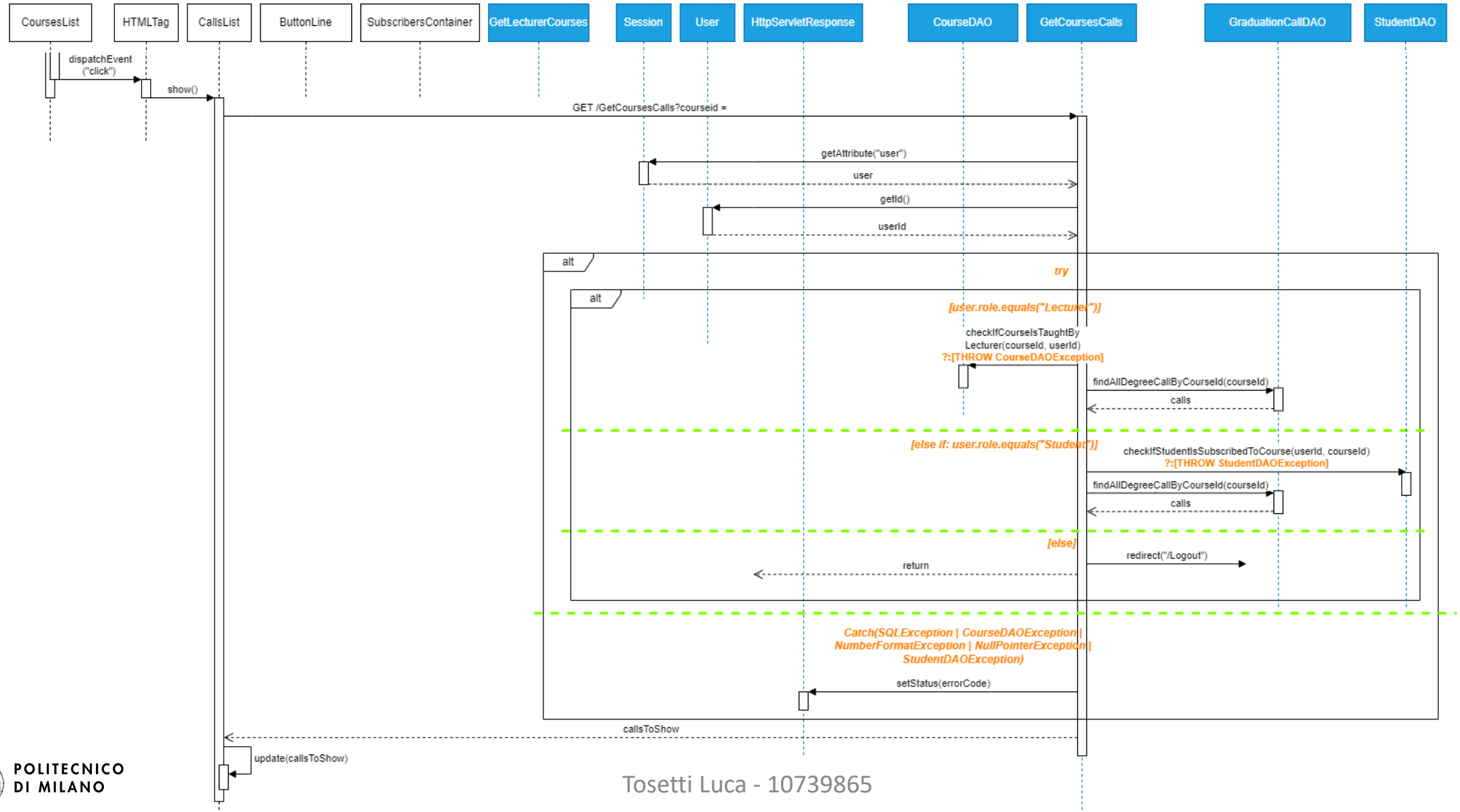
Sequence diagrams - HomeDocente



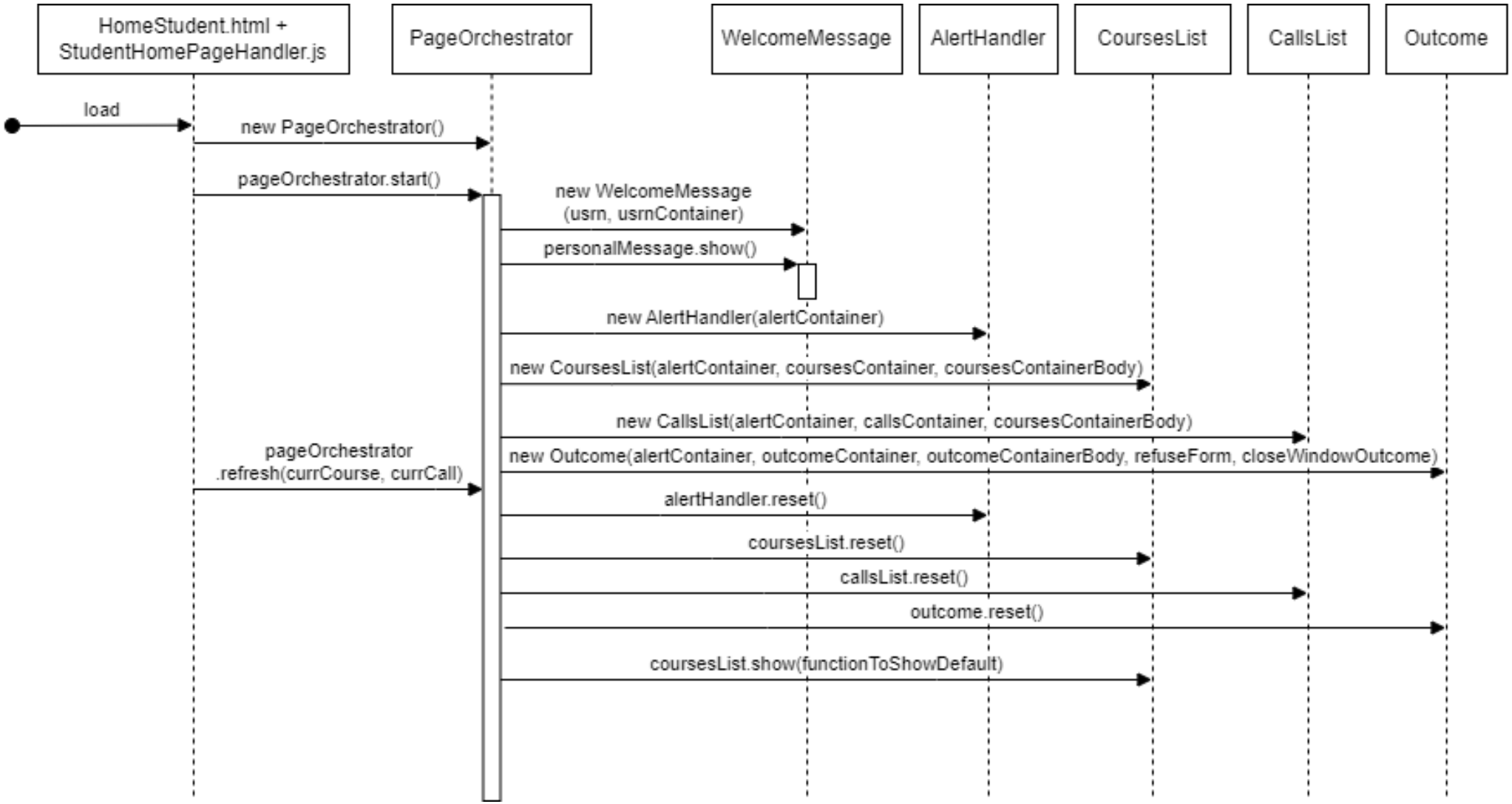
Sequence diagrams - HomeDocente



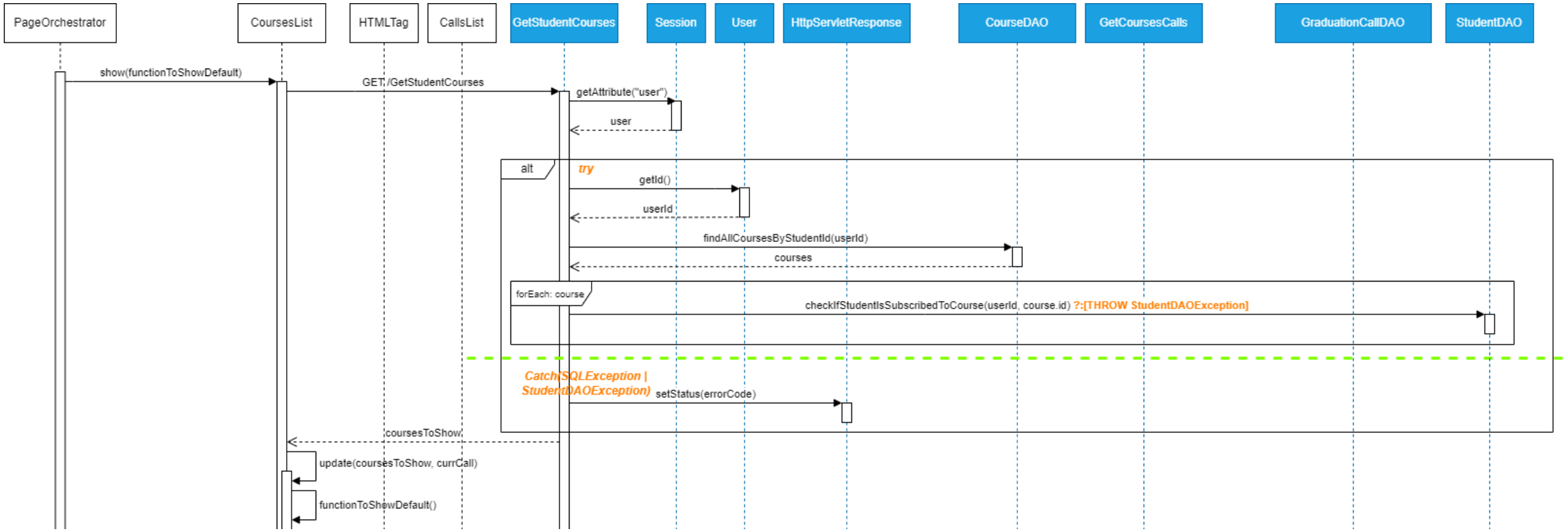
Sequence diagrams - HomeDocente



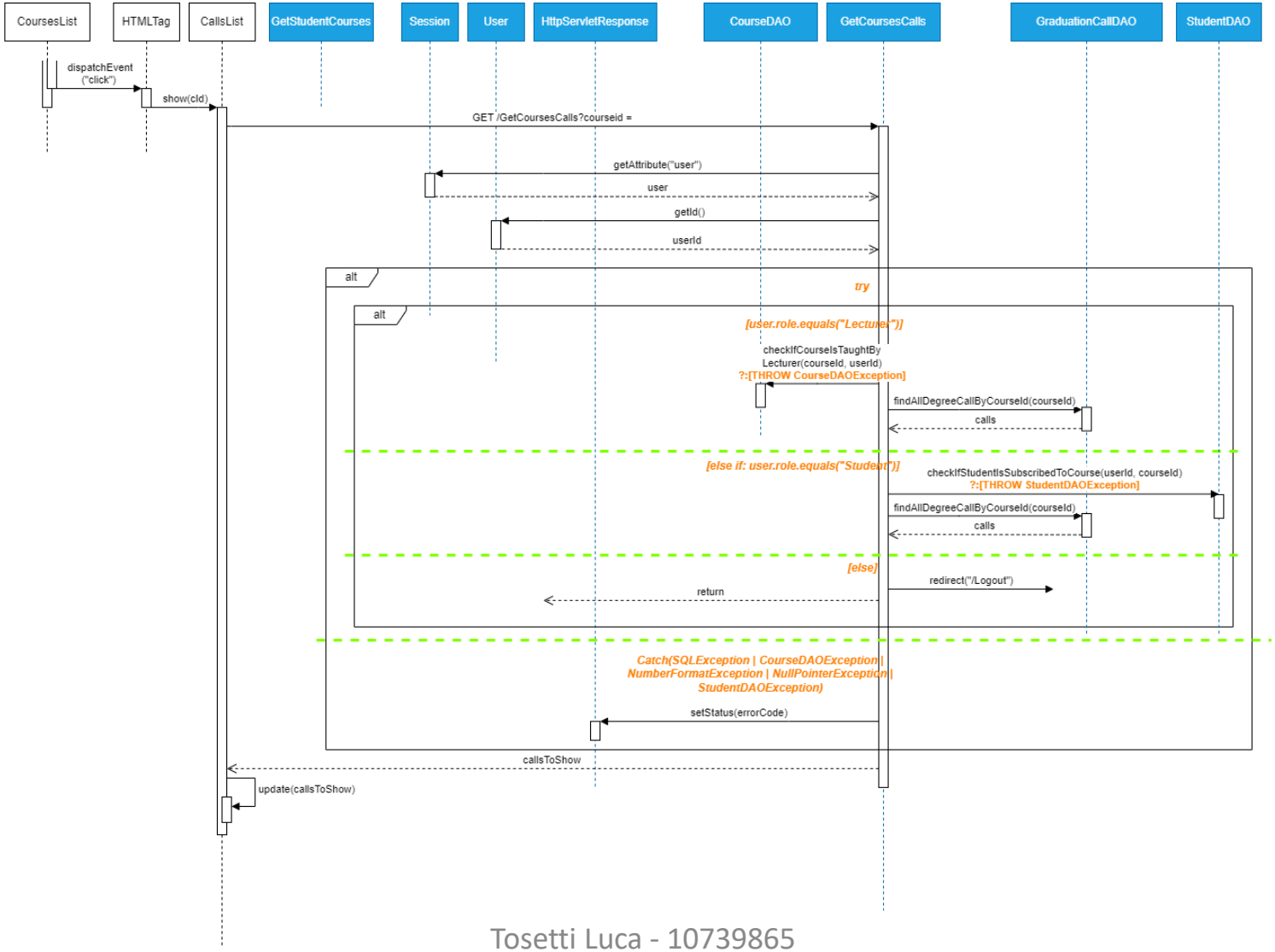
Sequence diagrams - HomeStudente



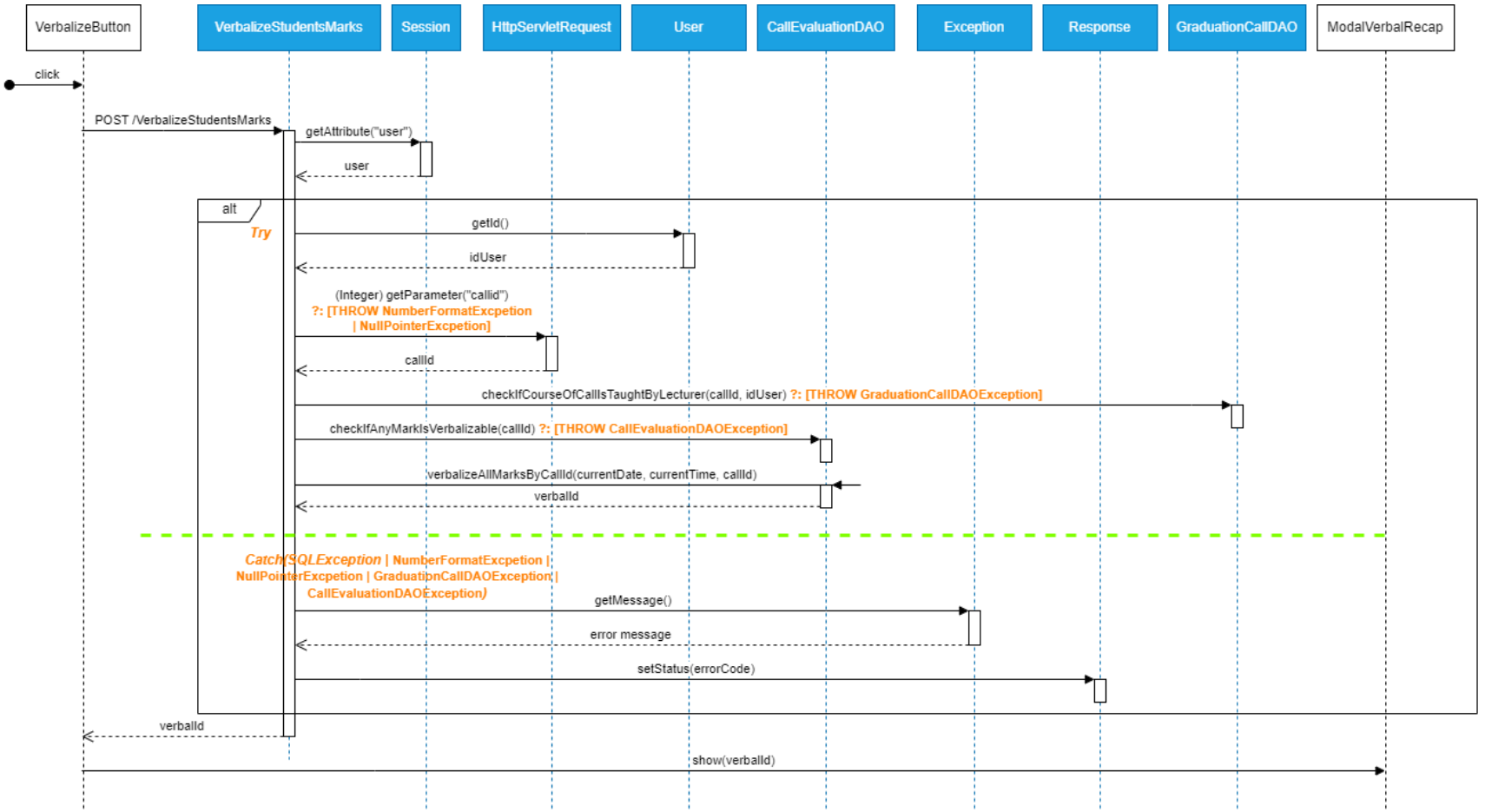
Sequence diagrams - HomeStudente



Sequence diagrams - HomeStudente



Sequence diagrams - VerbalizeButtonPress



Sequence diagrams - VerbalizeButtonPress