# Esercizio 4 – RIA Version

Paterna – Restelli - Sanchini

# Esercizio 4 : verbalizzazione degli esami

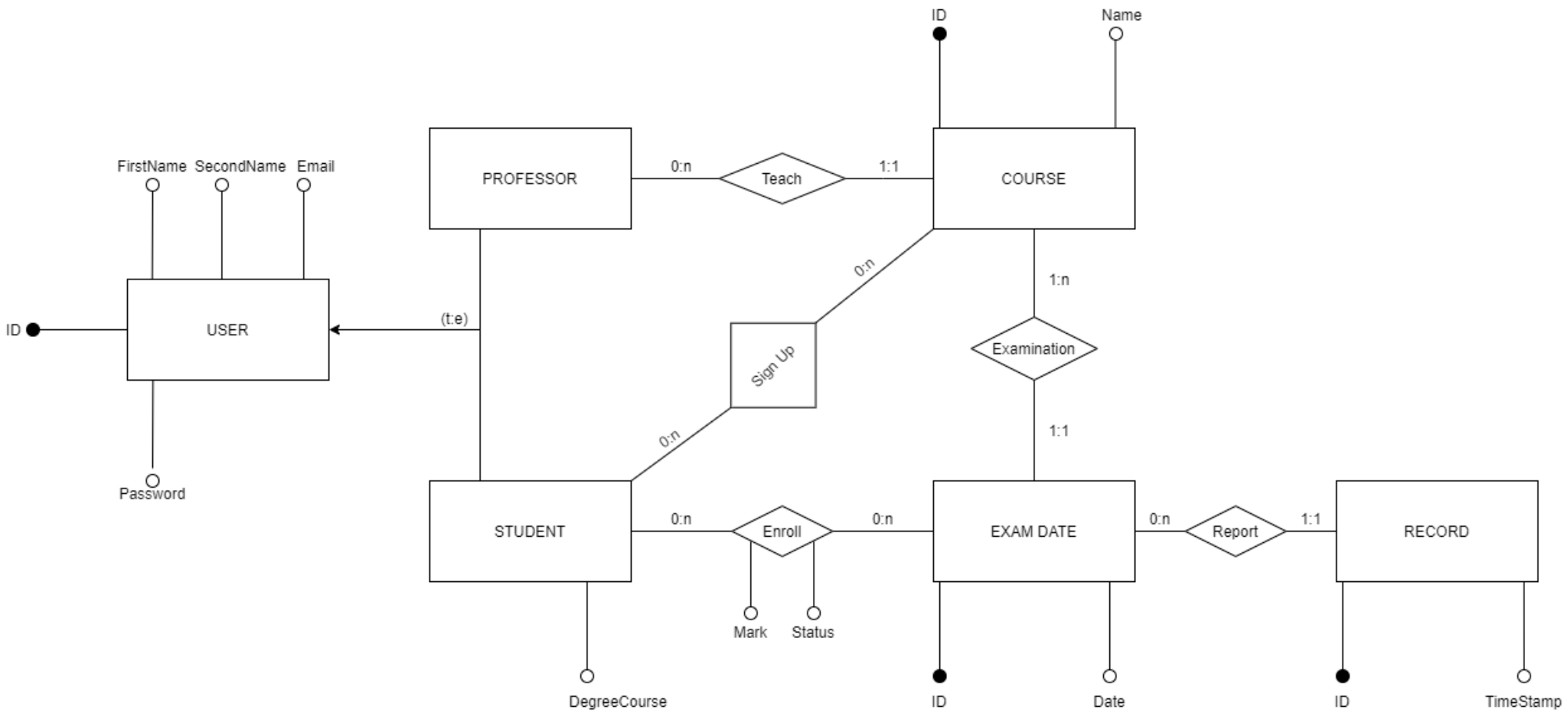
Un’applicazione permette di verbalizzare gli esiti degli esami di un appello. Il docente accede tramite login e seleziona nella HOME page un corso da una lista dei propri corsi ordinata in modo alfabetico decrescente e poi una data d’appello del corso scelto selezionata da un elenco ordinato per data 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**

# Completamento delle specifiche

- 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.

# Server-Side Components

Model Objects (beans):

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

Controllers:

- CheckLogin
- GetCourseDatePro
- GetCourseDateStud
- GetCoursePro
- GetCourseStud
- GetRecordedEnrolls
- GetResultDetails
- GetSessionEnrolls
- LogOut
- RecordScores
- UpdateMultipleScores
- UpdateResultStud
- UpdateScore
- UpdateStatus

Filters

- LoginChecker
- ProfessorChecker
- StudentChecker

Data Access Objects

- CourseDAO
  - `findCoursesByIdProf()`
  - `findCoursesByIdStudent()`
  - `findCourseNameByIdExam()`
- EnrollsDAO
  - `findEnrolls()`
  - `findEnrollsOrderedByIDAsc()`
  - `findStudentScore()`
  - `checkModifiableCondition()`
  - `insertMark()`
  - `refuseScore()`
  - `publishScore()`
  - `recordScore()`
  - `findRecordedStudents()`
  - `assertion_record()`
  - `assertion_published()`
- ExamDateDAO
  - `findExamDateBYCourseForProfessor()`
  - `findExamDateBYCourseForStudent()`
  - `checkExamDateByProf()`
- RecordDAO
  - `writeRecordOnDb()`
  - `getCurrentID()`
  - `getCurrentTimestamp()`
- UserDAO
  - `checkCredentials()`
  - `findProfessorByIdCourse()`

# Client-Side Components (Professor)

Login (index):

- Login form
  - Manages submit and errors

HomePro:

- PageOrchestrator:
  - Start() : initializes page components
  - Refresh() : updates the page hiding examDates list and sessionEnrolls and showing only courseList
- PersonalMessage:
  - Show() : shows custom personal message according to user's id and name-surname
- CourseList:
  - Reset(): hides the courseList table
  - Show(): shows the courseList getting it from the server via an AJAX GET
  - Update(courselist): builds the updated courseList table
  - Waiter(examdate): when a course is selected, hides all the others and adds the «show all courses» button
- CourseDate:
  - Reset(): hides the courseDate table
  - Show(): shows the courseDate list getting it from the server via an AJAX GET
  - Update(examdateslist): builds the updated courseDate table
  - Waiter(examdate): when an exam date is selected, hides all the others and adds the «show all dates» button
- SessionEnrolls:
  - resetMain(): hides the sessionEnrolls table and all the linked buttons
  - resetModal(): clears the modal page
  - Save(exam_date_id): saves th exam_date clicked by the user
  - Show(): shows the sessionEnrolls table getting them from the server via an AJAX GET, also shows the buttons linked
  - Update(enrolls): builds the updated the sessionEnrolls table and adds the event listener to all clickable buttons
  - recordFunction(e): sends the form containing the exam_date_id to record
  - PublishFunction(e): sends the form containing the exam_date_id to publish
  - isModifiable(status) : checks if an enroll is modifiable
  - appendIfNotInserted(examDate): appends a student to the multiple modify modal page only if he is in NOT_INSERTED status
  - closeButton(): closes the modal page
  - multipleEvent(): shows the multiple modify modal page
  - Update_single_modifier(): updates data in the single modify modal window
  - showRecordedEnrolls(): shows the record in a modal page

# Client-Side Components (Student)

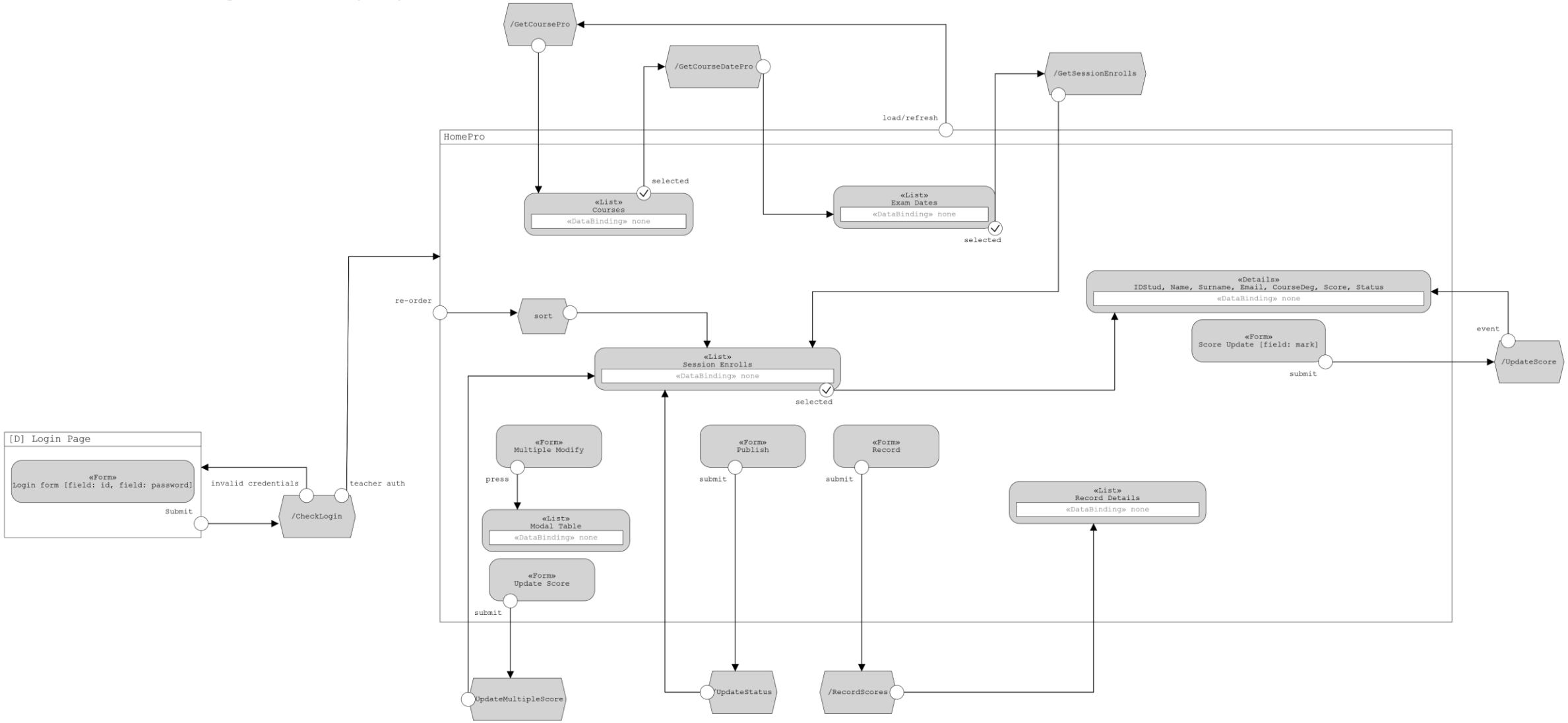
Login (index):

- Login form
  - Manages submit and errors

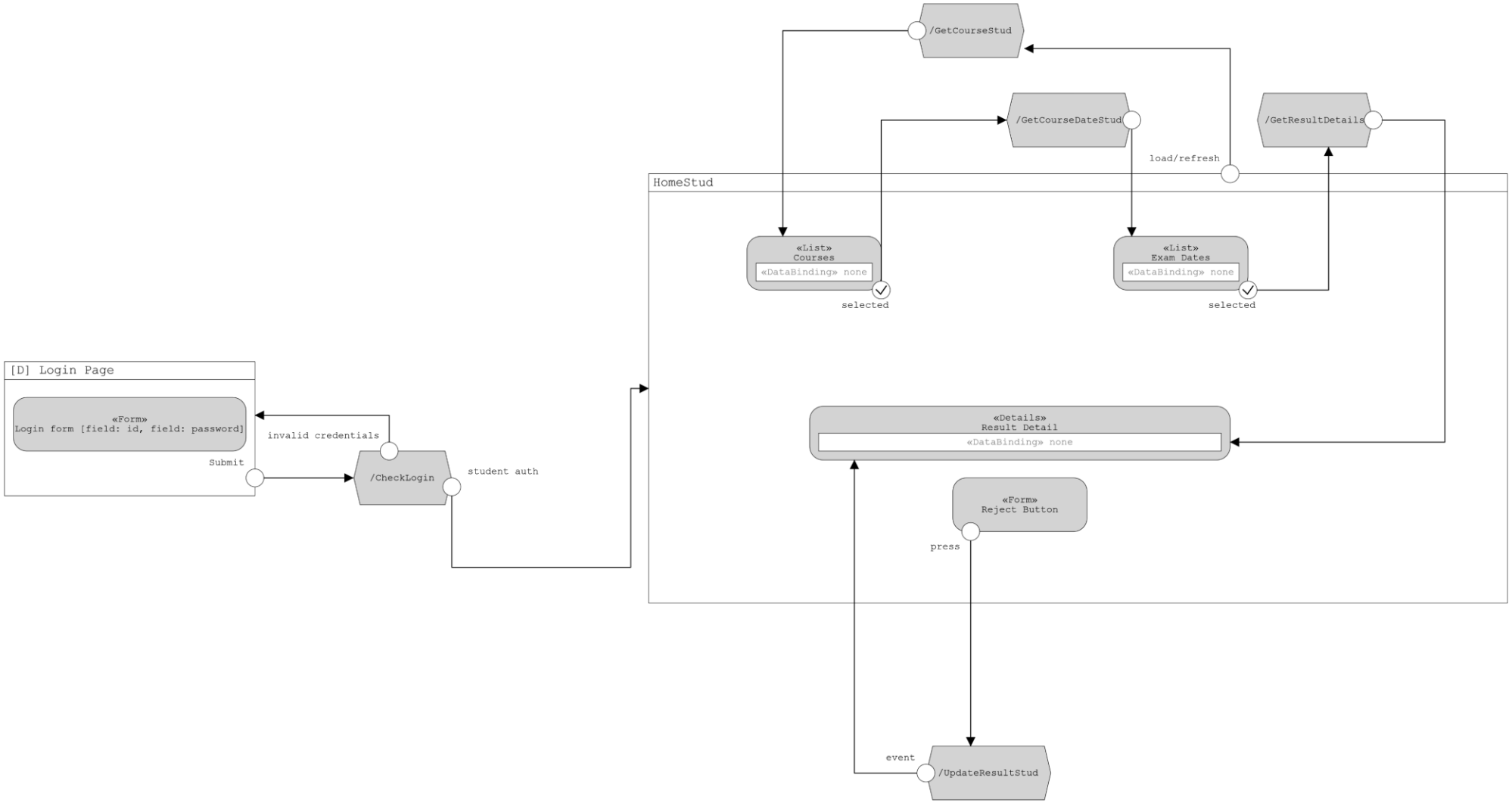
HomeStud:

- PageOrchestrator:
  - Start() : initializes page components
  - Refresh() : updates the page hiding examDates list and resultDetails and showing only courseList
- PersonalMessage:
  - Show() : shows custom personal message according to user's id and name-surname
- CourseList:
  - Reset(): hides the courseList table
  - Show(): shows the courseList getting it from the server via an AJAX GET
  - Update(courselist): builds the updated courseList table
  - Waiter(examdate): when a course is selected, hides all the others and adds the «show all courses» button
- CourseDate:
  - Reset(): hides the courseDate table
  - Show(): shows the courseDate list getting it from the server via an AJAX GET
  - Update(examdateslist): builds the updated courseDate table
  - Waiter(examdate): when an exam date is selected, hides all the others and adds the «show all dates» button
- ResultDetails:
  - reset(): hides the result table and all the linked buttons
  - Show(): shows the resultDetails table getting it from the server via an AJAX GET, also shows the buttons linked
  - Update(result): builds the updated the sessionEnrolls table and adds the refuse button only if the mark is refusable
  - isRefusable(mark, status) : checks if an enroll is modifiable
  - refuseScore(form) : sends the form containing the exam_date_id of the mark to refuse
  - removeColumn(): hides «options» column if the mark is not refusable

# Design Applicativo (IFML Professore)



# Design Applicativo (IFML Studente)



# Eventi & azioni Professore

NB: i controlli di validità dei dati (client e server side) e di autorizzazione (server side) all'accesso sono previsti per tutti gli eventi che li richiedono e non sono riportati nella tabella per brevità

| Client side | | Server side | |
|---|---|---|---|
| **Evento** | **Azione** | **Evento** | **Azione** |
| index → login form → submit | Controllo dati | POST id password | Controllo credenziali |
| HomePagePro→ load | Aggiorna view con lista dei corsi | GET (nessun parametro) | Estrazione corsi del professore |
| HomePagePro → elenco corsi → seleziona corso | Aggiorna view con date esame e riorganizzazione della tabella dei corsi | GET( id corso) | Estrazione date d'esame del corso |
| Home page→ elenco date → seleziona data | Aggiorna view con elenco iscritti all'appello e riorganizzazione della tabella delle date | GET(id esame) | Estrazione iscritti all'appello d'esame |
| HomePage → elenco iscritti al corso → click intestazione tabella | Riordinamento elementi della tabella | - | - |
| ... → iscritti al corso → click bottone modify → click bottone update → submit | Ajax post | POST (id esame, id studente, voto) | Modifica voto studente |
| ...→ iscritti al corso → click bottone modify multipla → click bottone update → submit | Ajax post con oggetto JSON contenente lista degli studenti (MakeCallJSON) | POST (id esame, lista studenti, voto) | Modifica multipla voti studenti |

# Eventi & azioni Professore

NB: i controlli di validità dei dati (client e server side) e di autorizzazione (server side) all'accesso sono previsti per tutti gli eventi che li richiedono e non sono riportati nella tabella per brevità

| Client side | | Server side | |
|---|---|---|---|
| **Evento** | **Azione** | **Evento** | **Azione** |
| ...→ iscritti al corso → click bottone publish → submit | Ajax post | POST(id esame) | Pubblicazione voti |
| ...→ iscritti al corso → click bottone record → submit | Ajax post e comparsa pagina verbale | POST(id esame) | Verbalizzazione esami ed estrazione verbale |
| ...→ iscritti al corso → click bottone record → finestra modale verbale → bottone print | Stampa verbale in pdf | - | - |
| HomePagePro → elenco corsi → click bottone mostra tutti i corsi | Rimozione tabelle iscritti e date e riorganizzazione tabella corsi | - | - |
| HomePagePro → elenco corsi → elenco date → click bottone mostra tutte le date | Rimozione tabelle iscritti e riorganizzazione tabella date | - | - |
| Logout | | GET | Terminazione della sessione |

# Eventi & azioni Studente

NB: i controlli di validità dei dati (client e server side) e di autorizzazione (server side) all'accesso sono previsti per tutti gli eventi che li richiedono e non sono riportati nella tabella per brevità

| Client side | | Server side | |
|---|---|---|---|
| **Evento** | **Azione** | **Evento** | **Azione** |
| index → login form → submit | Controllo dati | POST id password | Controllo credenziali |
| HomePageStud→ load | Aggiorna view con lista dei corsi | GET (nessun parametro) | Estrazione corsi a cui lo studente è iscritto |
| HomePageStud → elenco corsi → seleziona corso | Aggiorna view con date esame e riorganizzazione della tabella dei corsi | GET( id corso) | Estrazione date d'esame del corso |
| HomePageStud → elenco date → seleziona data | Aggiorna view con dettagli esito esame | GET(id esame) | Estrazione dettagli esito esame |
| ... → seleziona data → dettagli esame → click bottone rifiuta | Ajax post | POST(id esame) | Rifiuto del voto |
| HomePageStud → elenco corsi → click bottone mostra tutti i corsi | Rimozione tabelle dettagli esame e date e riorganizzazione tabella corsi | - | - |
| HomePageStud→ elenco corsi → elenco date → click bottone mostra tutte le date | Rimozione tabella dettagli esame e riorganizzazione tabella date | - | - |
| Logout | | GET | Terminazione della sessione |

# Controller / event handler Professore

| Client side | | Server side | |
|---|---|---|---|
| **Evento** | **Controllore** | **Evento** | **Controllore** |
| index → login form → submit | Funzione MakeCall | POST id password | CheckLogin (servlet) |
| HomePagePro→ load | Funzione pageOrchestrator | GET (nessun parametro) | GetCoursePro (servlet) |
| HomePagePro → elenco corsi → seleziona corso | Funzione courseDate.show()<br>Funzione courseList.waiter() | GET( id corso) | GetCourseDatePro (servlet) |
| Home page→ elenco date → seleziona data | Funzione sessionEnrolls.show()<br>Funzione courseDate.waiter() | GET(id esame) | GetSessionEnrolls (servlet) |
| HomePage → elenco iscritti al corso → click intestazione tabella | Funzione sort | - | - |
| ... → iscritti al corso → click bottone modify → click bottone update → submit | Funzione MakeCall | POST (id esame, id studente, voto) | UpdateScore (servlet) |
| ...→ iscritti al corso → click bottone modify multipla → click bottone update → submit | Funzione MakeCallJSON | POST (id esame, lista studenti, voto) | UpdateMultipleScore (servlet) |

# Controller / event handler Professore

| Client side | | Server side | |
|---|---|---|---|
| **Evento** | **Controllore** | **Evento** | **Controllore** |
| ...→ iscritti al corso → click bottone publish → submit | Funzione MakeCall | POST(id esame) | UpdateStatus |
| ...→ iscritti al corso → click bottone record → submit | Funzione MakeCall | POST(id esame) | RecordScores |
| ...→ iscritti al corso → click bottone record → finestra modale verbale → bottone print | HTML script (printDiv) | - | - |
| HomePagePro → elenco corsi → click bottone mostra tutti i corsi | Event handler bottone mostra tutti i corsi | - | - |
| HomePagePro → elenco corsi → elenco date → click bottone mostra tutte le date | Event handler bottone mostra tutte le date | - | - |
| Logout | | GET | Logout (servlet) |

# Controller / event handler Studente

| Client side | | Server side | |
|---|---|---|---|
| **Evento** | **Controllore** | **Evento** | **Controllore** |
| index → login form → submit | Funzione MakeCall | POST id password | CheckLogin (servlet) |
| HomePageStud→ load | Funzione PageOrchestrator | GET (nessun parametro) | GetCourseStud (servlet) |
| HomePageStud → elenco corsi → seleziona corso | Funzione courseDate.show()<br>Funzione courseList.waiter() | GET( id corso) | GetCourseDateStud (servlet) |
| HomePageStud → elenco date → seleziona data | Funzione resultDetails.show()<br>Funzione courseDate.waiter() | GET(id esame) | GetResultDetails (servlet) |
| ... → seleziona data → dettagli esame → click bottone rifiuta | Funzione MakeCall | POST(id esame) | UpdateResultStud (servlet) |
| HomePageStud → elenco corsi → click bottone mostra tutti i corsi | Event handler bottone mostra tutti i corsi | - | - |
| HomePageStud→ elenco corsi → elenco date → click bottone mostra tutte le date | Event handler bottone mostra tutte le date | - | - |
| Logout | | GET | Terminazione della sessione |

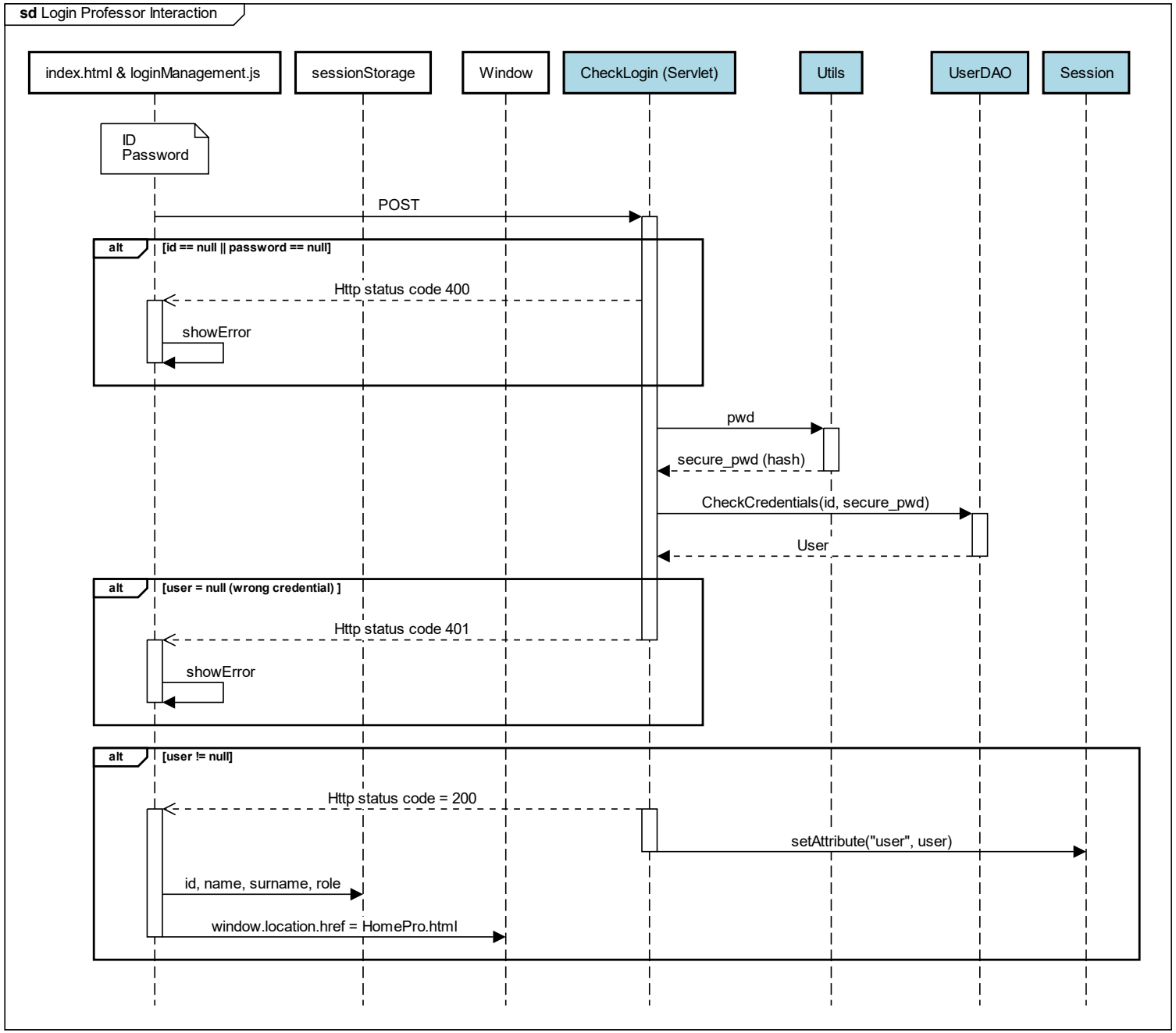
# Sequence Diagrams

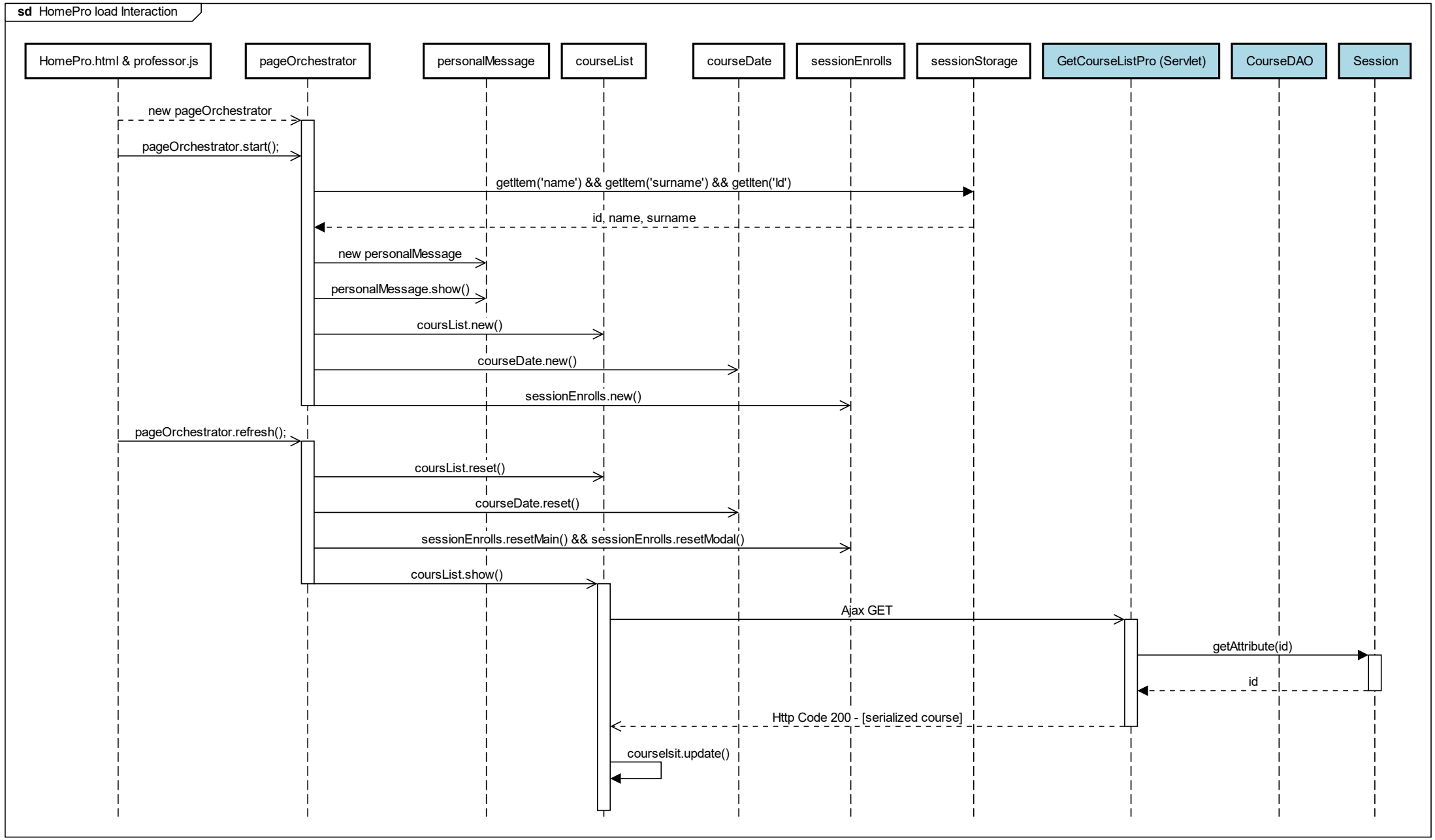
The following slides shows the sequence diagram of each possible interaction. Some details have been omitted to avoid repetitiveness (e.g. internal server errors, database access errors, validity parameters check).

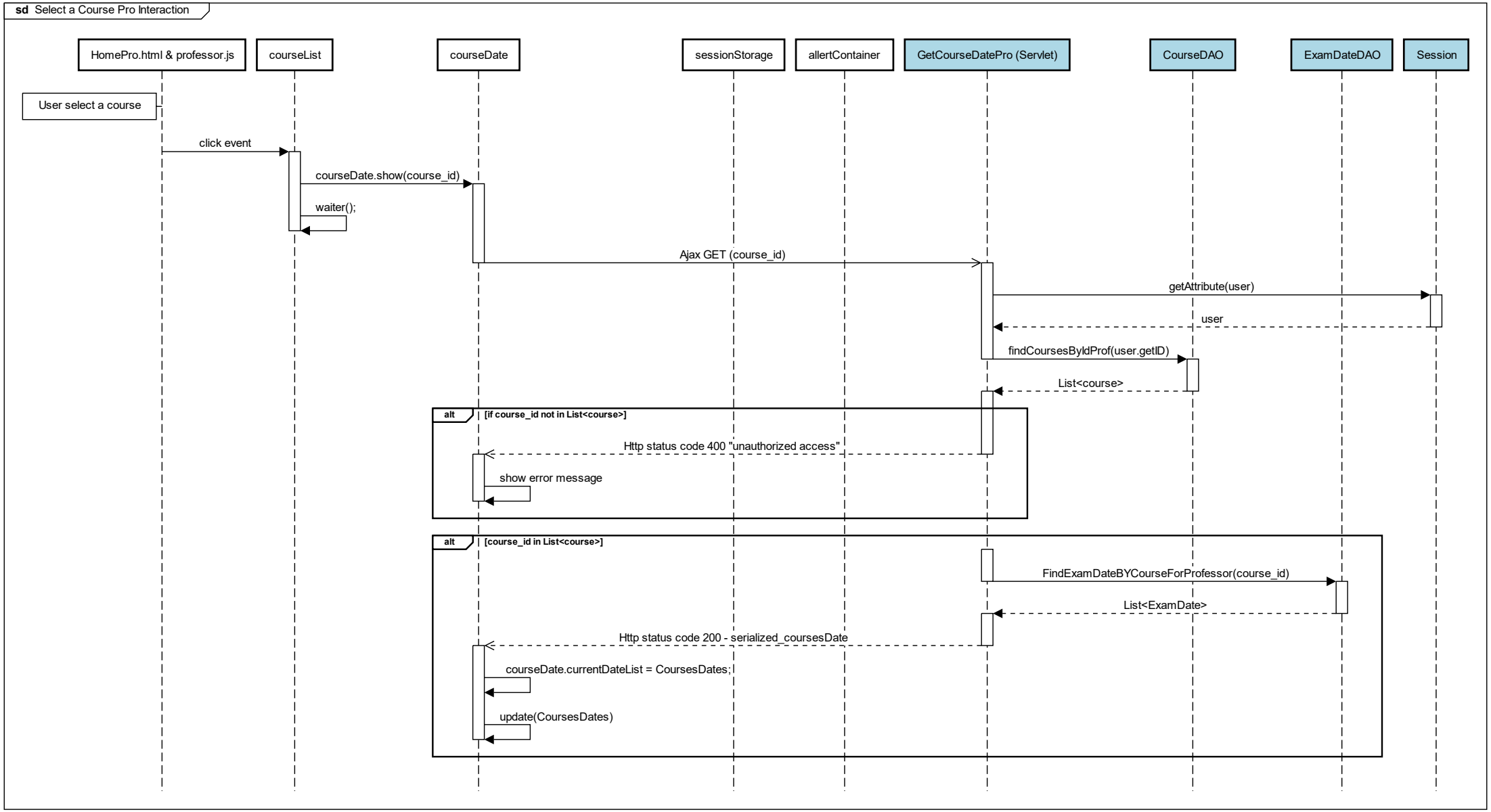
Filters checks have been omitted in this version because they behave as in HTML pure version. See the html pure documentation for further details on filters behaviour.

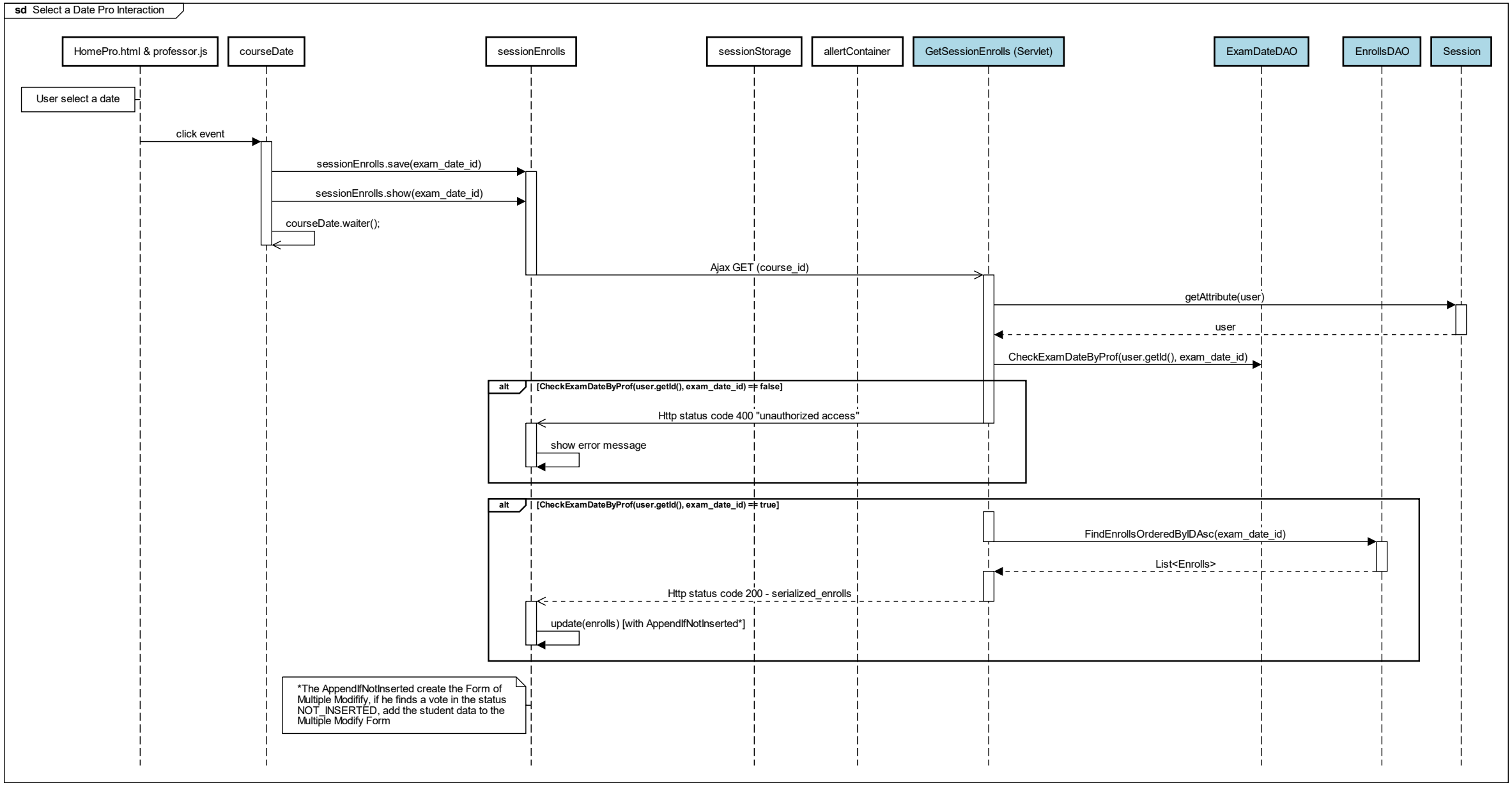
Blue lifelines indicate server-side components, while white lifelines indicate client-side components.

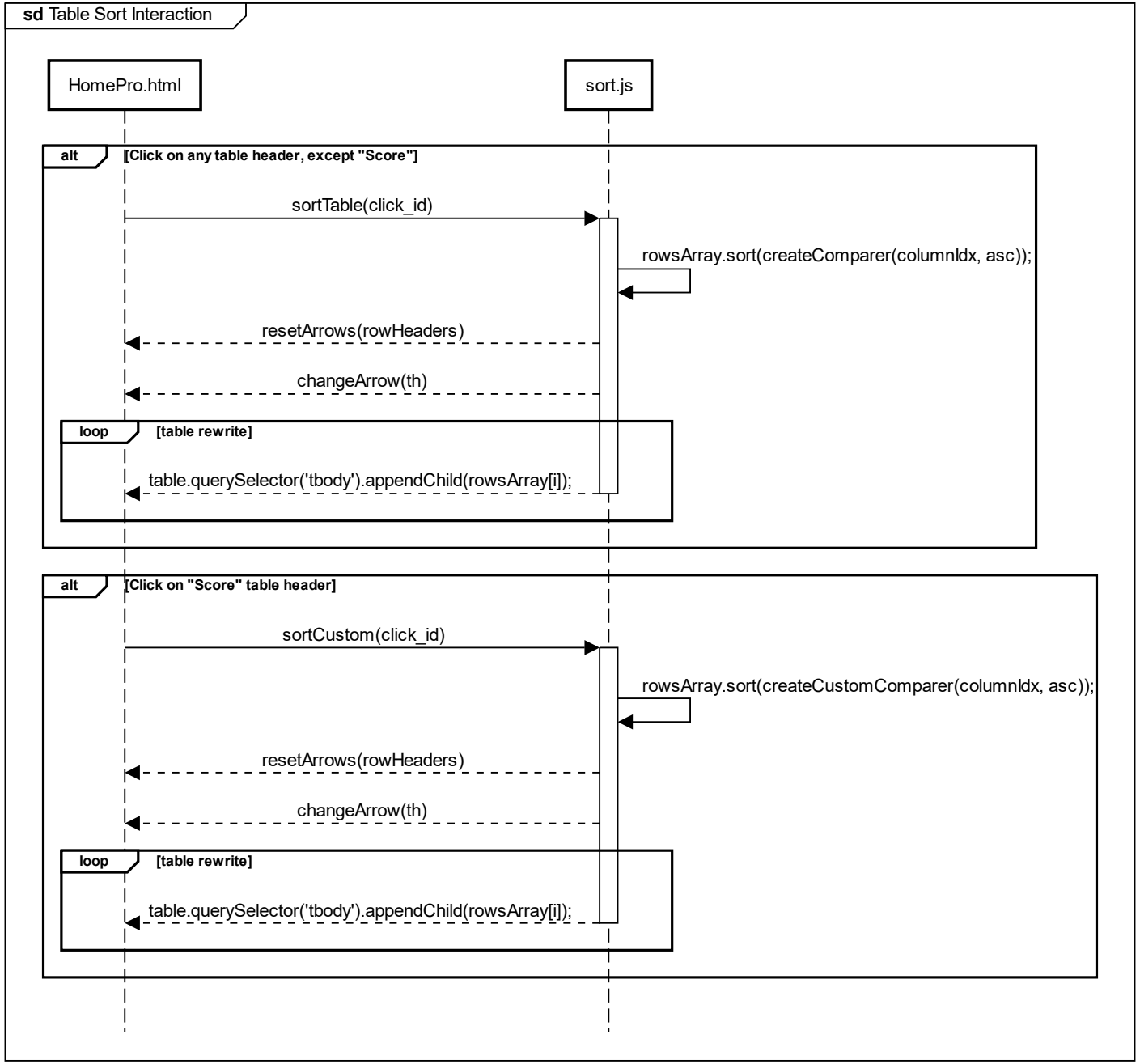
# Professor

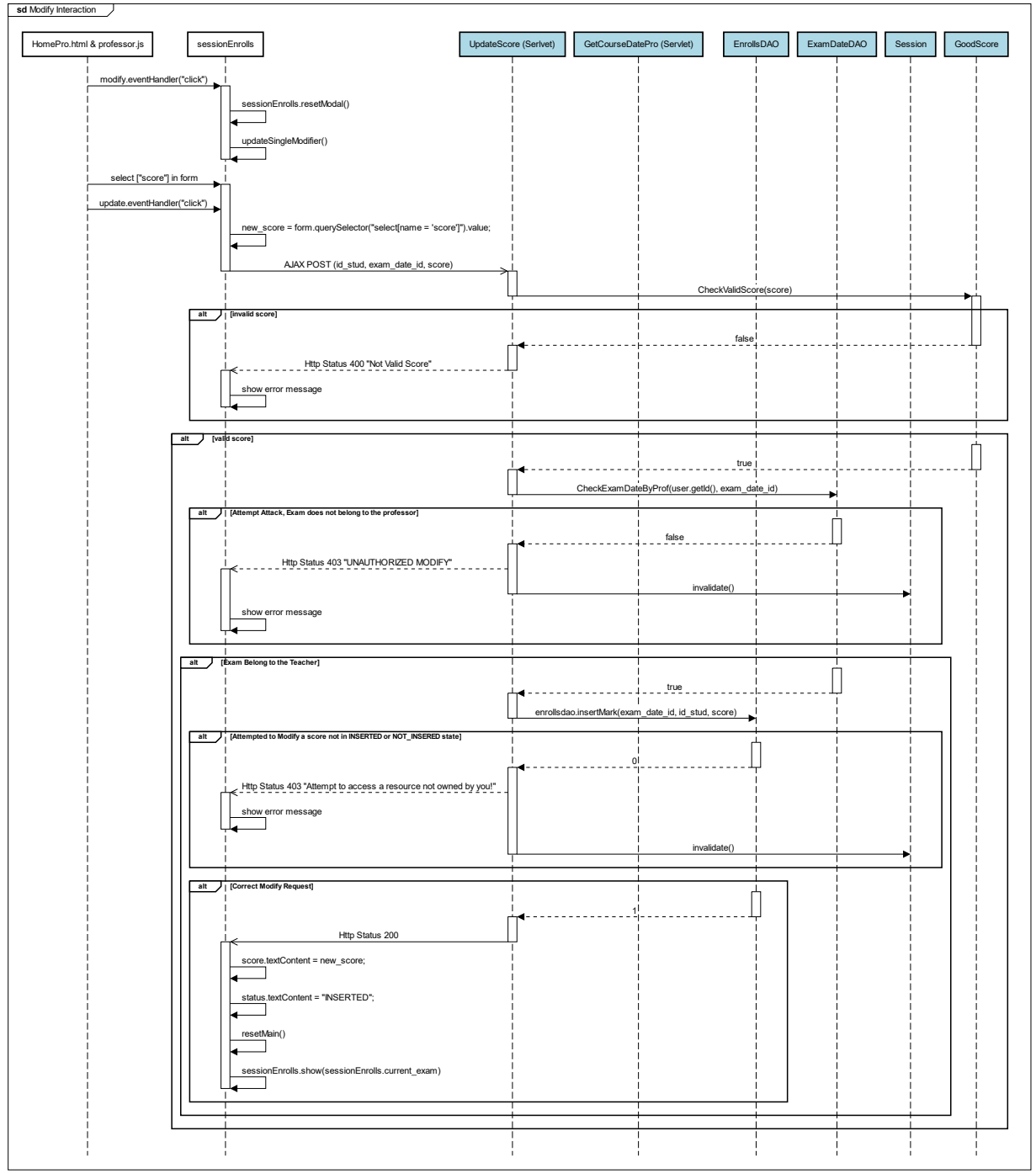

sd Login Professor Interaction
index.html & loginManagement.js
sessionStorage
Window
CheckLogin (Servlet)
Utils
UserDAO
Session
ID
Password
POST
alt
[id == null || password == null]
Http status code 400
showError
pwd
secure_pwd (hash)
CheckCredentials(id, secure_pwd)
User
alt
[user = null (wrong credential) ]
Http status code 401
showError
alt
[user != null]
Http status code = 200
setAttribute("user", user)
id, name, surname, role
window.location.href = HomePro.html

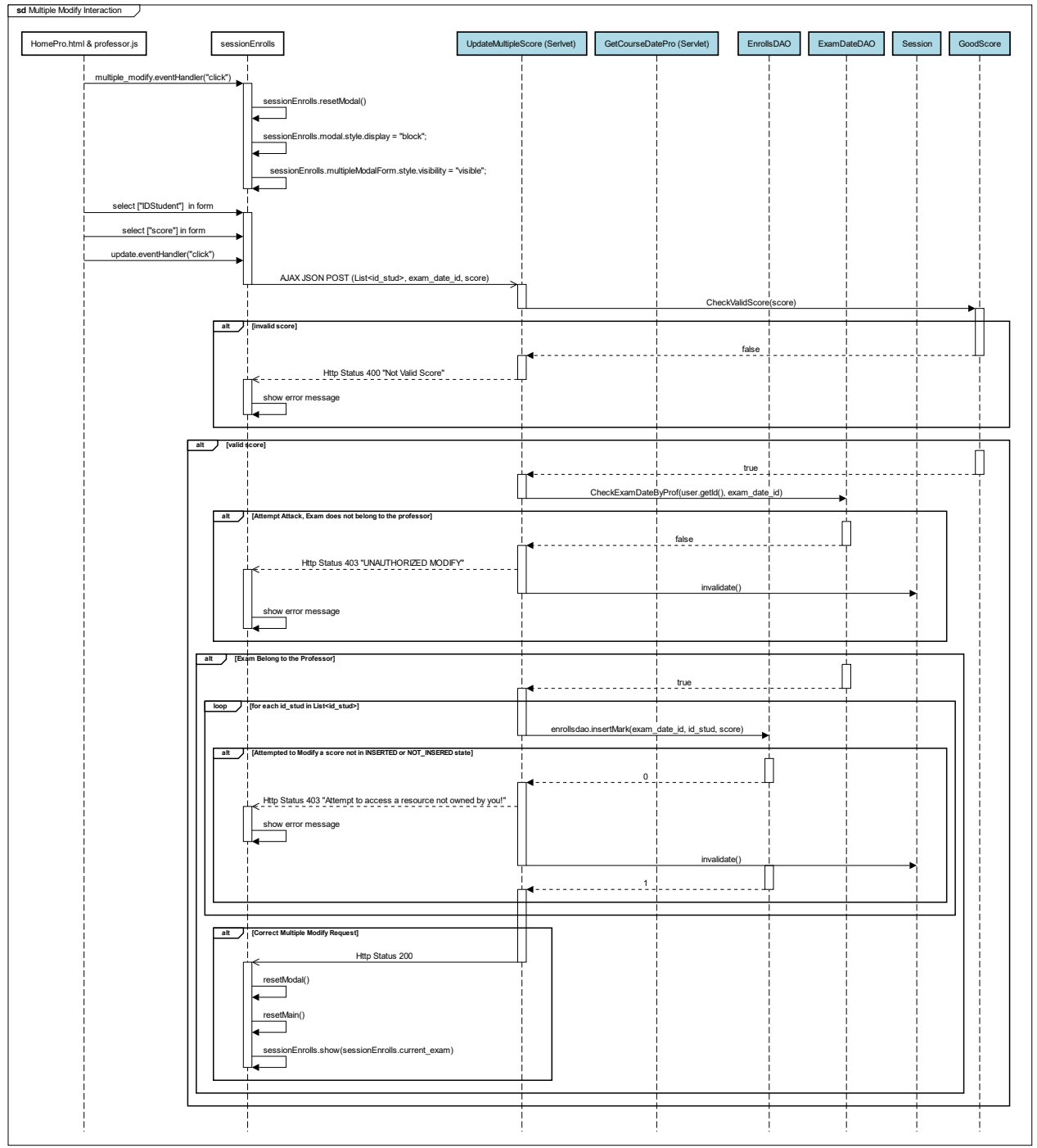

sd HomePro load Interaction
HomePro.html & professor.js
pageOrchestrator
personalMessage
courseList
courseDate
sessionEnrolls
sessionStorage
GetCourseListPro (Servlet)
CourseDAO
Session
new pageOrchestrator
pageOrchestrator.start();
getItem('name') && getItem('surname') && getIten('Id')
id, name, surname
new personalMessage
personalMessage.show()
coursList.new()
courseDate.new()
sessionEnrolls.new()
pageOrchestrator.refresh();
coursList.reset()
courseDate.reset()
sessionEnrolls.resetMain() && sessionEnrolls.resetModal()
coursList.show()
Ajax GET
getAttribute(id)
id
Http Code 200 - [serialized course]
courselsit.update()

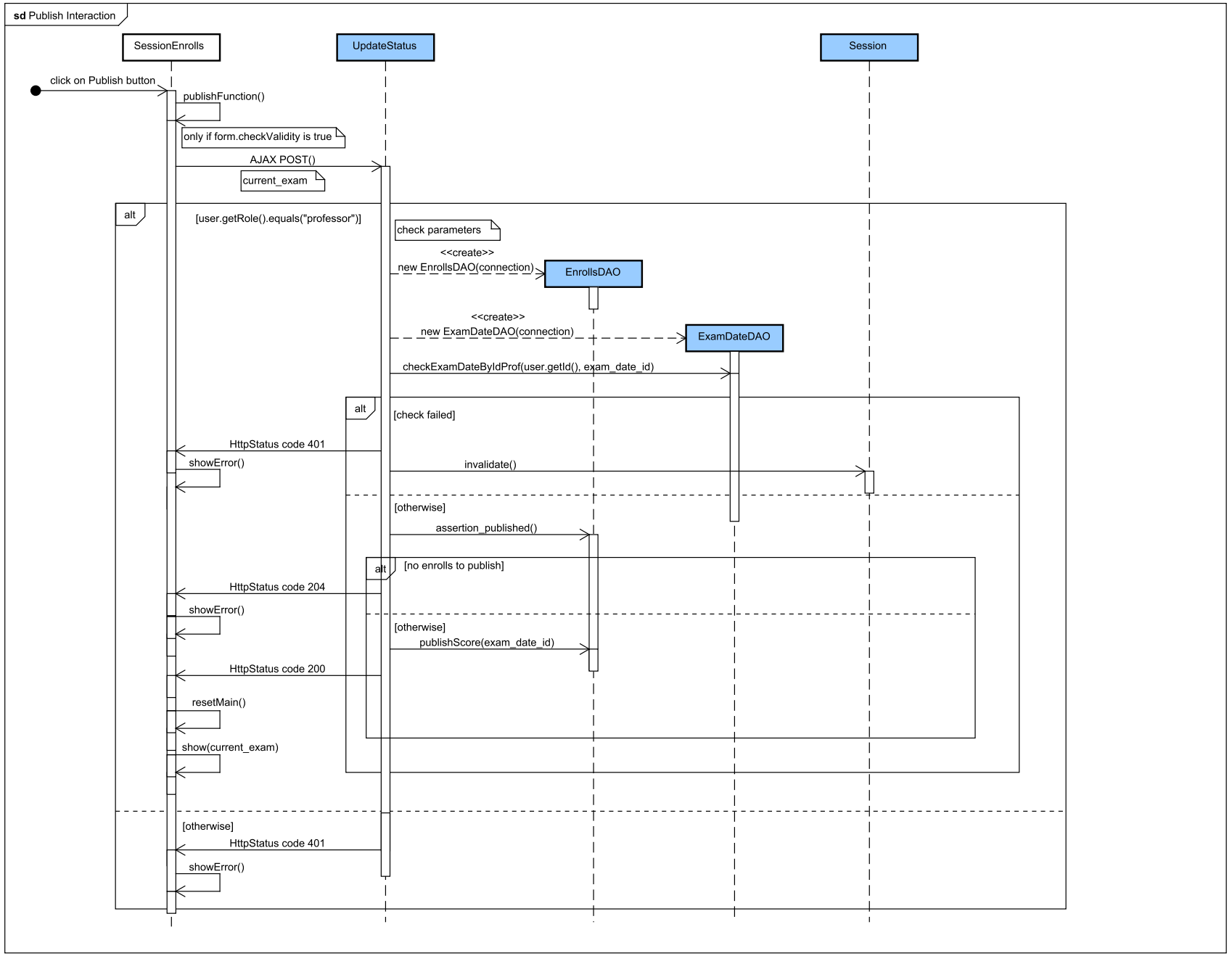

sd Select a Course Pro Interaction
HomePro.html & professor.js
courseList
courseDate
sessionStorage
allertContainer
GetCourseDatePro (Servlet)
CourseDAO
ExamDateDAO
Session
User select a course
click event
courseDate.show(course_id)
waiter();
Ajax GET (course_id)
getAttribute(user)
user
findCoursesByIdProf(user.getID)
List<course>
alt
[if course_id not in List<course>]
Http status code 400 "unauthorized access"
show error message
alt
[course_id in List<course>]
FindExamDateBYCourseForProfessor(course_id)
List<ExamDate>
Http status code 200 - serialized_coursesDate
courseDate.currentDateList = CoursesDates;
update(CoursesDates)

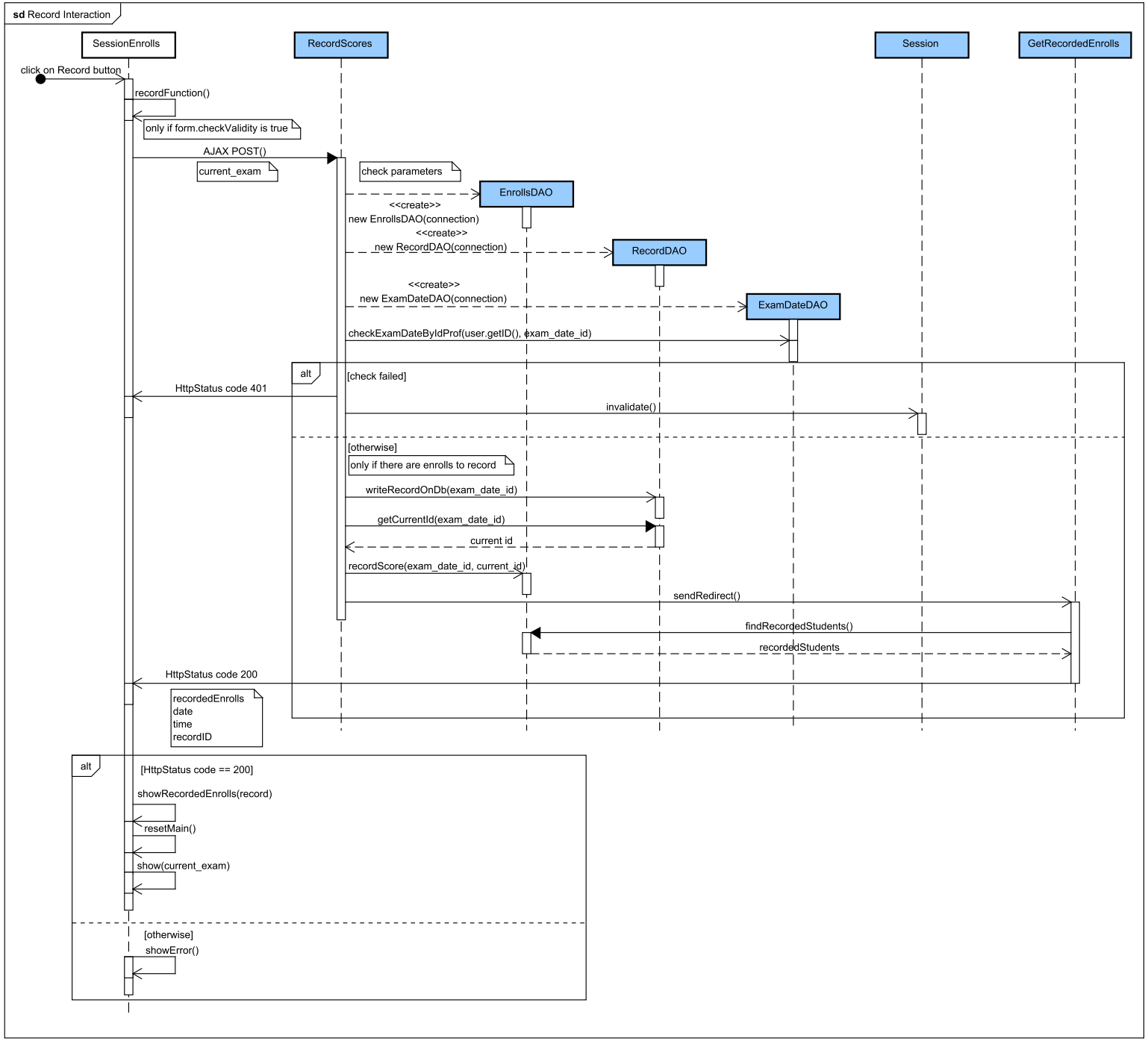

sd Select a Date Pro Interaction
HomePro.html & professor.js
courseDate
sessionEnrolls
sessionStorage
allertContainer
GetSessionEnrolls (Servlet)
ExamDateDAO
EnrollsDAO
Session
User select a date
click event
sessionEnrolls.save(exam_date_id)
sessionEnrolls.show(exam_date_id)
courseDate.waiter();
Ajax GET (course_id)
getAttribute(user)
user
CheckExamDateByProf(user.getId(), exam_date_id)
alt
[CheckExamDateByProf(user.getId(), exam_date_id) == false]
Http status code 400 "unauthorized access"
show error message
alt
[CheckExamDateByProf(user.getId(), exam_date_id) == true]
FindEnrollsOrderedByIDAsc(exam_date_id)
List<Enrolls>
Http status code 200 - serialized_enrolls
update(enrolls) [with AppendIfNotInserted*]
*The AppendIfNotInserted create the Form of Multiple Modifify, if he finds a vote in the status NOT_INSERTED, add the student data to the Multiple Modify Form

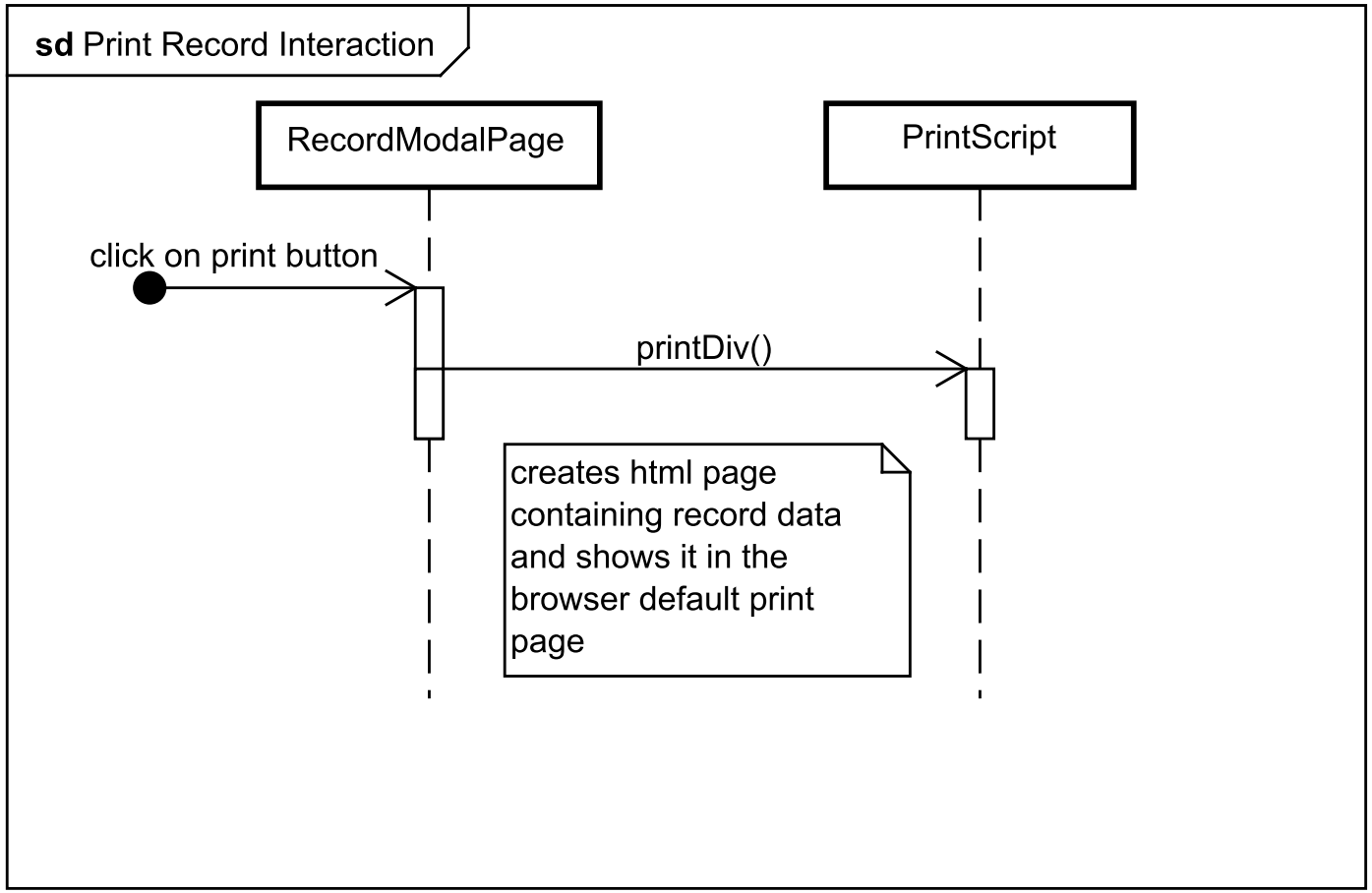

sd Table Sort Interaction
HomePro.html
sort.js
alt
[Click on any table header, except "Score"]
sortTable(click_id)
rowsArray.sort(createComparer(columnIdx, asc));
resetArrows(rowHeaders)
changeArrow(th)
loop
[table rewrite]
table.querySelector('tbody').appendChild(rowsArray[i]);
alt
[Click on "Score" table header]
sortCustom(click_id)
rowsArray.sort(createCustomComparer(columnIdx, asc));
resetArrows(rowHeaders)
changeArrow(th)
loop
[table rewrite]
table.querySelector('tbody').appendChild(rowsArray[i]);

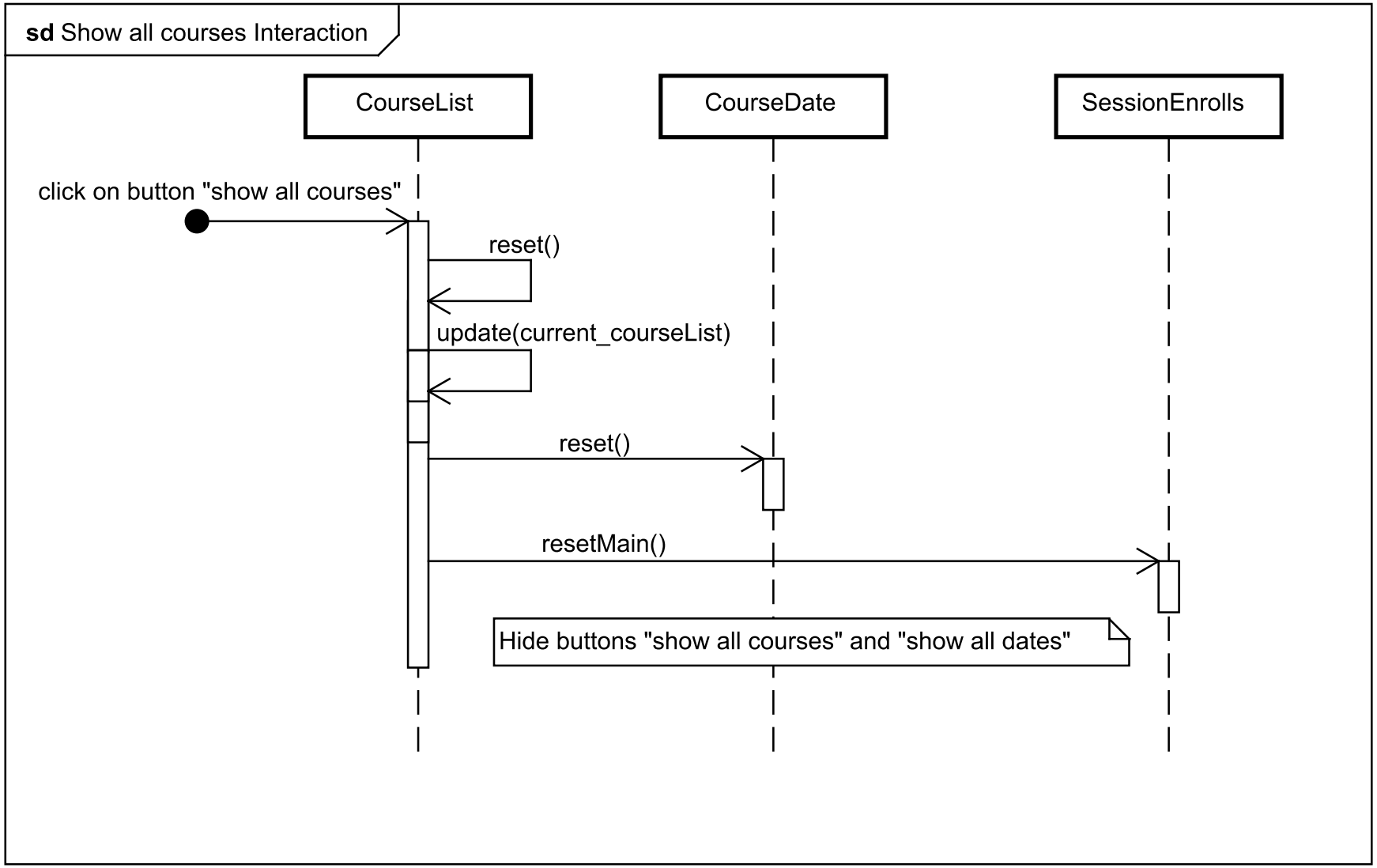

sd Modify Interaction
HomePro.html & professor.js
sessionEnrolls
UpdateScore (Serlvet)
GetCourseDatePro (Servlet)
EnrollsDAO
ExamDateDAO
Session
GoodScore
modify.eventHandler("click")
sessionEnrolls.resetModal()
updateSingleModifier()
select ["score"] in form
update.eventHandler("click")
new_score = form.querySelector("select[name = 'score']").value;
AJAX POST (id_stud, exam_date_id, score)
CheckValidScore(score)
alt
[invalid score]
false
Http Status 400 "Not Valid Score"
show error message
alt
[valid score]
true
CheckExamDateByProf(user.getId(), exam_date_id)
alt
[Attempt Attack, Exam does not belong to the professor]
false
Http Status 403 "UNAUTHORIZED MODIFY"
invalidate()
show error message
alt
[Exam Belong to the Teacher]
true
enrollsdao.insertMark(exam_date_id, id_stud, score)
alt
[Attempted to Modify a score not in INSERTED or NOT_INSERED state]
0
Http Status 403 "Attempt to access a resource not owned by you!"
show error message
invalidate()
alt
[Correct Modify Request]
1
Http Status 200
score.textContent = new_score;
status.textContent = "INSERTED";
resetMain()
sessionEnrolls.show(sessionEnrolls.current_exam)

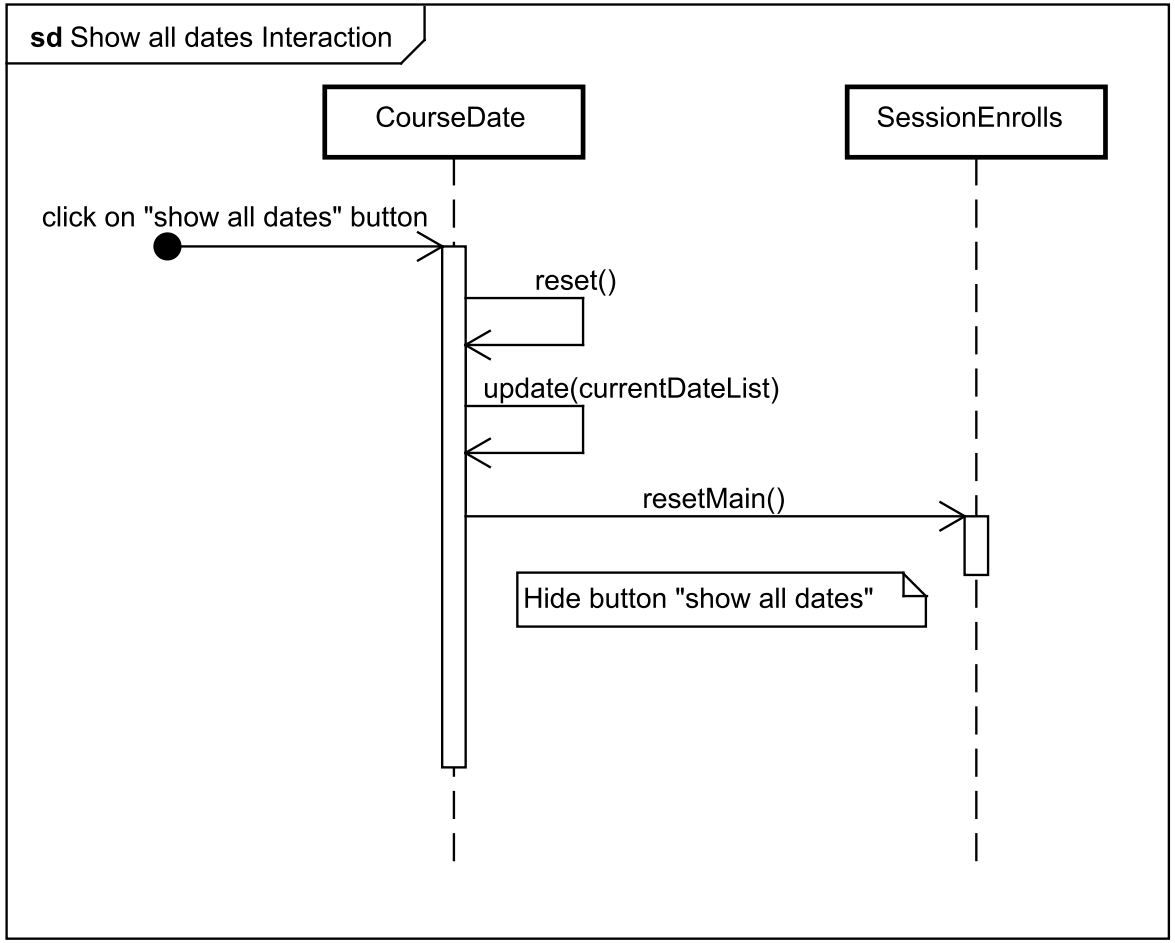

sd Multiple Modify Interaction
HomePro.html & professor.js
sessionEnrolls
UpdateMultipleScore (Servlet)
GetCourseDatePro (Servlet)
EnrollsDAO
ExamDateDAO
Session
GoodScore
multiple_modify.eventHandler("click")
sessionEnrolls.resetModal()
sessionEnrolls.modal.style.display = "block";
sessionEnrolls.multipleModalForm.style.visibility = "visible";
select ["IDStudent"] in form
select ["score"] in form
update.eventHandler("click")
AJAX JSON POST (List<id_stud>, exam_date_id, score)
CheckValidScore(score)
alt
[invalid score]
false
Http Status 400 "Not Valid Score"
show error message
alt
[valid score]
true
CheckExamDateByProf(user.getId(), exam_date_id)
alt
[Attempt Attack, Exam does not belong to the professor]
false
Http Status 403 "UNAUTHORIZED MODIFY"
invalidate()
show error message
alt
[Exam Belong to the Professor]
true
loop
[for each id_stud in List<id_stud>]
enrollsdao.insertMark(exam_date_id, id_stud, score)
alt
[Attempted to Modify a score not in INSERTED or NOT_INSERED state]
0
Http Status 403 "Attempt to access a resource not owned by you!"
show error message
invalidate()
1
alt
[Correct Multiple Modify Request]
Http Status 200
resetModal()
resetMain()
sessionEnrolls.show(sessionEnrolls.current_exam)

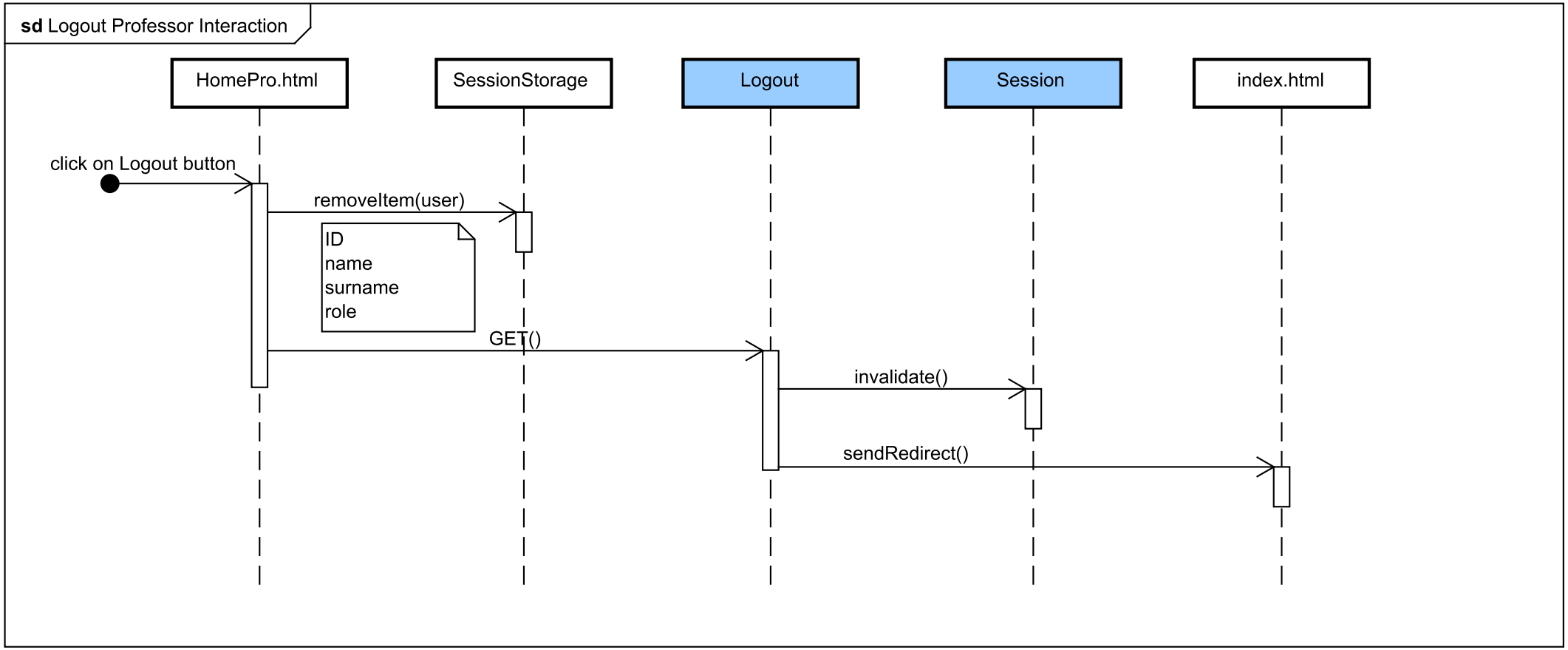

sd Publish Interaction
SessionEnrolls
UpdateStatus
Session
click on Publish button
publishFunction()
only if form.checkValidity is true
AJAX POST()
current_exam
alt
[user.getRole().equals("professor")]
check parameters
<<create>>
new EnrollsDAO(connection)
EnrollsDAO
<<create>>
new ExamDateDAO(connection)
ExamDateDAO
checkExamDateByIdProf(user.getId(), exam_date_id)
alt
[check failed]
HttpStatus code 401
showError()
invalidate()
[otherwise]
assertion_published()
alt
[no enrolls to publish]
HttpStatus code 204
showError()
[otherwise]
publishScore(exam_date_id)
HttpStatus code 200
resetMain()
show(current_exam)
[otherwise]
HttpStatus code 401
showError()


sd Record Interaction
SessionEnrolls
RecordScores
Session
GetRecordedEnrolls
click on Record button
recordFunction()
only if form.checkValidity is true
AJAX POST()
current_exam
check parameters
EnrollsDAO
<<create>>
new EnrollsDAO(connection)
<<create>>
new RecordDAO(connection)
RecordDAO
<<create>>
new ExamDateDAO(connection)
ExamDateDAO
checkExamDateByIdProf(user.getID(), exam_date_id)
alt
[check failed]
HttpStatus code 401
invalidate()
[otherwise]
only if there are enrolls to record
writeRecordOnDb(exam_date_id)
getCurrentId(exam_date_id)
current id
recordScore(exam_date_id, current_id)
sendRedirect()
findRecordedStudents()
recordedStudents
HttpStatus code 200
recordedEnrolls
date
time
recordID
alt
[HttpStatus code == 200]
showRecordedEnrolls(record)
resetMain()
show(current_exam)
[otherwise]
showError()

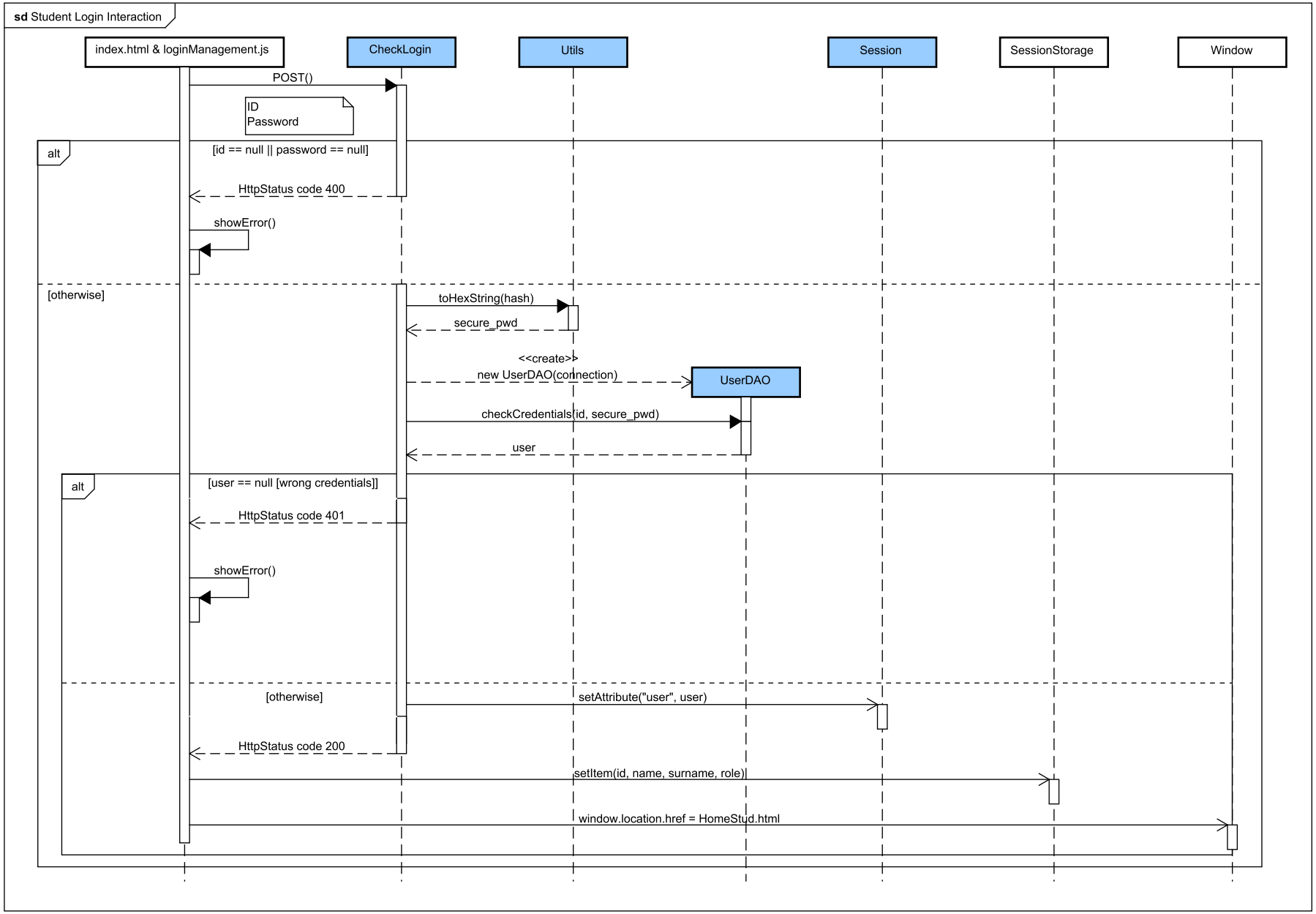

sd Print Record Interaction
RecordModalPage
PrintScript
click on print button
printDiv()
creates html page containing record data and shows it in the browser default print page

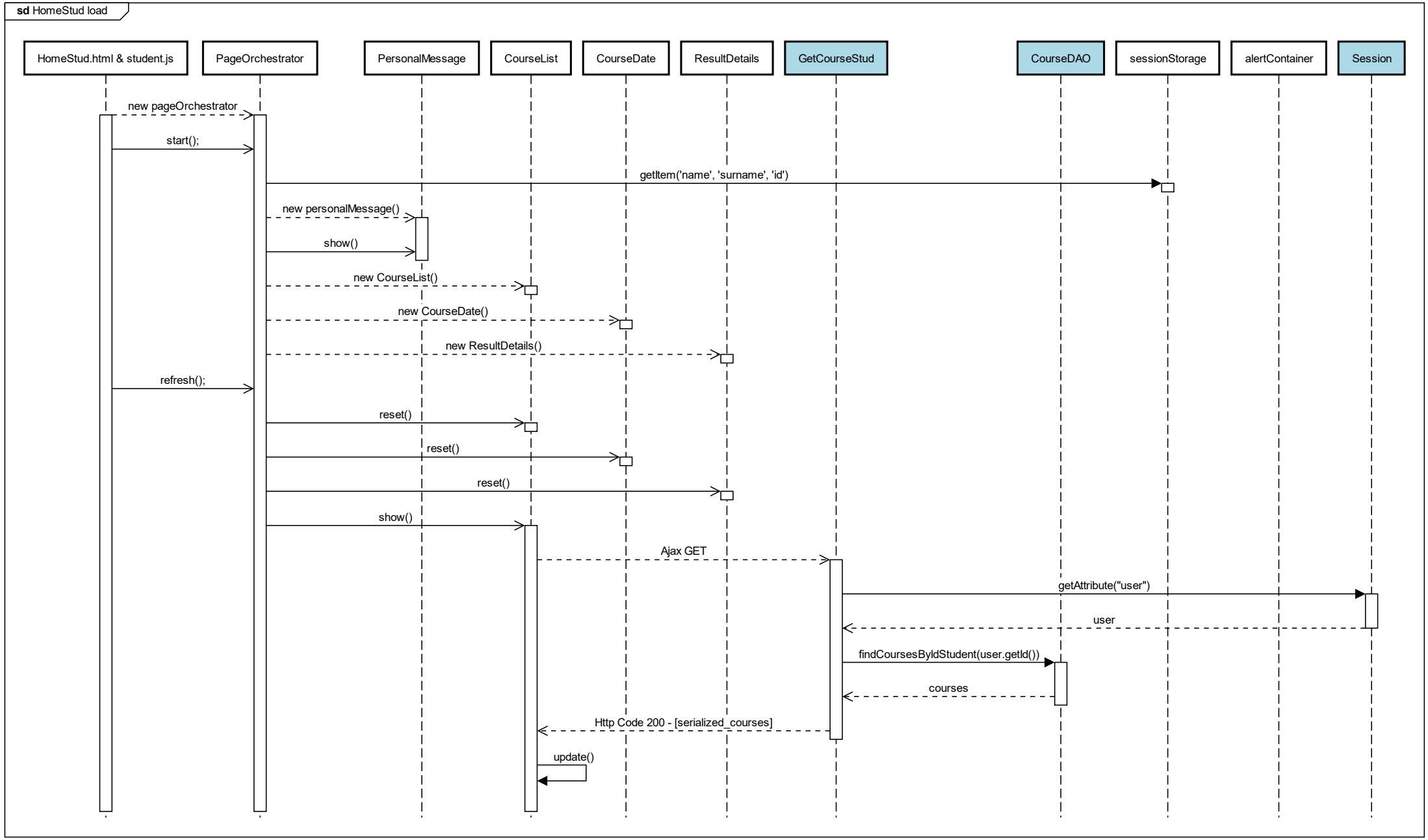

sd Show all courses Interaction
CourseList
CourseDate
SessionEnrolls
click on button "show all courses"
reset()
update(current_courseList)
reset()
resetMain()
Hide buttons "show all courses" and "show all dates"

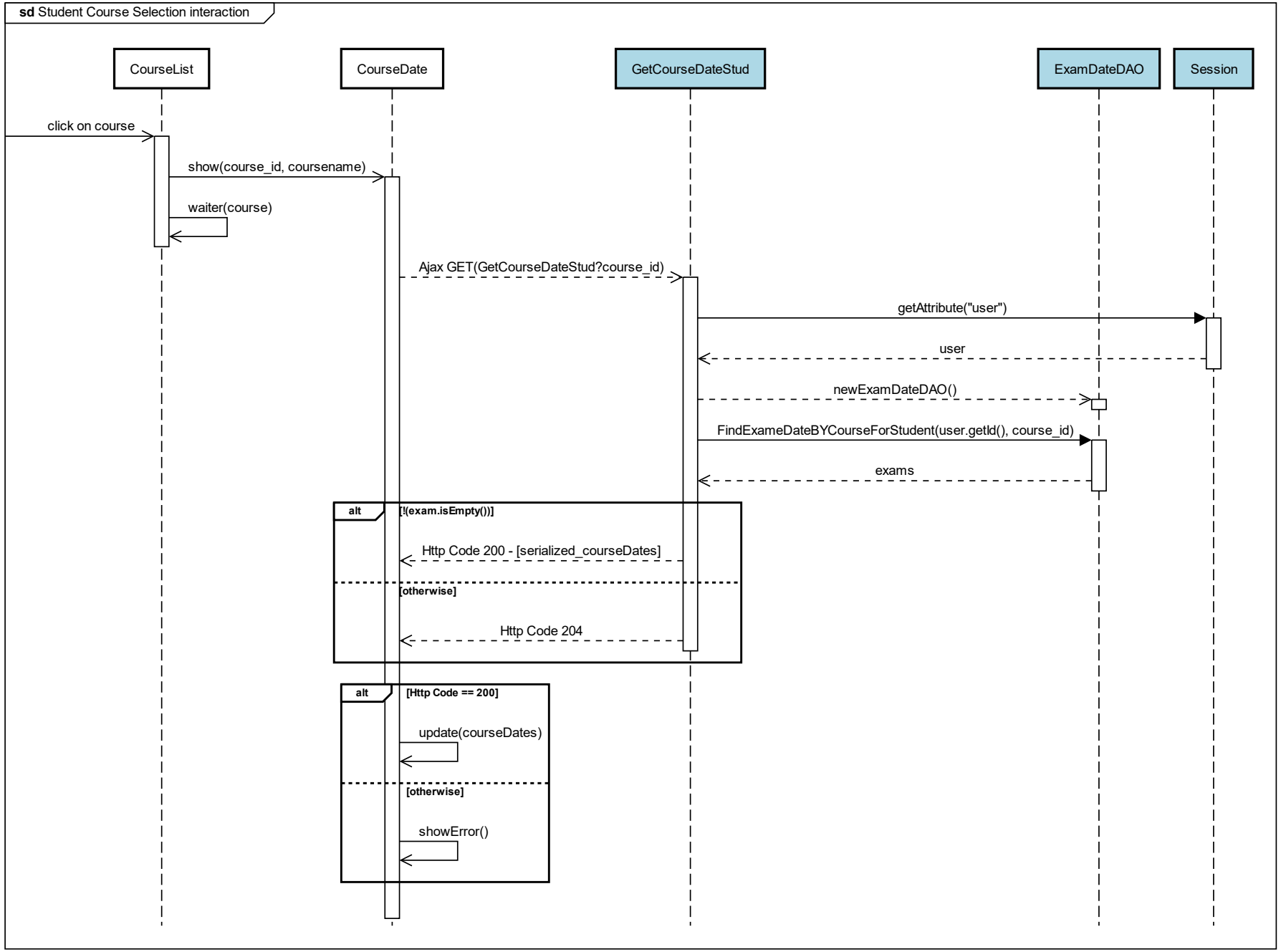

sd Show all dates Interaction
CourseDate
SessionEnrolls
click on "show all dates" button
reset()
update(currentDateList)
resetMain()
Hide button "show all dates"

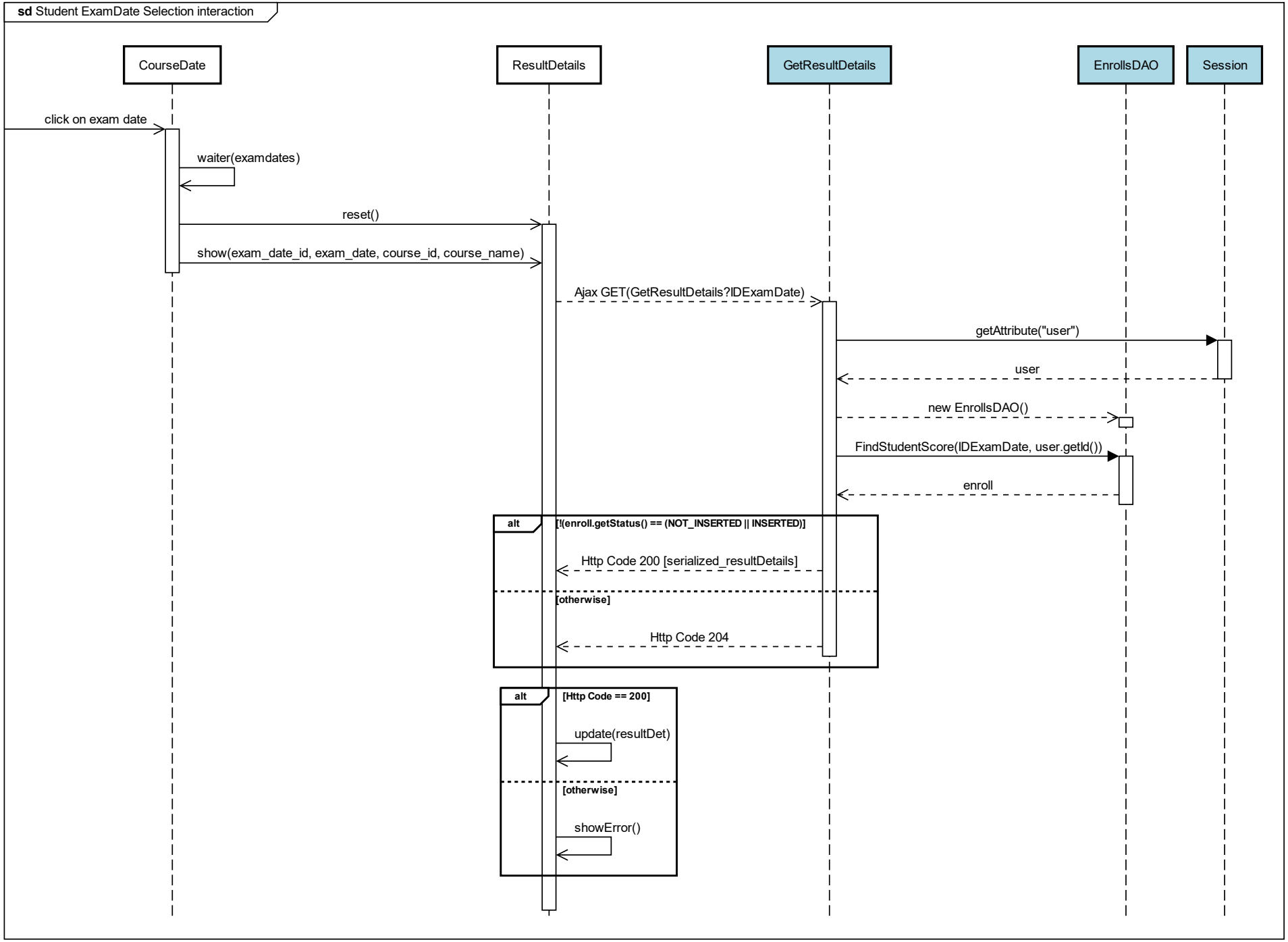

sd Logout Professor Interaction
HomePro.html
SessionStorage
Logout
Session
index.html
click on Logout button
removeItem(user)
ID
name
surname
role
GET()
invalidate()
sendRedirect()

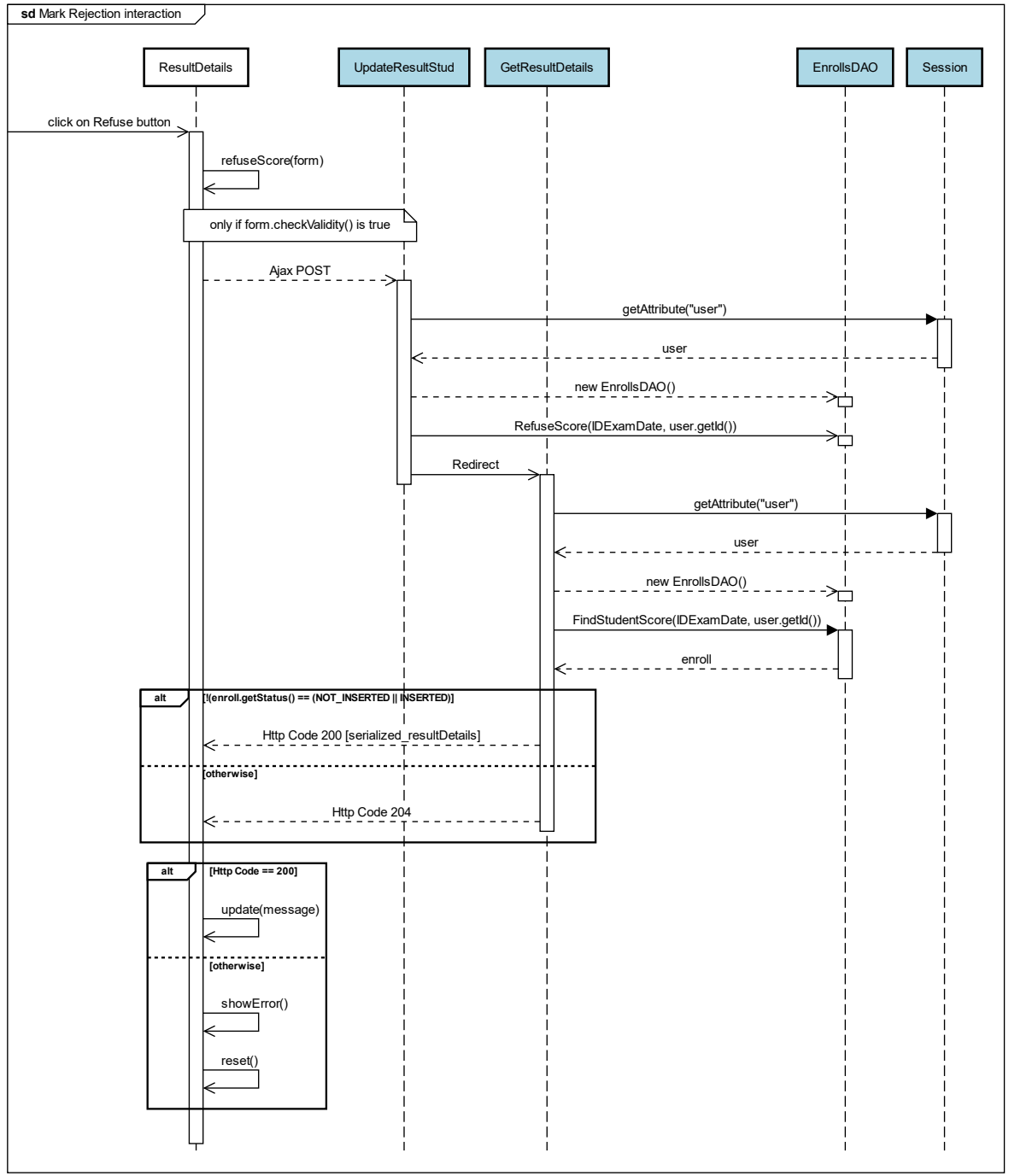
# Student

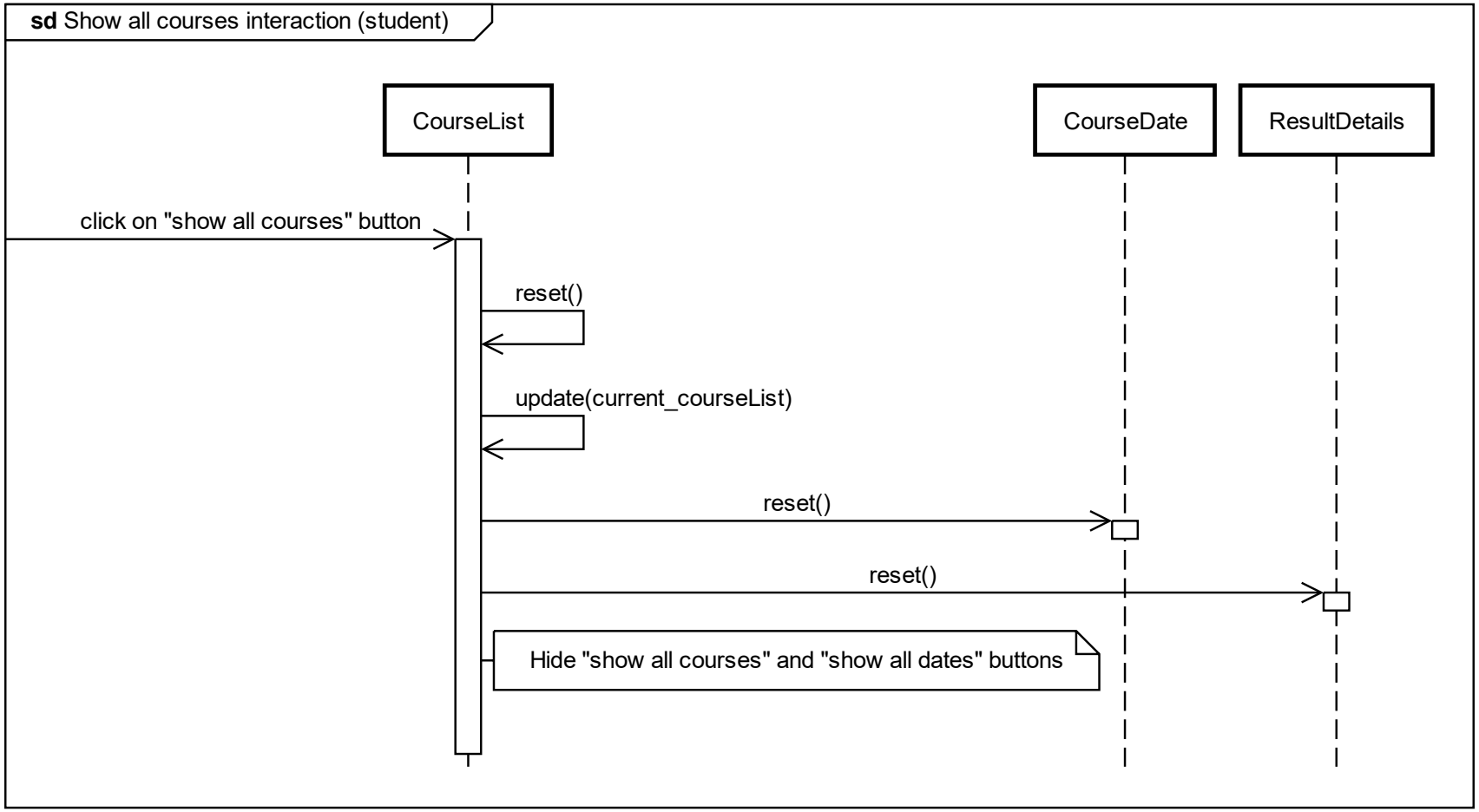

sd Student Login Interaction
index.html & loginManagement.js
CheckLogin
Utils
Session
SessionStorage
Window
POST()
ID
Password
alt
[id == null || password == null]
HttpStatus code 400
showError()
[otherwise]
toHexString(hash)
secure_pwd
<<create>>
new UserDAO(connection)
UserDAO
checkCredentials(id, secure_pwd)
user
alt
[user == null [wrong credentials]]
HttpStatus code 401
showError()
[otherwise]
setAttribute("user", user)
HttpStatus code 200
setItem(id, name, surname, role)
window.location.href = HomeStud.html

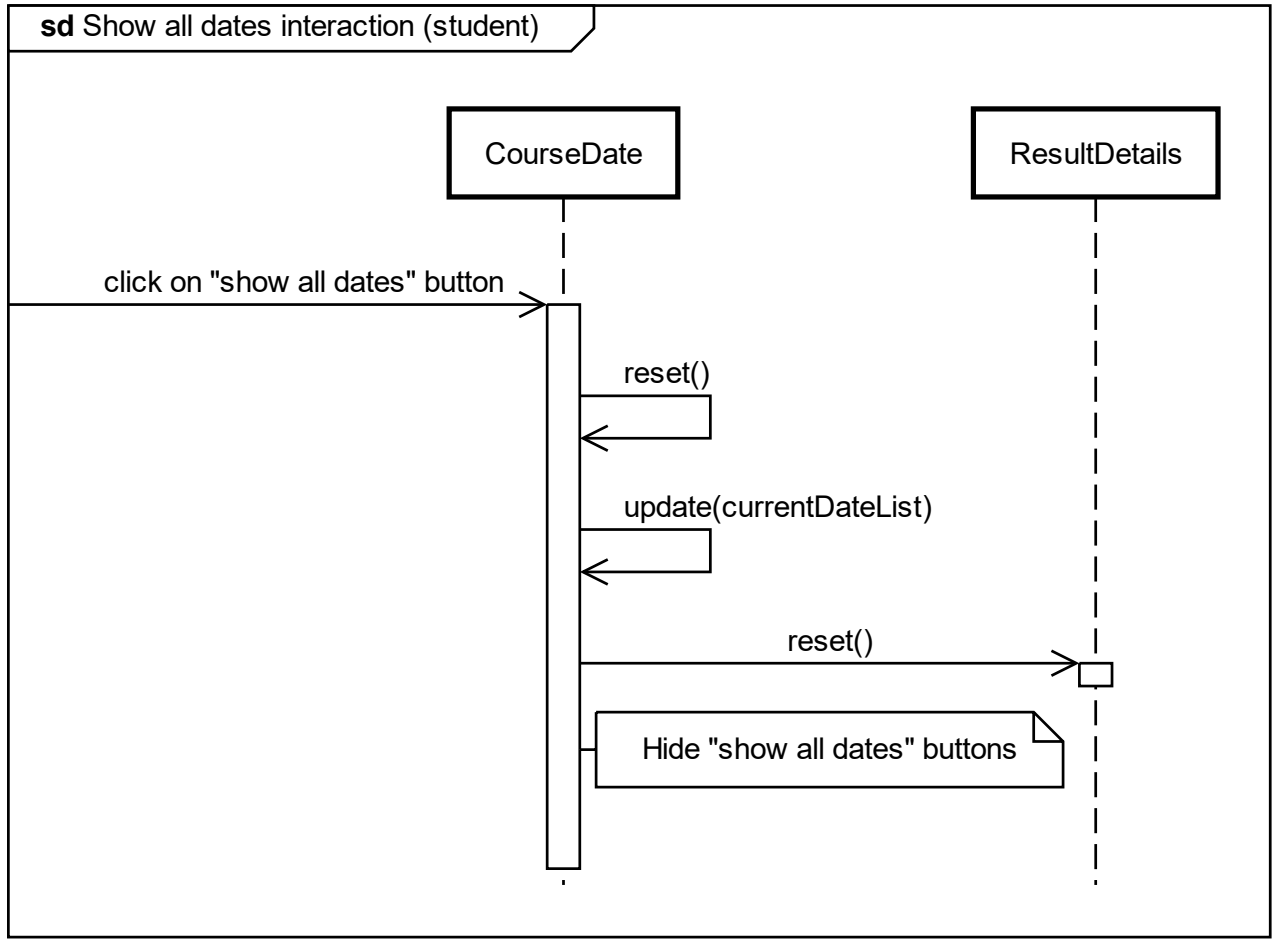

sd HomeStud load
HomeStud.html & student.js
PageOrchestrator
PersonalMessage
CourseList
CourseDate
ResultDetails
GetCourseStud
CourseDAO
sessionStorage
alertContainer
Session
new pageOrchestrator
start();
getItem('name', 'surname', 'id')
new personalMessage()
show()
new CourseList()
new CourseDate()
new ResultDetails()
refresh();
reset()
reset()
reset()
show()
Ajax GET
getAttribute("user")
user
findCoursesByIdStudent(user.getId())
courses
Http Code 200 - [serialized_courses]
update()

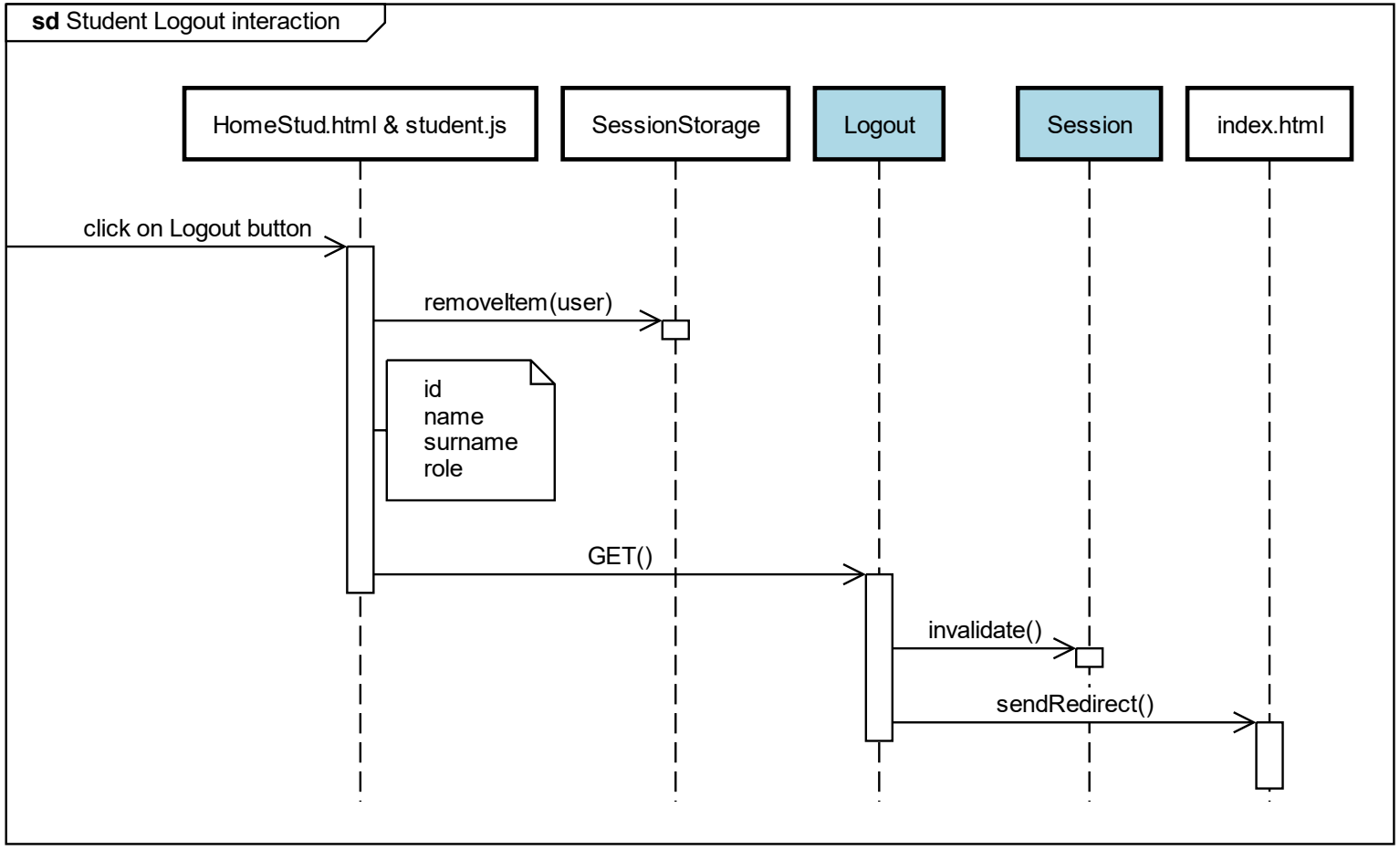

sd Student Course Selection interaction
CourseList
CourseDate
GetCourseDateStud
ExamDateDAO
Session
click on course
show(course_id, coursename)
waiter(course)
Ajax GET(GetCourseDateStud?course_id)
getAttribute("user")
user
newExamDateDAO()
FindExameDateBYCourseForStudent(user.getId(), course_id)
exams
alt
[!(exam.isEmpty())]
Http Code 200 - [serialized_courseDates]
[otherwise]
Http Code 204
alt
[Http Code == 200]
update(courseDates)
[otherwise]
showError()


sd Student ExamDate Selection interaction
CourseDate
ResultDetails
GetResultDetails
EnrollsDAO
Session
click on exam date
waiter(examdates)
reset()
show(exam_date_id, exam_date, course_id, course_name)
Ajax GET(GetResultDetails?IDExamDate)
getAttribute("user")
user
new EnrollsDAO()
FindStudentScore(IDExamDate, user.getId())
enroll
alt
[!(enroll.getStatus() == (NOT_INSERTED || INSERTED)]
Http Code 200 [serialized_resultDetails]
[otherwise]
Http Code 204
alt
[Http Code == 200]
update(resultDet)
[otherwise]
showError()


sd Mark Rejection interaction
ResultDetails
UpdateResultStud
GetResultDetails
EnrollsDAO
Session
click on Refuse button
refuseScore(form)
only if form.checkValidity() is true
Ajax POST
getAttribute("user")
user
new EnrollsDAO()
RefuseScore(IDExamDate, user.getId())
Redirect
getAttribute("user")
user
new EnrollsDAO()
FindStudentScore(IDExamDate, user.getId())
enroll
alt
![enroll.getStatus() == (NOT_INSERTED || INSERTED)]
Http Code 200 [serialized_resultDetails]
[otherwise]
Http Code 204
alt
[Http Code == 200]
update(message)
[otherwise]
showError()
reset()


sd Show all courses interaction (student)
CourseList
CourseDate
ResultDetails
click on "show all courses" button
reset()
update(current_courseList)
reset()
reset()
Hide "show all courses" and "show all dates" buttons


sd Show all dates interaction (student)
CourseDate
ResultDetails
click on "show all dates" button
reset()
update(currentDateList)
reset()
Hide "show all dates" buttons


sd Student Logout interaction
HomeStud.html & student.js
SessionStorage
Logout
Session
index.html
click on Logout button
removeItem(user)
id
name
surname
role
GET()
invalidate()
sendRedirect()