Esercizio 4 – HTML pure Version

Paterna – Restelli - Sanchini

Esercizio 4 : verbalizzazione degli esami

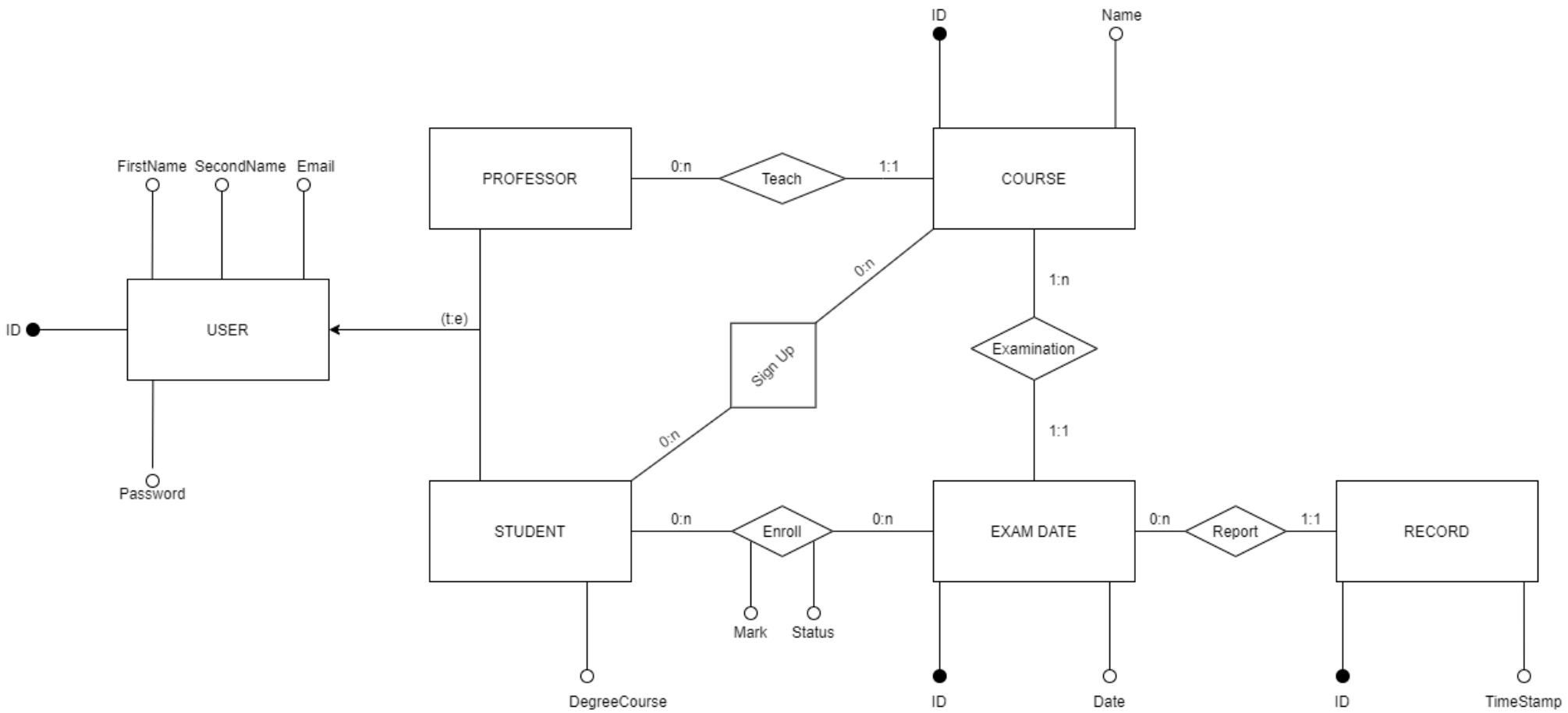
Un’applicazione permette di verbalizzare gli esiti degli esami di un appello. Il docente accede tramite login e seleziona nella HOME page un corso da una lista dei propri corsi ordinata in modo alfabetico decrescente e poi una data d’appello del corso scelto selezionata da un elenco ordinato per data 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: , assente, rimandato, riprovato, 18, 19, …, 30, 30 e lode. 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”. Alla tabella 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 iscriversi 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.

Data requirements analysis

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: , assente, rimandato, riprovato, 18, 19, …, 30, 30 e lode. 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". Alla tabella 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 iscriversi 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.

- Entities, attributes, relationships

Database design



Local database schema (1/3)

```
CREATE TABLE `user` (
  `ID` int NOT NULL,
  `name` varchar(45) NOT NULL,
  `surname` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `role` varchar(45) CHARACTER SET armscii8 COLLATE armscii8_general_ci NOT NULL,
  `coursedeg` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

```
CREATE TABLE `signup` (
  `IDStudent` int NOT NULL,
  `IDCourse` int NOT NULL,
  PRIMARY KEY (`IDStudent`,`IDCourse`),
  KEY `ID_Course_idx` (`IDCourse`),
  CONSTRAINT `ID_Course` FOREIGN KEY (`IDCourse`) REFERENCES `course` (`ID`),
  CONSTRAINT `ID_Student` FOREIGN KEY (`IDStudent`) REFERENCES `user` (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Local database schema (2/3)

```
CREATE TABLE `course` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `IDprofessor` int NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `IDprofessor_idx` (`IDprofessor`),
  CONSTRAINT `IDprofessor` FOREIGN KEY (`IDprofessor`) REFERENCES `user` (`ID`) ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

```
CREATE TABLE `examdate` (
  `IDExam` int NOT NULL AUTO_INCREMENT,
  `date` date NOT NULL,
  `IDCourse` int NOT NULL,
  PRIMARY KEY (`IDExam`),
  KEY `IDCourse_idx` (`IDCourse`),
  CONSTRAINT `IDCourse` FOREIGN KEY (`IDCourse`) REFERENCES `course` (`ID`) ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=1012 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Local database schema (3/3)

```
CREATE TABLE `enroll` (
  `IDStudent` int NOT NULL,
  `IDExamDate` int NOT NULL,
  `mark` varchar(15) DEFAULT NULL,
  `status` varchar(45) NOT NULL,
  `IDRecord` int DEFAULT NULL,
  PRIMARY KEY (`IDStudent`,`IDExamDate`),
  KEY `IDExamDate_idx` (`IDExamDate`),
  KEY `IDRecord_idx` (`IDRecord`),
  CONSTRAINT `ID_ExamDate` FOREIGN KEY (`IDExamDate`) REFERENCES `examdate` (`IDExam`) ON UPDATE CASCADE,
  CONSTRAINT `IDRecord` FOREIGN KEY (`IDRecord`) REFERENCES `record` (`ID`) ON UPDATE CASCADE,
  CONSTRAINT `IDStudent` FOREIGN KEY (`IDStudent`) REFERENCES `user` (`ID`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

```
CREATE TABLE `record` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `IDExamDate` int DEFAULT NULL,
  PRIMARY KEY (`ID`),
  KEY `IDExamDate_idx` (`IDExamDate`)
) ENGINE=InnoDB AUTO_INCREMENT=4283 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Application requirements analysis

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: , assente, rimandato, riprovato, 18, 19, …, 30, 30 e lode. 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”. Alla tabella 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 iscriversi 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

Components

Model Objects (beans):

- Course
- Enroll
- ExamDate
- Record
- Signup
- User

Controllers:

- CheckLogin
- GoToExamDatesPro
- GoToExamDatesStud
- GoToHomePagePro
- GoToHomePageStud
- GoToModify
- GoToRecord
- GoToResult
- GoToSessionEnrolls
- LogOut
- UpdateResultStud
- UpdateScore
- UpdateStatus

Filters

- LoginChecker
- ProfessorChecker
- StudentChecker

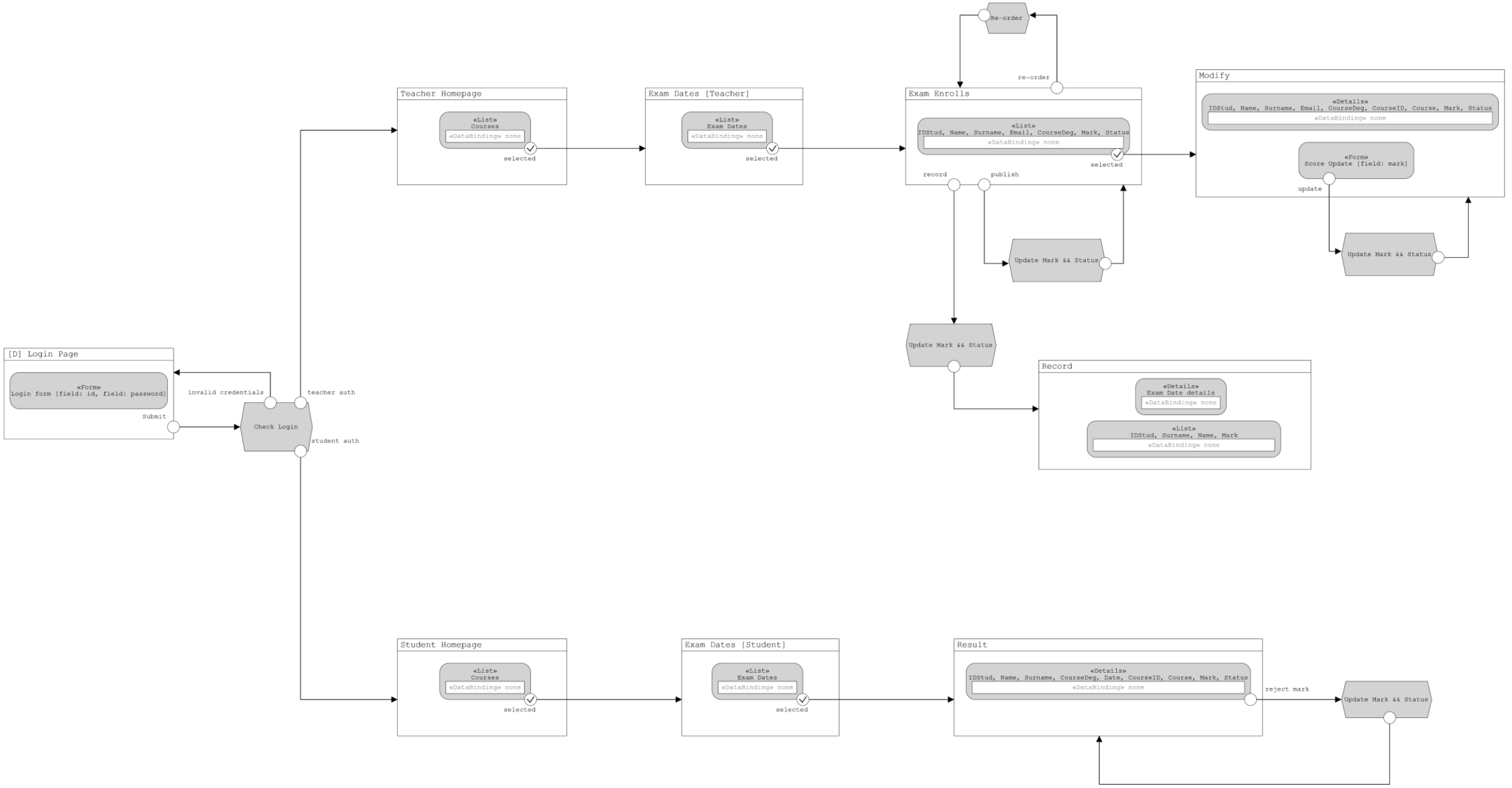
Views (templates)

- ExamDatesPro.html
- ExamDatesStud.html
- ExamDatesStudEmpty.html
- ExamEnrolls.html
- Forbidden.html
- HomePro.html
- HomeStud.html
- Modify.html
- Record.html
- Result.html
- ResultEmpty.html
- ResultLocked.html
- Warning.html
- Index.html

Data Access Objects

- CourseDAO
  - `findCoursesByIdProf()`
  - `findCoursesByIdStudent()`
  - `findCourseNameByIdExam()`
- EnrollsDAO
  - `findEnrolls()`
  - `findEnrollsOrderedByIDAsc() ... And for all the others possible orders`
  - `findStudentScore()`
  - `checkModifiableCondition()`
  - `insertMark()`
  - `refuseScore()`
  - `publishScore()`
  - `recordScore()`
  - `findRecordedStudents()`
  - `assertion_record()`
  - `assertion_published()`
- ExamDateDAO
  - `findExamDateBYCourseForProfessor()`
  - `findExamDateBYCourseForStudent()`
  - `checkExamDateByProf()`
- RecordDAO
  - `writeRecordOnDb()`
  - `getCurrentID()`
  - `getCurrentTimestamp()`
- UserDAO
  - `checkCredentials()`
  - `findProfessorByIdCourse()`

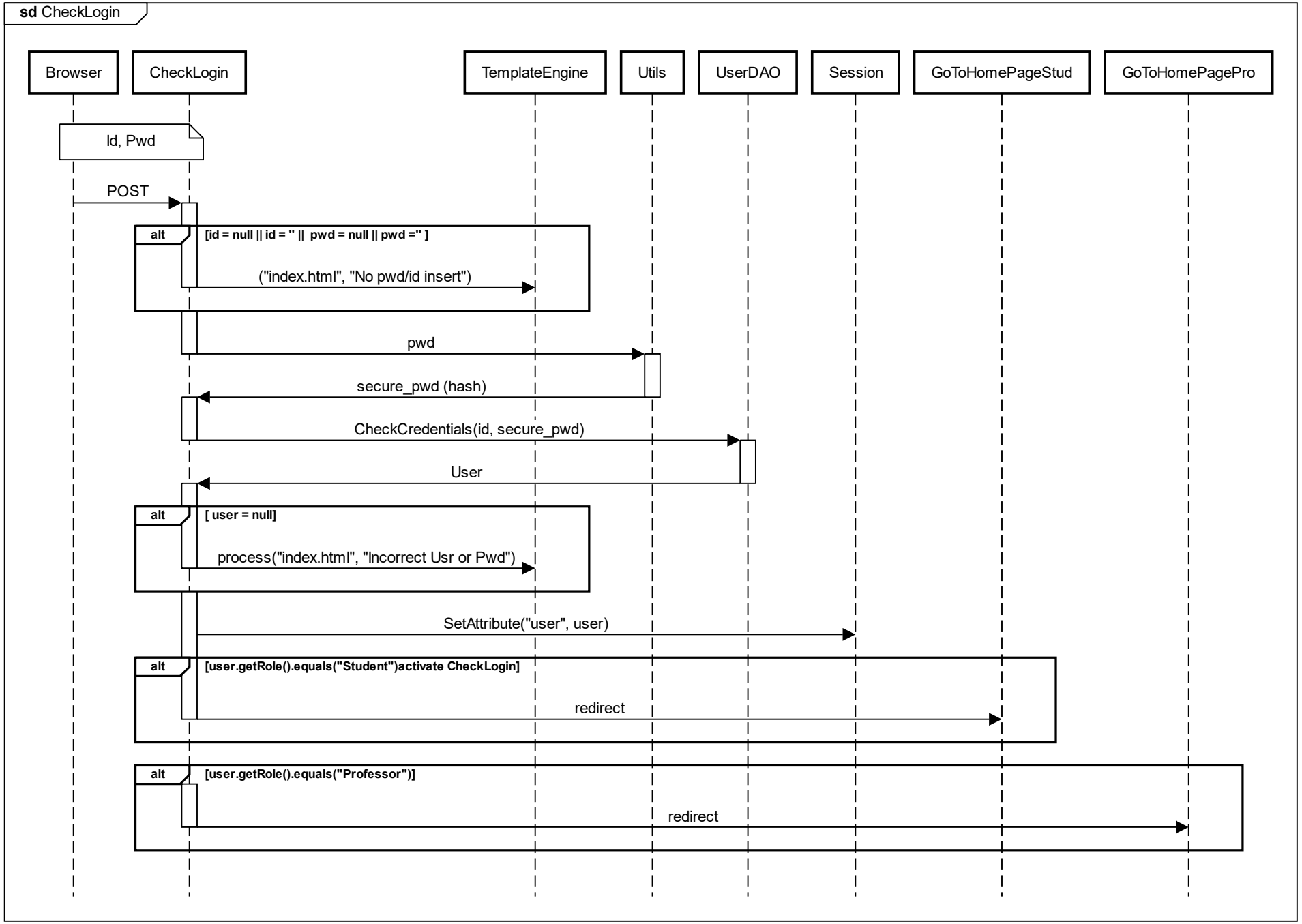
Design applicativo



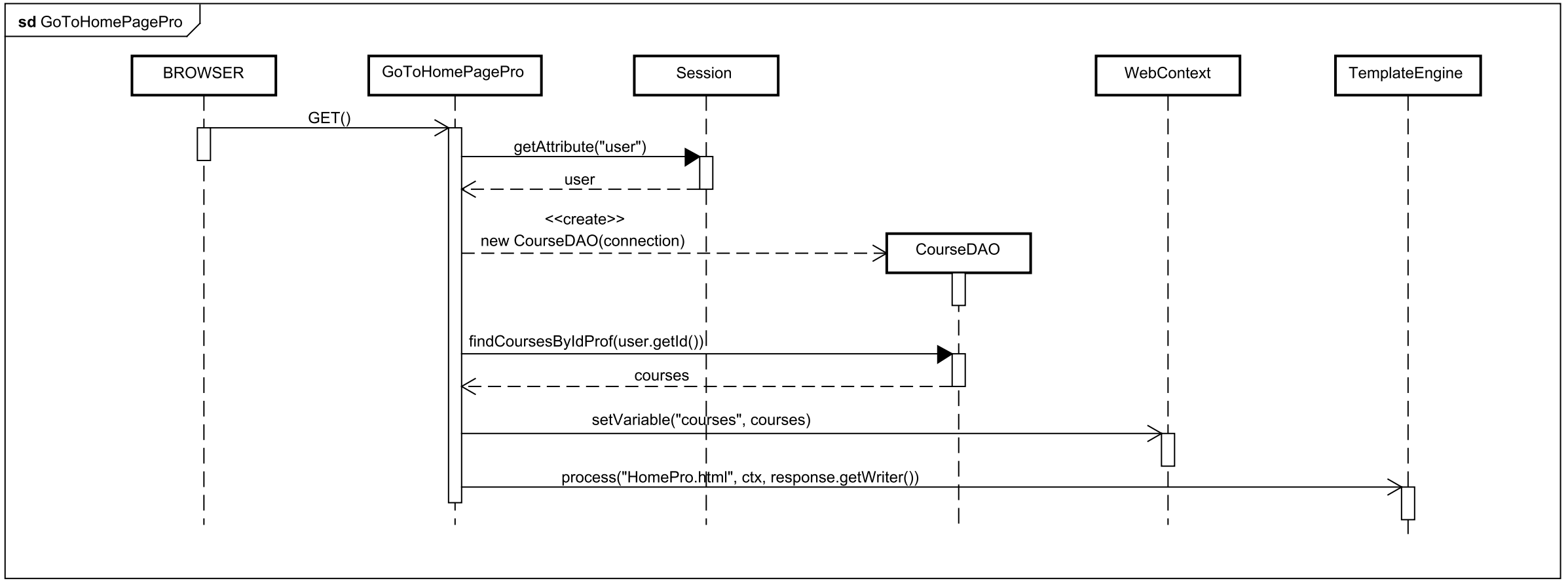
Sequence Diagrams

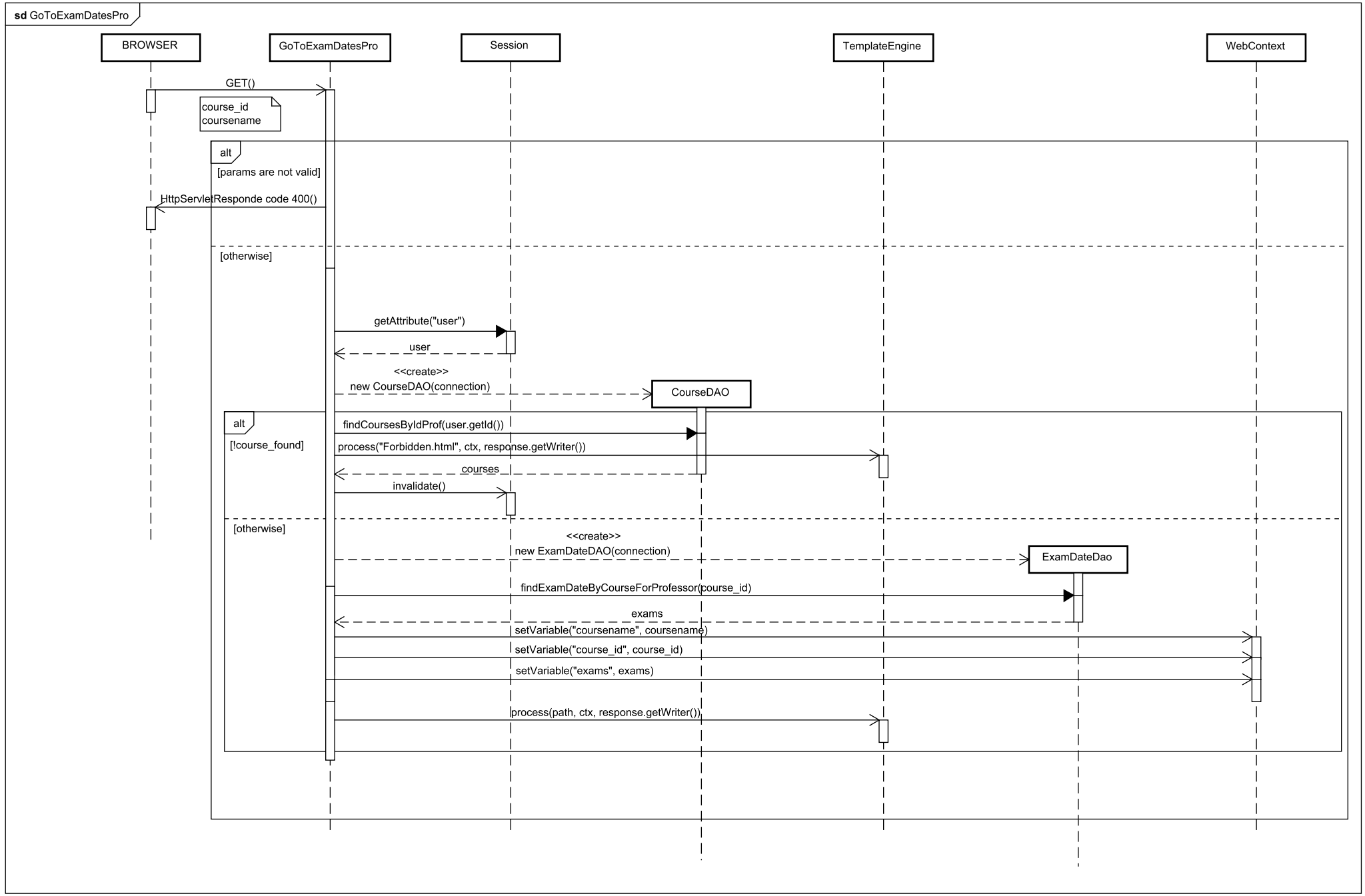
The following slides shows the sequence diagram of each possible interaction. Some details have been omitted to avoid repetitiveness (e.g. internal server errors, database access errors, validity parameters check).

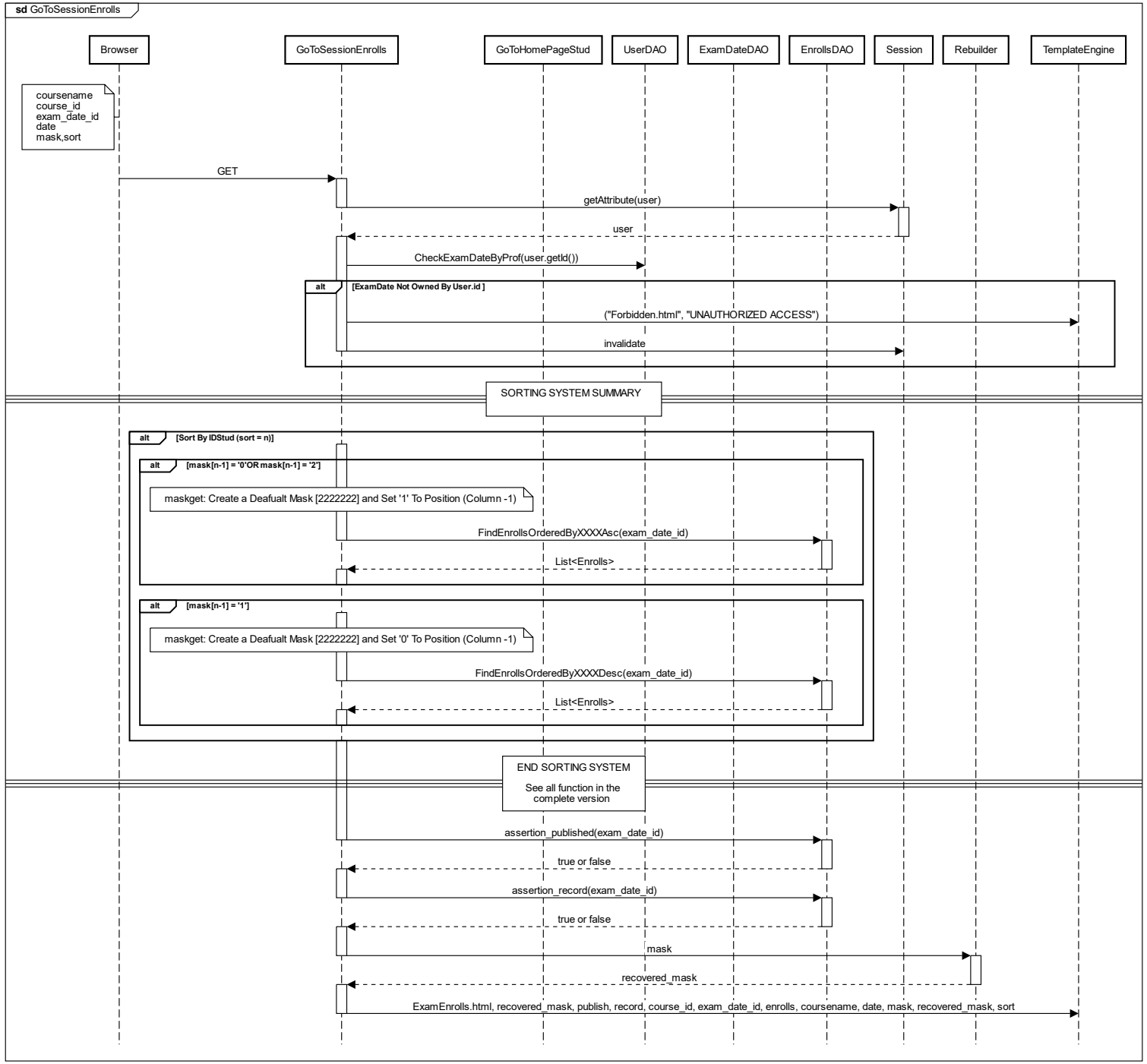
Filters checks behaviour is shown in specific diagrams. Their execution is omitted in every servlet diagram to avoid repetitiveness. They are executed before every servlet according to the filter mapping in web.xml file.

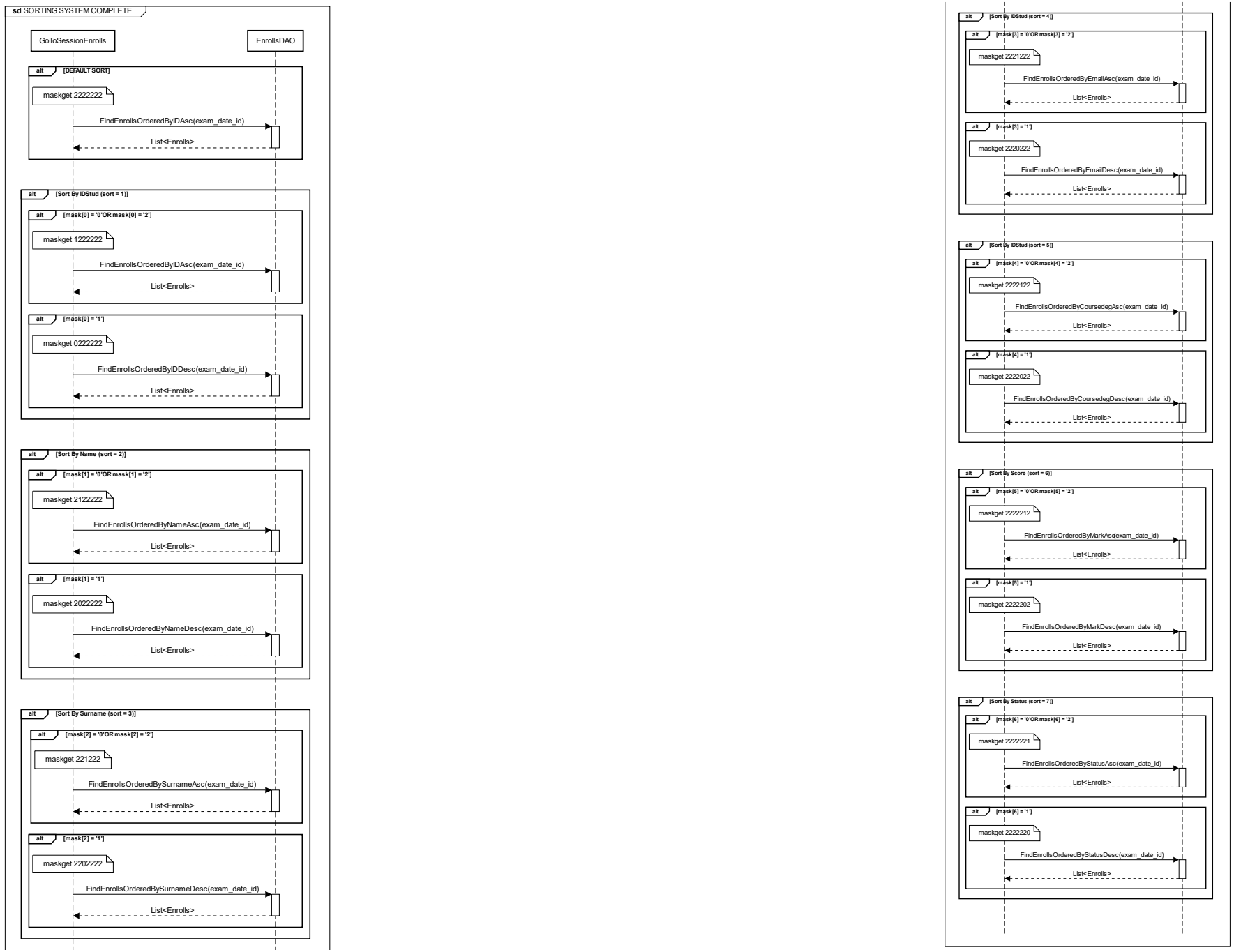

sd CheckLogin
Browser
CheckLogin
TemplateEngine
Utils
UserDAO
Session
GoToHomePageStud
GoToHomePagePro
Id, Pwd
POST
alt
[id = null || id = '' || pwd = null || pwd ='']
("index.html", "No pwd/id insert")
pwd
secure_pwd (hash)
CheckCredentials(id, secure_pwd)
User
alt
[user = null]
process("index.html", "Incorrect Usr or Pwd")
SetAttribute("user", user)
alt
[user.getRole().equals("Student")activate CheckLogin]
redirect
alt
[user.getRole().equals("Professor")]
redirect

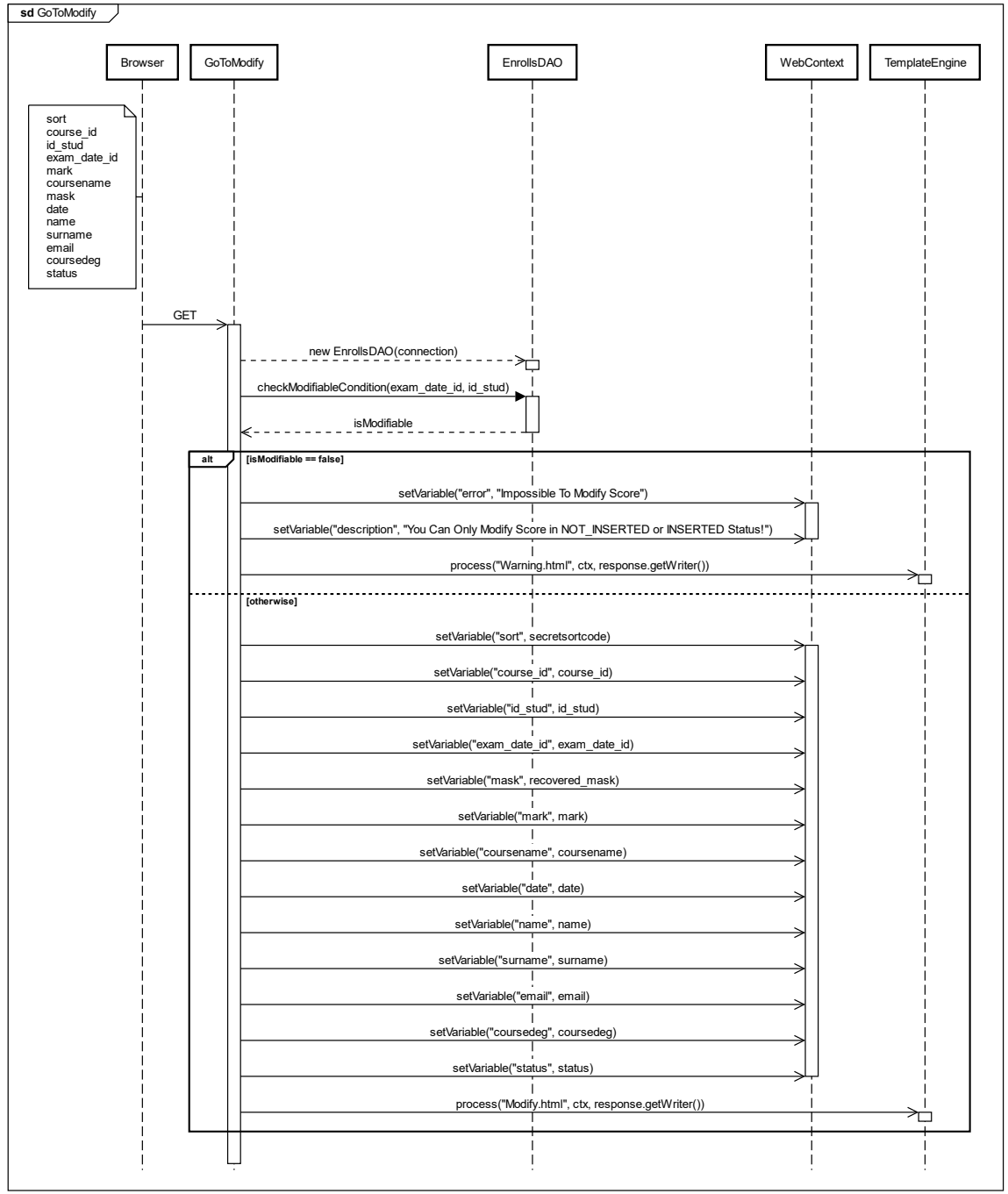
Professor

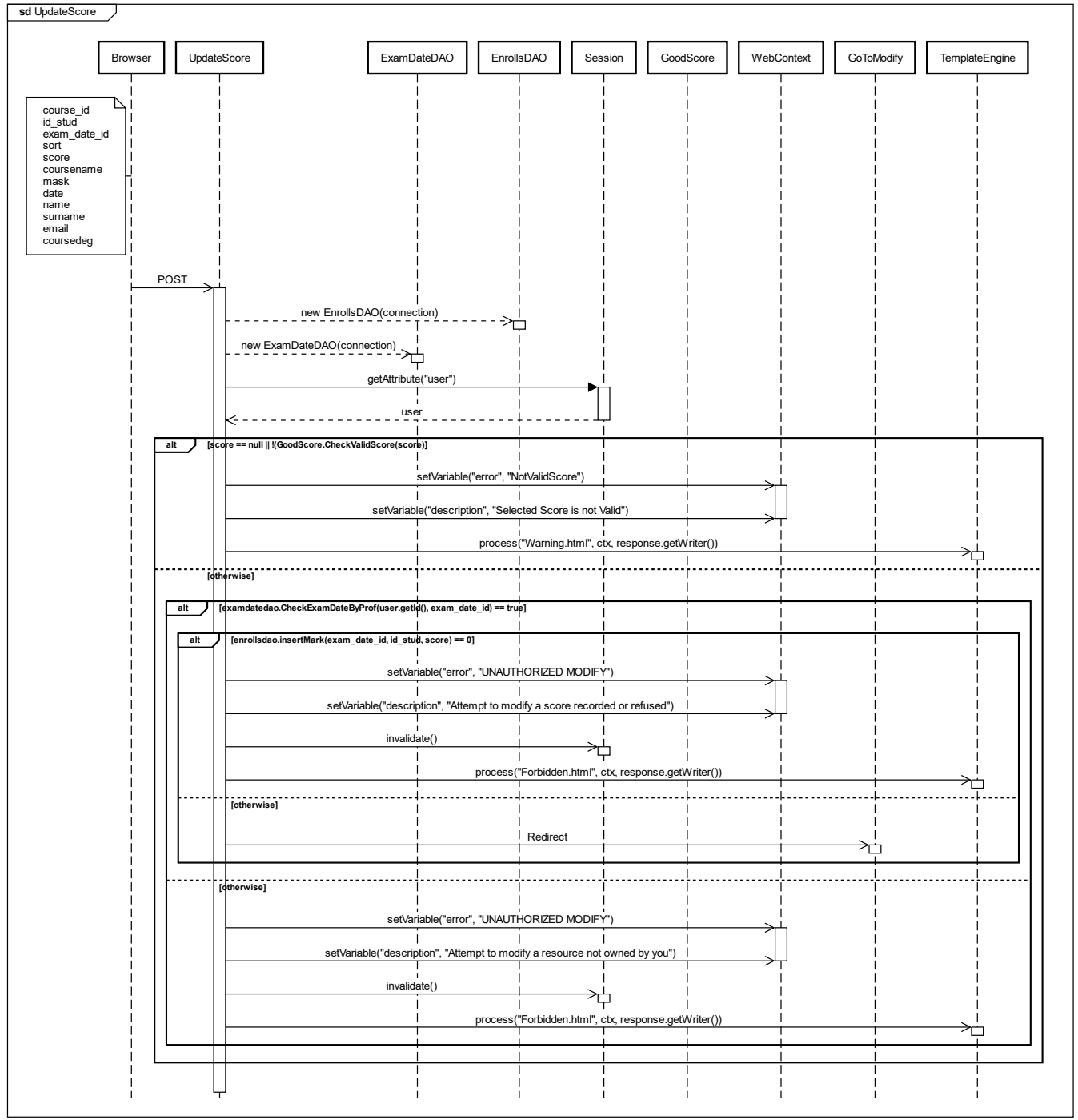

sd GoToHomePagePro
BROWSER
GoToHomePagePro
Session
WebContext
TemplateEngine
GET()
getAttribute("user")
user
<<create>>
new CourseDAO(connection)
CourseDAO
findCoursesByIdProf(user.getId())
courses
setVariable("courses", courses)
process("HomePro.html", ctx, response.getWriter())

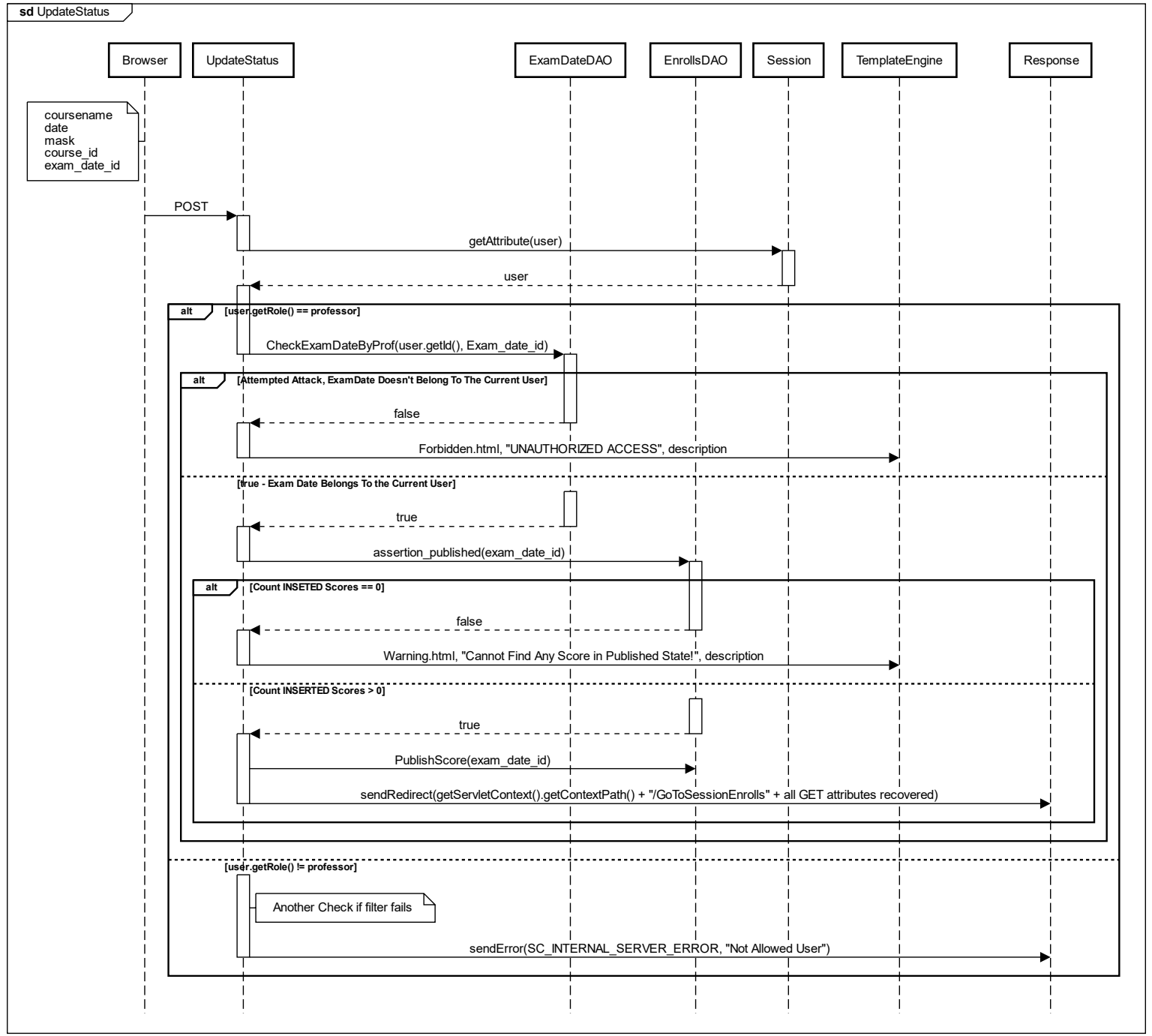

sd GoToExamDatesPro
BROWSER
GoToExamDatesPro
Session
TemplateEngine
WebContext
GET()
course_id
coursename
alt
[params are not valid]
HttpServletResponde code 400()
[otherwise]
getAttribute("user")
user
<<create>>
new CourseDAO(connection)
CourseDAO
alt
[!course_found]
findCoursesByIdProf(user.getId())
process("Forbidden.html", ctx, response.getWriter())
courses
invalidate()
[otherwise]
<<create>>
new ExamDateDAO(connection)
ExamDateDao
findExamDateByCourseForProfessor(course_id)
exams
setVariable("coursename", coursename)
setVariable("course_id", course_id)
setVariable("exams", exams)
process(path, ctx, response.getWriter())

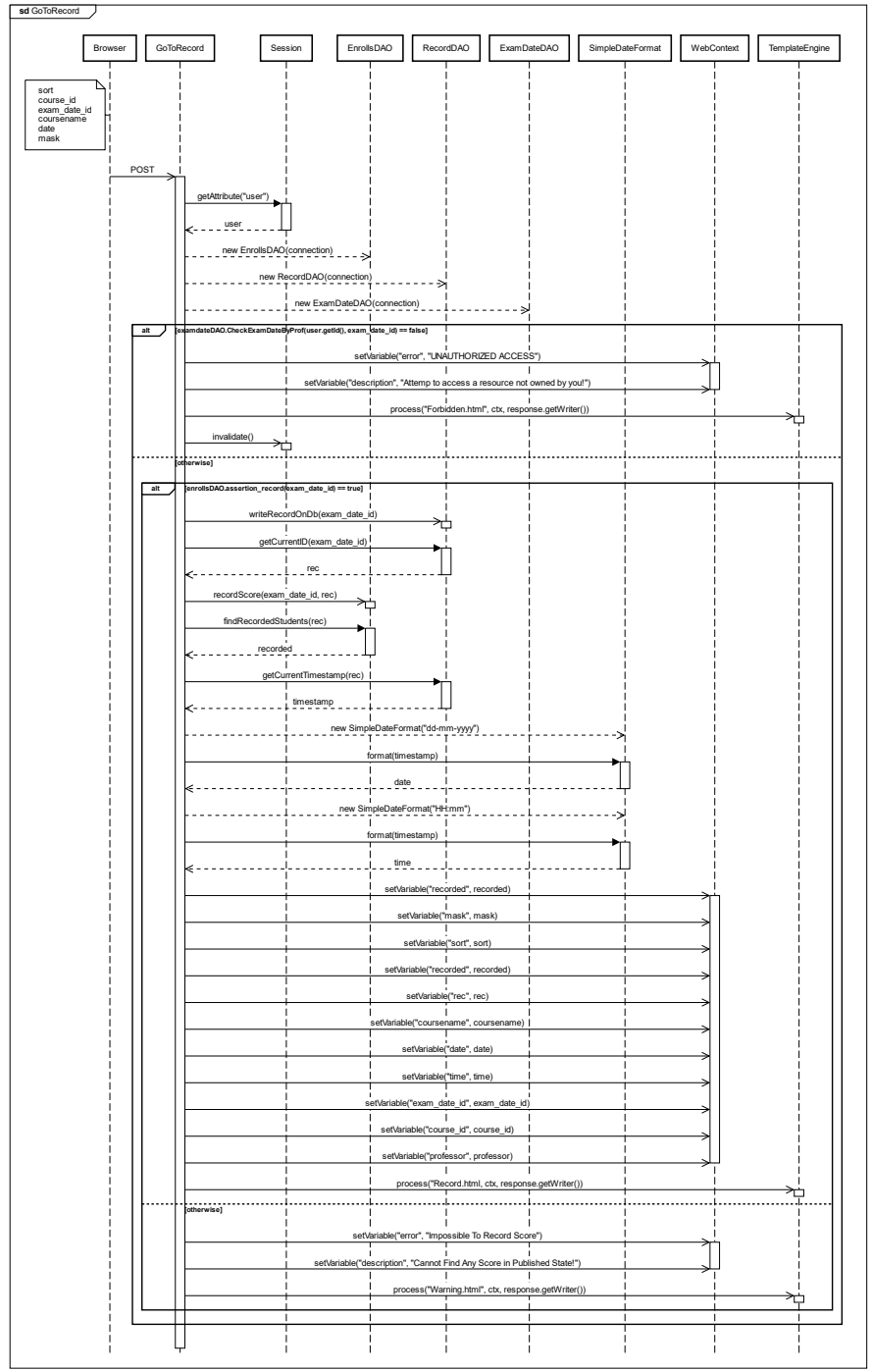

sd GoToSessionEnrolls
Browser
GoToSessionEnrolls
GoToHomePageStud
UserDAO
ExamDateDAO
EnrollsDAO
Session
Rebuilder
TemplateEngine
coursename
course_id
exam_date_id
date
mask,sort
GET
getAttribute(user)
user
CheckExamDateByProf(user.getId())
alt
[ExamDate Not Owned By User.id]
("Forbidden.html", "UNAUTHORIZED ACCESS")
invalidate
SORTING SYSTEM SUMMARY
alt
[Sort By IDStud (sort = n)]
alt
[mask[n-1] = '0'OR mask[n-1] = '2']
maskget: Create a Deafualt Mask [2222222] and Set '1' To Position (Column -1)
FindEnrollsOrderedByXXXXAsc(exam_date_id)
List<Enrolls>
alt
[mask[n-1] = '1']
maskget: Create a Deafualt Mask [2222222] and Set '0' To Position (Column -1)
FindEnrollsOrderedByXXXXDesc(exam_date_id)
List<Enrolls>
END SORTING SYSTEM
See all function in the complete version
assertion_published(exam_date_id)
true or false
assertion_record(exam_date_id)
true or false
mask
recovered_mask
ExamEnrolls.html, recovered_mask, publish, record, course_id, exam_date_id, enrolls, coursename, date, mask, recovered_mask, sort


sd SORTING SYSTEM COMPLETE
GoToSessionEnrolls
EnrollsDAO
alt [DEFAULT SORT]
maskget 2222222
FindEnrollsOrderedByIDAsc(exam_date_id)
List<Enrolls>
alt [Sort By IDStud (sort = 1)]
alt [mask[0] = '0'OR mask[0] = '2']
maskget 1222222
FindEnrollsOrderedByIDAsc(exam_date_id)
List<Enrolls>
alt [mask[0] = '1']
maskget 0222222
FindEnrollsOrderedByIDDesc(exam_date_id)
List<Enrolls>
alt [Sort By Name (sort = 2)]
alt [mask[1] = '0'OR mask[1] = '2']
maskget 2122222
FindEnrollsOrderedByNameAsc(exam_date_id)
List<Enrolls>
alt [mask[1] = '1']
maskget 2022222
FindEnrollsOrderedByNameDesc(exam_date_id)
List<Enrolls>
alt [Sort By Surname (sort = 3)]
alt [mask[2] = '0'OR mask[2] = '2']
maskget 221222
FindEnrollsOrderedBySurnameAsc(exam_date_id)
List<Enrolls>
alt [mask[2] = '1']
maskget 2202222
FindEnrollsOrderedBySurnameDesc(exam_date_id)
List<Enrolls>
alt [Sort By IDStud (sort = 4)]
alt [mask[3] = '0'OR mask[3] = '2']
maskget 2221222
FindEnrollsOrderedByEmailAsc(exam_date_id)
List<Enrolls>
alt [mask[3] = '1']
maskget 2220222
FindEnrollsOrderedByEmailDesc(exam_date_id)
List<Enrolls>
alt [Sort By IDStud (sort = 5)]
alt [mask[4] = '0'OR mask[4] = '2']
maskget 2222122
FindEnrollsOrderedByCoursedegAsc(exam_date_id)
List<Enrolls>
alt [mask[4] = '1']
maskget 2222022
FindEnrollsOrderedByCoursedegDesc(exam_date_id)
List<Enrolls>
alt [Sort By Score (sort = 6)]
alt [mask[5] = '0'OR mask[5] = '2']
maskget 2222212
FindEnrollsOrderedByMarkAsc(exam_date_id)
List<Enrolls>
alt [mask[5] = '1']
maskget 2222202
FindEnrollsOrderedByMarkDesc(exam_date_id)
List<Enrolls>
alt [Sort By Status (sort = 7)]
alt [mask[6] = '0'OR mask[6] = '2']
maskget 2222221
FindEnrollsOrderedByStatusAsc(exam_date_id)
List<Enrolls>
alt [mask[6] = '1']
maskget 2222220
FindEnrollsOrderedByStatusDesc(exam_date_id)
List<Enrolls>

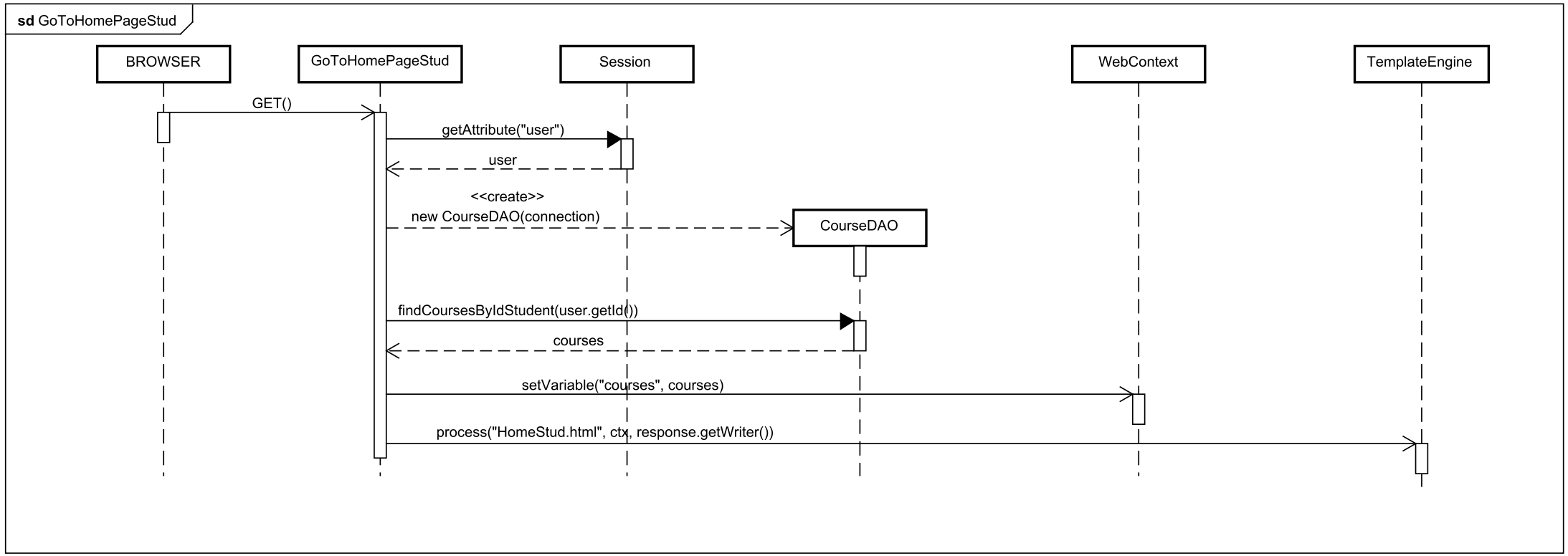

sd GoToModify
Browser
GoToModify
EnrollsDAO
WebContext
TemplateEngine
sort
course_id
id_stud
exam_date_id
mark
coursename
mask
date
name
surname
email
coursedeg
status
GET
new EnrollsDAO(connection)
checkModifiableCondition(exam_date_id, id_stud)
isModifiable
alt
[isModifiable == false]
setVariable("error", "Impossible To Modify Score")
setVariable("description", "You Can Only Modify Score in NOT_INSERTED or INSERTED Status!")
process("Warning.html", ctx, response.getWriter())
[otherwise]
setVariable("sort", secretsortcode)
setVariable("course_id", course_id)
setVariable("id_stud", id_stud)
setVariable("exam_date_id", exam_date_id)
setVariable("mask", recovered_mask)
setVariable("mark", mark)
setVariable("coursename", coursename)
setVariable("date", date)
setVariable("name", name)
setVariable("surname", surname)
setVariable("email", email)
setVariable("coursedeg", coursedeg)
setVariable("status", status)
process("Modify.html", ctx, response.getWriter())

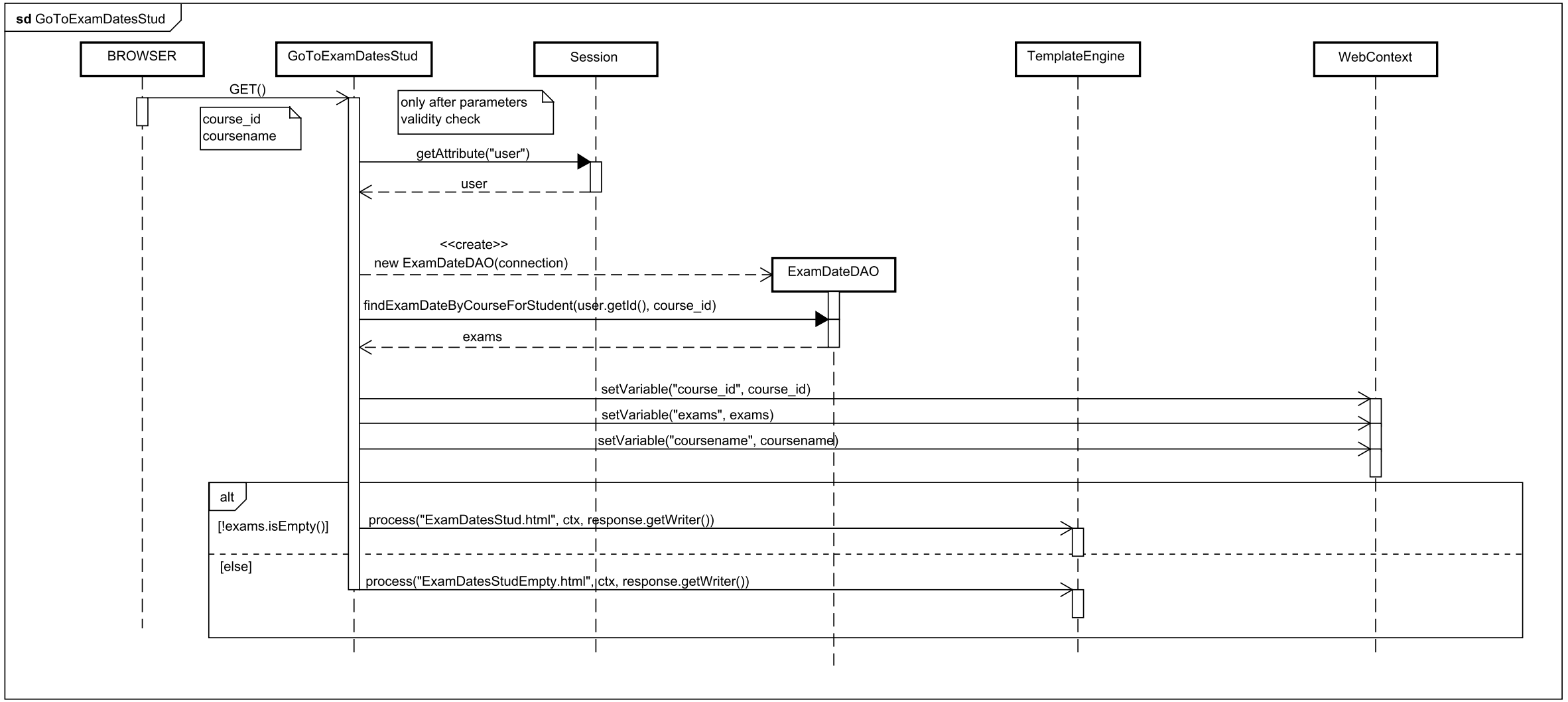

sd UpdateScore
Browser
UpdateScore
ExamDateDAO
EnrollsDAO
Session
GoodScore
WebContext
GoToModify
TemplateEngine
course_id
id_stud
exam_date_id
sort
score
coursename
mask
date
name
surname
email
coursedeg
POST
new EnrollsDAO(connection)
new ExamDateDAO(connection)
getAttribute("user")
user
alt
[score == null || !(GoodScore.CheckValidScore(score))]
setVariable("error", "NotValidScore")
setVariable("description", "Selected Score is not Valid")
process("Warning.html", ctx, response.getWriter())
[otherwise]
alt
[examdatedao.CheckExamDateByProf(user.getId(), exam_date_id) == true]
alt
[enrollsdao.insertMark(exam_date_id, id_stud, score) == 0]
setVariable("error", "UNAUTHORIZED MODIFY")
setVariable("description", "Attempt to modify a score recorded or refused")
invalidate()
process("Forbidden.html", ctx, response.getWriter())
[otherwise]
Redirect
[otherwise]
setVariable("error", "UNAUTHORIZED MODIFY")
setVariable("description", "Attempt to modify a resource not owned by you")
invalidate()
process("Forbidden.html", ctx, response.getWriter())

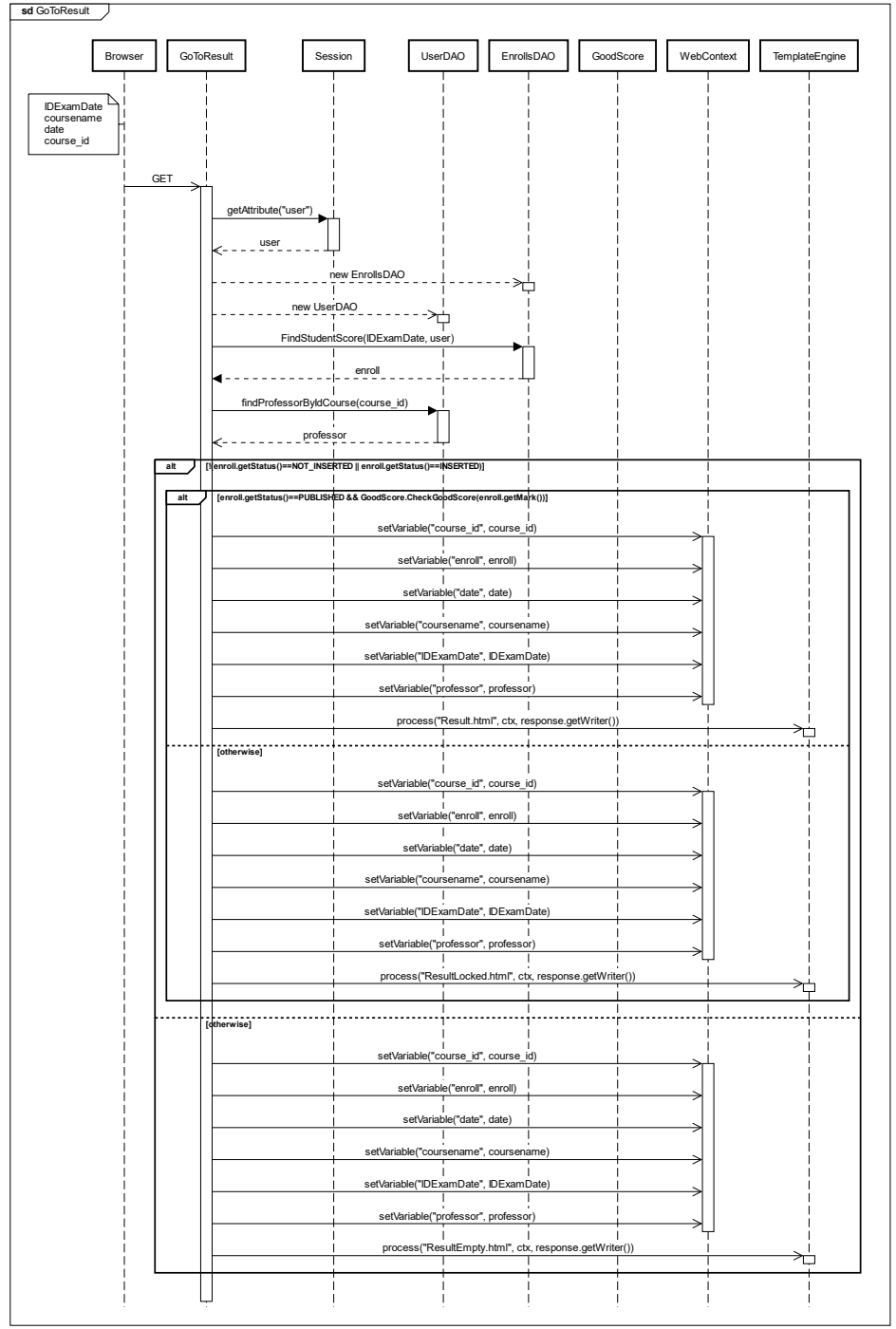

sd UpdateStatus
Browser
UpdateStatus
ExamDateDAO
EnrollsDAO
Session
TemplateEngine
Response
coursename
date
mask
course_id
exam_date_id
POST
getAttribute(user)
user
alt
[user.getRole() == professor]
CheckExamDateByProf(user.getId(), Exam_date_id)
alt
[Attempted Attack, ExamDate Doesn't Belong To The Current User]
false
Forbidden.html, "UNAUTHORIZED ACCESS", description
[true - Exam Date Belongs To the Current User]
true
assertion_published(exam_date_id)
alt
[Count INSETED Scores == 0]
false
Warning.html, "Cannot Find Any Score in Published State!", description
[Count INSERTED Scores > 0]
true
PublishScore(exam_date_id)
sendRedirect(getServletContext().getContextPath() + "/GoToSessionEnrolls" + all GET attributes recovered)
[user.getRole() != professor]
Another Check if filter fails
sendError(SC_INTERNAL_SERVER_ERROR, "Not Allowed User")

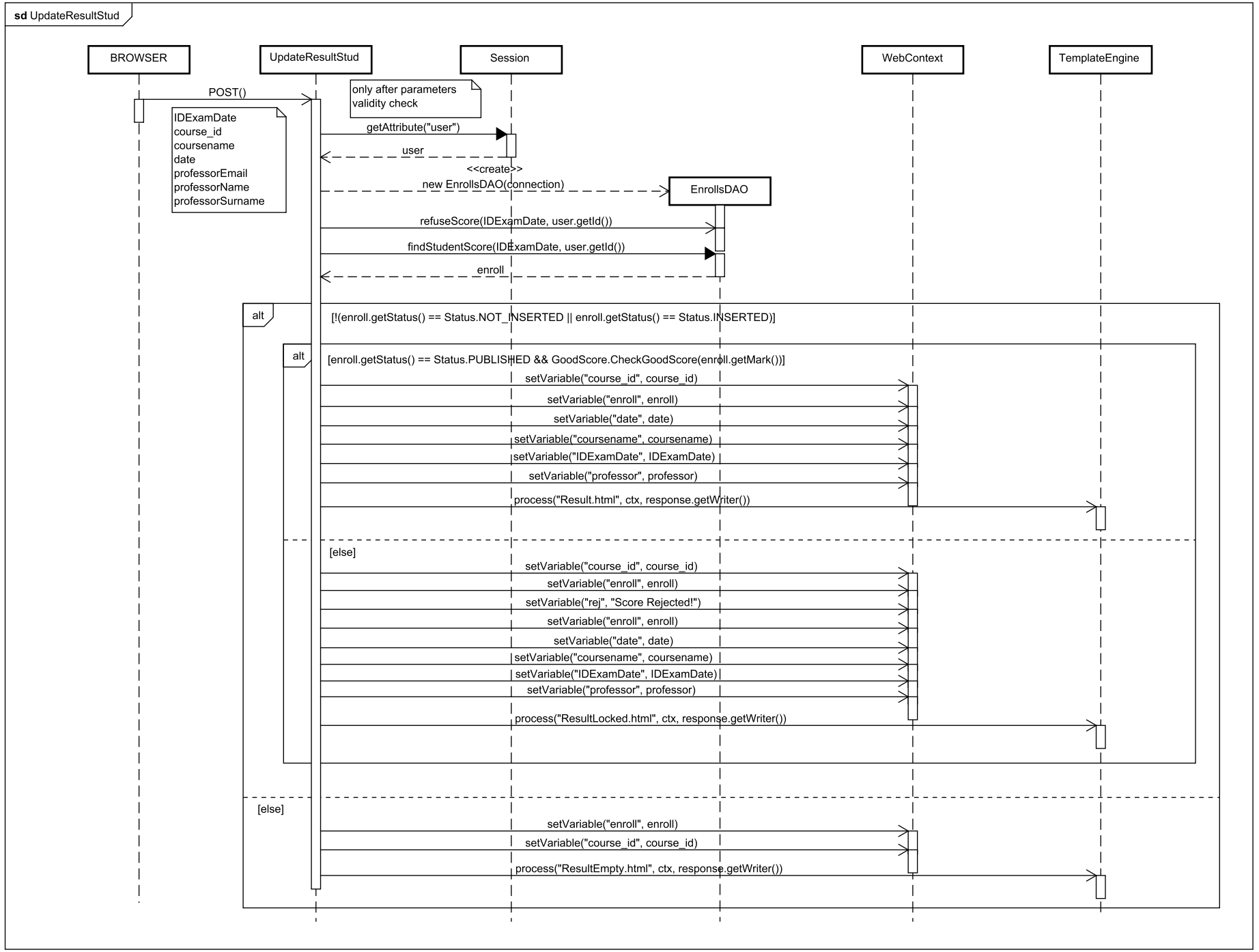

sd GoToRecord
Browser
GoToRecord
Session
EnrollsDAO
RecordDAO
ExamDateDAO
SimpleDateFormat
WebContext
TemplateEngine
sort
course_id
exam_date_id
coursename
date
mask
POST
getAttribute("user")
user
new EnrollsDAO(connection)
new RecordDAO(connection)
new ExamDateDAO(connection)
alt
[examdateDAO.CheckExamDateByProf(user.getId(), exam_date_id) == false]
setVariable("error", "UNAUTHORIZED ACCESS")
setVariable("description", "Attemp to access a resource not owned by you!")
process("Forbidden.html", ctx, response.getWriter())
invalidate()
[otherwise]
alt
[enrollsDAO.assertion_record(exam_date_id) == true]
writeRecordOnDb(exam_date_id)
getCurrentID(exam_date_id)
rec
recordScore(exam_date_id, rec)
findRecordedStudents(rec)
recorded
getCurrentTimestamp(rec)
timestamp
new SimpleDateFormat("dd-mm-yyyy")
format(timestamp)
date
new SimpleDateFormat("HH:mm")
format(timestamp)
time
setVariable("recorded", recorded)
setVariable("mask", mask)
setVariable("sort", sort)
setVariable("recorded", recorded)
setVariable("rec", rec)
setVariable("coursename", coursename)
setVariable("date", date)
setVariable("time", time)
setVariable("exam_date_id", exam_date_id)
setVariable("course_id", course_id)
setVariable("professor", professor)
process("Record.html, ctx, response.getWriter())
[otherwise]
setVariable("error", "Impossible To Record Score")
setVariable("description", "Cannot Find Any Score in Published State!")
process("Warning.html", ctx, response.getWriter())

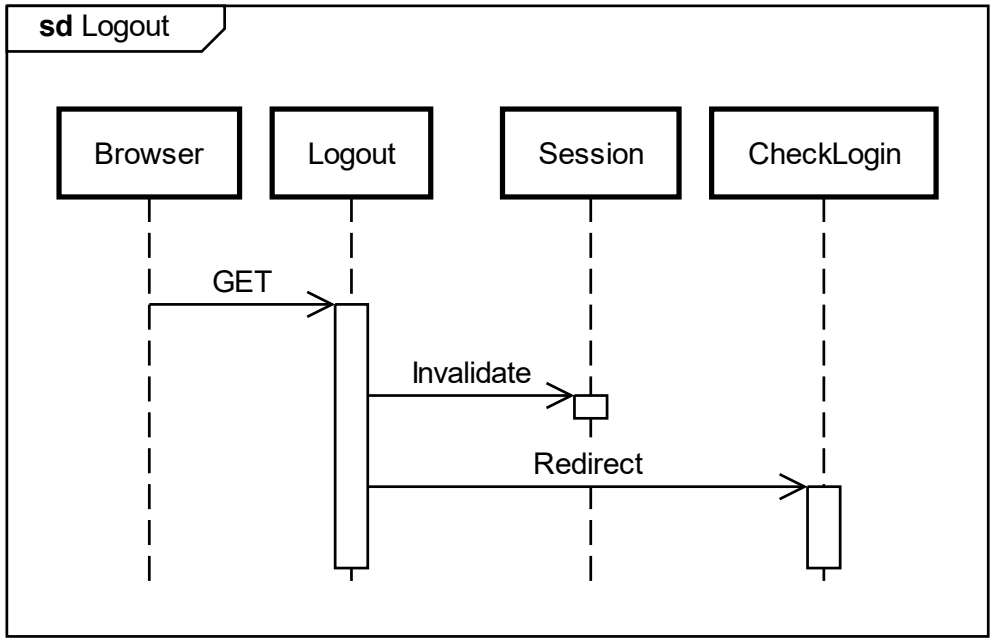
Student


sd GoToHomePageStud
BROWSER
GoToHomePageStud
Session
WebContext
TemplateEngine
GET()
getAttribute("user")
user
<<create>>
new CourseDAO(connection)
CourseDAO
findCoursesByIdStudent(user.getId())
courses
setVariable("courses", courses)
process("HomeStud.html", ctx, response.getWriter())

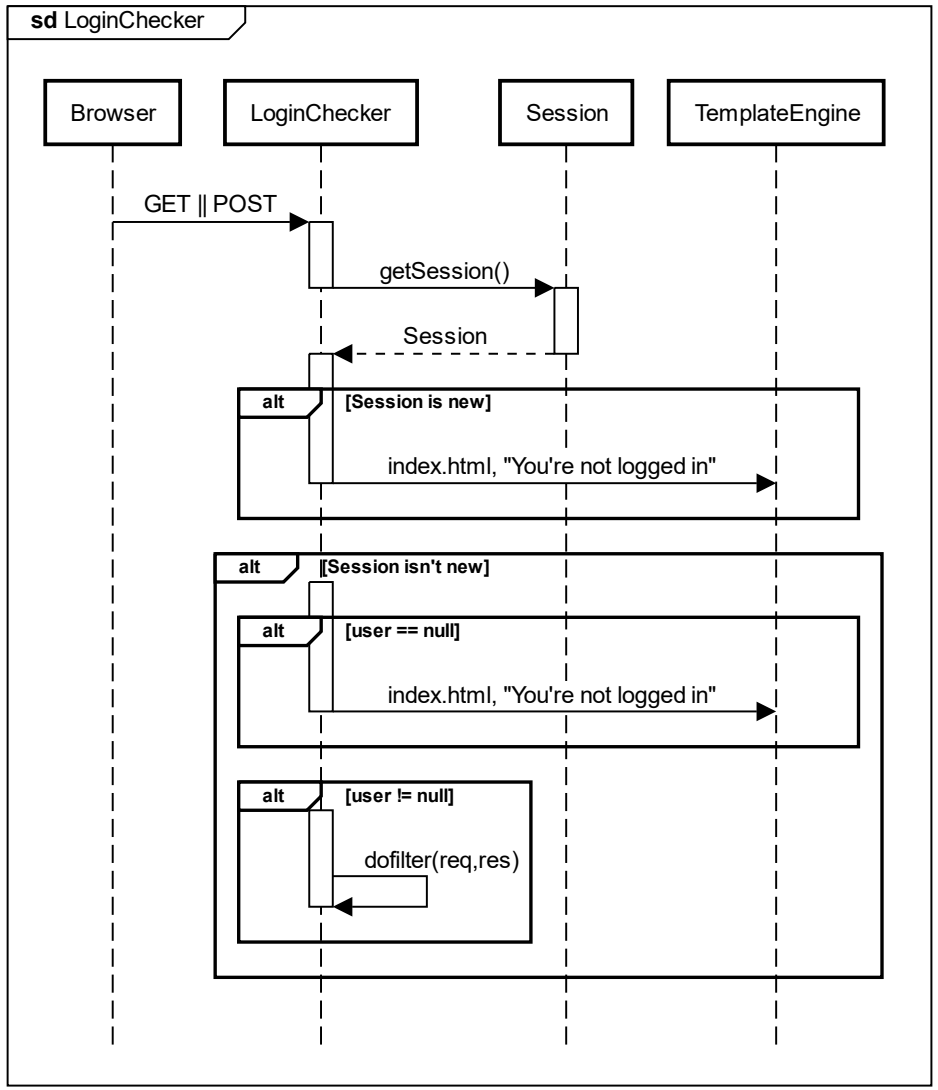

sd GoToExamDatesStud
BROWSER
GoToExamDatesStud
Session
TemplateEngine
WebContext
GET()
course_id
coursename
only after parameters
validity check
getAttribute("user")
user
<<create>>
new ExamDateDAO(connection)
ExamDateDAO
findExamDateByCourseForStudent(user.getId(), course_id)
exams
setVariable("course_id", course_id)
setVariable("exams", exams)
setVariable("coursename", coursename)
alt
[!exams.isEmpty()]
process("ExamDatesStud.html", ctx, response.getWriter())
[else]
process("ExamDatesStudEmpty.html", ctx, response.getWriter())

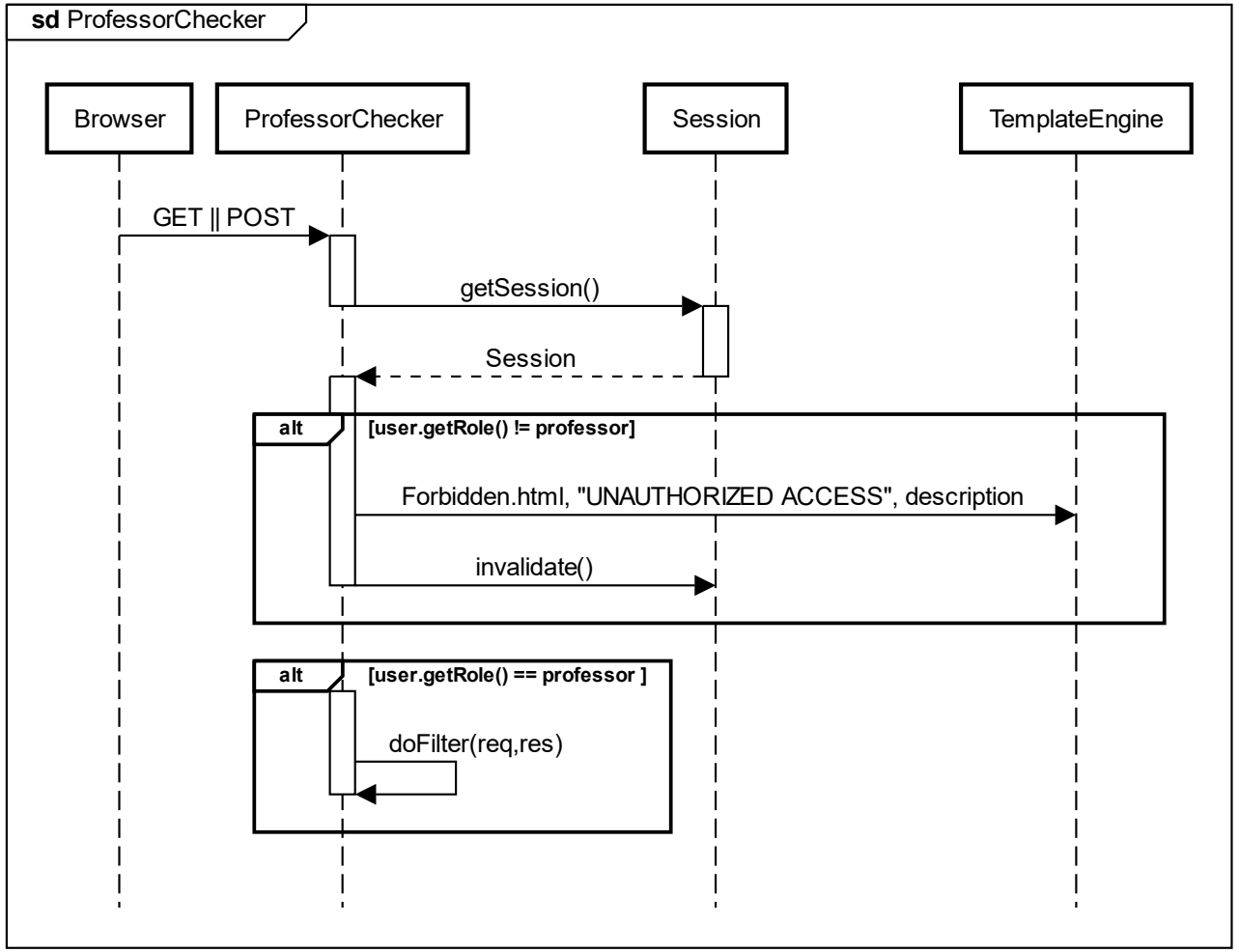

sd GoToResult
Browser
GoToResult
Session
UserDAO
EnrollsDAO
GoodScore
WebContext
TemplateEngine
IDExamDate
coursename
date
course_id
GET
getAttribute("user")
user
new EnrollsDAO
new UserDAO
FindStudentScore(IDExamDate, user)
enroll
findProfessorByIdCourse(course_id)
professor
alt
[!enroll.getStatus()==NOT_INSERTED || enroll.getStatus()==INSERTED)]
alt
[enroll.getStatus()==PUBLISHED && GoodScore.CheckGoodScore(enroll.getMark())]
setVariable("course_id", course_id)
setVariable("enroll", enroll)
setVariable("date", date)
setVariable("coursename", coursename)
setVariable("IDExamDate", IDExamDate)
setVariable("professor", professor)
process("Result.html", ctx, response.getWriter())
[otherwise]
setVariable("course_id", course_id)
setVariable("enroll", enroll)
setVariable("date", date)
setVariable("coursename", coursename)
setVariable("IDExamDate", IDExamDate)
setVariable("professor", professor)
process("ResultLocked.html", ctx, response.getWriter())
[otherwise]
setVariable("course_id", course_id)
setVariable("enroll", enroll)
setVariable("date", date)
setVariable("coursename", coursename)
setVariable("IDExamDate", IDExamDate)
setVariable("professor", professor)
process("ResultEmpty.html", ctx, response.getWriter())

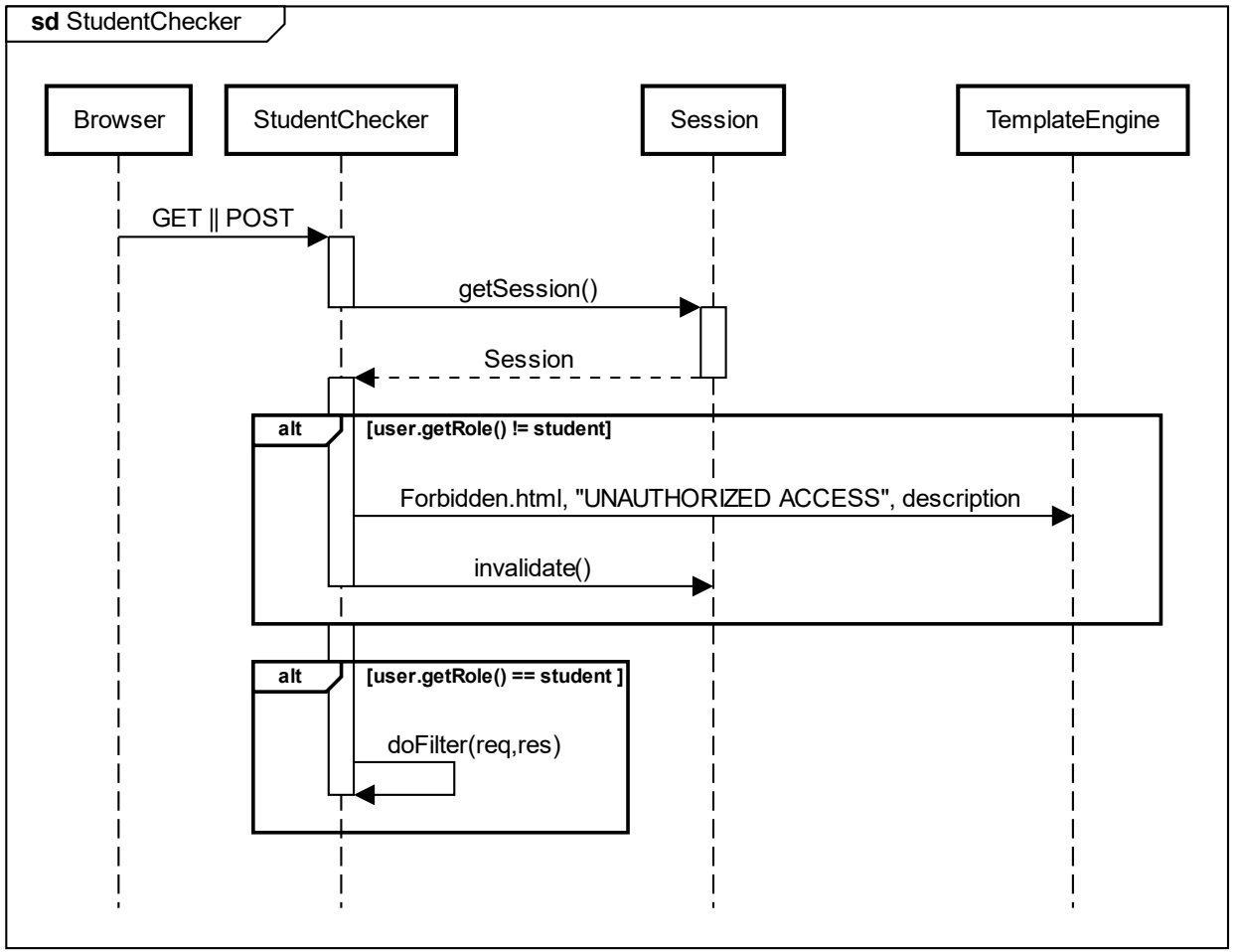

sd UpdateResultStud
BROWSER
UpdateResultStud
Session
WebContext
TemplateEngine
POST()
IDExamDate
course_id
coursename
date
professorEmail
professorName
professorSurname
only after parameters validity check
getAttribute("user")
user
<<create>>
new EnrollsDAO(connection)
EnrollsDAO
refuseScore(IDExamDate, user.getId())
findStudentScore(IDExamDate, user.getId())
enroll
alt
[!(enroll.getStatus() == Status.NOT_INSERTED || enroll.getStatus() == Status.INSERTED)]
alt
[enroll.getStatus() == Status.PUBLISHED && GoodScore.CheckGoodScore(enroll.getMark())]
setVariable("course_id", course_id)
setVariable("enroll", enroll)
setVariable("date", date)
setVariable("coursename", coursename)
setVariable("IDExamDate", IDExamDate)
setVariable("professor", professor)
process("Result.html", ctx, response.getWriter())
[else]
setVariable("course_id", course_id)
setVariable("enroll", enroll)
setVariable("rej", "Score Rejected!")
setVariable("enroll", enroll)
setVariable("date", date)
setVariable("coursename", coursename)
setVariable("IDExamDate", IDExamDate)
setVariable("professor", professor)
process("ResultLocked.html", ctx, response.getWriter())
[else]
setVariable("enroll", enroll)
setVariable("course_id", course_id)
process("ResultEmpty.html", ctx, response.getWriter())


sd Logout
Browser
Logout
Session
CheckLogin
GET
Invalidate
Redirect

Filters


sd LoginChecker
Browser
LoginChecker
Session
TemplateEngine
GET || POST
getSession()
Session
alt
[Session is new]
index.html, "You're not logged in"
alt
[Session isn't new]
alt
[user == null]
index.html, "You're not logged in"
alt
[user != null]
dofilter(req,res)


sd ProfessorChecker
Browser
ProfessorChecker
Session
TemplateEngine
GET || POST
getSession()
Session
alt
[user.getRole() != professor]
Forbidden.html, "UNAUTHORIZED ACCESS", description
invalidate()
alt
[user.getRole() == professor]
doFilter(req,res)


sd StudentChecker
Browser
StudentChecker
Session
TemplateEngine
GET || POST
getSession()
Session
alt
[user.getRole() != student]
Forbidden.html, "UNAUTHORIZED ACCESS", description
invalidate()
alt
[user.getRole() == student]
doFilter(req,res)