Università degli Studi di Verona

DIPARTIMENTO DI INFORMATICA
Corso di Laurea in Informatica

Consegna G4 - Sistema informativo per la gestione dell'emissione di biglietti aerei

Relazione finale del progetto di Basi di Dati

Candidati:
Giovanni Liboni
Matricola VR363021

Omar Dabbagh
Matricola VR359092

Anno Accademico 2013-2014

Contents

I Progetto della basi di dati **2**

1 Progetto concettuale **2**

2 Progetto logico **3**

3 Popolamento della base di dati **6**

II Progetto del sito web **7**

4 Progettazione logica **7**

5 Struttura dell'applicazione web **10**

III Scelte progettuali **15**

6 Ipotesi e strategie adotattate **15**

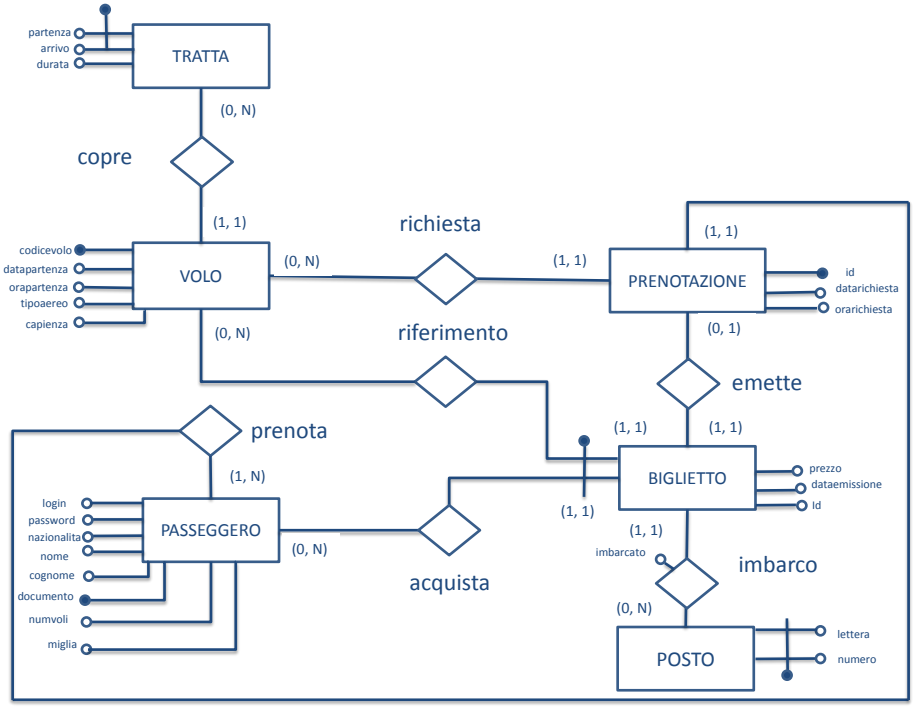
7 Hibernate **16**

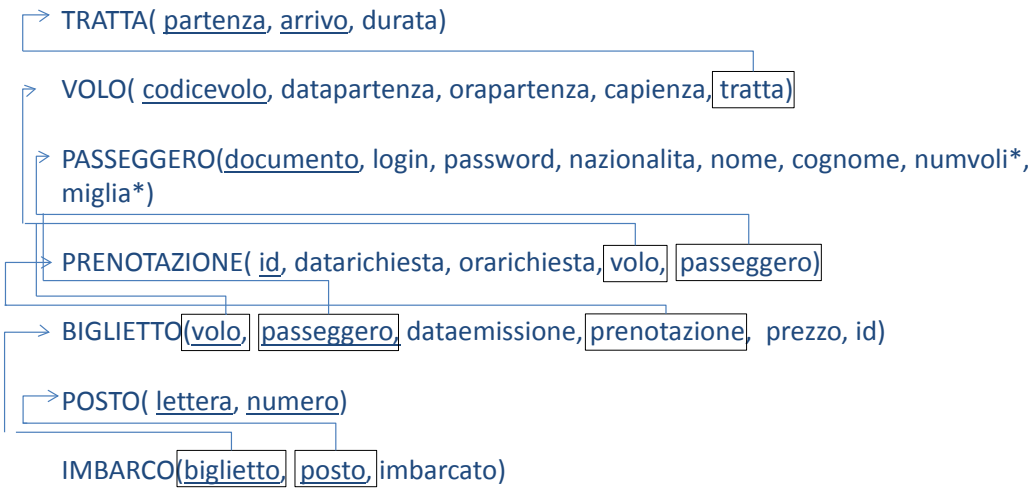
8 AJAX **16**

Part I
Progetto della basi di dati

1 Progetto concettuale

Qui di seguito si riporta il diagramma Entità - Relazione e lo schema concettuale ricavati dalle specifiche del progetto. A partire da questo schema si sono scritte le tabelle per la Base di Dati.





Descrivere il diagramma e lo schema logico.

2 Progetto logico

Abbiamo usato un singolo script per la creazione delle tabelle sul database. Il codice viene riportato qui di seguito.

```
CREATE  TABLE tratta(
  partenza  VARCHAR(100) NOT NULL,
  arrivo    VARCHAR(100) NOT NULL,
  durata    INTEGER NOT NULL,
  distanza  NUMERIC(10,1) NOT NULL DEFAULT '0.0',
  PRIMARY KEY ( partenza, arrivo ),
  UNIQUE ( partenza, arrivo )
);
CREATE TABLE volo(
  codicevolo  VARCHAR(10) NOT NULL,
  partenza  VARCHAR(100) NOT NULL,
  arrivo    VARCHAR(100) NOT NULL,
  datapartenza  DATE NOT NULL,
  orapartenza TIME NOT NULL,
  tipoaereo VARCHAR(50) NOT NULL ,
  capienza  INTEGER NOT NULL,
  PRIMARY KEY( codicevolo ),
  UNIQUE ( codicevolo ),
  FOREIGN KEY( partenza, arrivo )
```

```
      REFERENCES tratta( partenza, arrivo )
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
CREATE TABLE passeggero(
  login    VARCHAR(100) NOT NULL,
  password  VARCHAR(100) NOT NULL,
  nazionalita VARCHAR(100) NOT  NULL,
  nome     VARCHAR(100) NOT NULL,
  cognome    VARCHAR(100) NOT NULL,
  documento VARCHAR(50) NOT NULL,
  picture    BYTEA,

  numvoli    INTEGER DEFAULT 0,
  miglia     FLOAT DEFAULT 0,
  tessera    BOOLEAN DEFAULT FALSE,
  PRIMARY KEY( documento ),
  UNIQUE ( login )
);
CREATE TABLE prenotazione(
  id     SERIAL,
  codicevolo  VARCHAR(10) NOT NULL
      REFERENCES volo( codicevolo )
        ON UPDATE CASCADE
        ON DELETE CASCADE,
  documento VARCHAR(50) NOT NULL
      REFERENCES passeggero( documento )
      ON UPDATE CASCADE
      ON DELETE CASCADE,
  datarichiesta DATE,
  orarichiesta  TIME,
  PRIMARY KEY( id ),
  UNIQUE ( codicevolo, documento )
);
CREATE TABLE biglietto(
  id     SERIAL,
  codicevolo  VARCHAR(10) NOT NULL
      REFERENCES volo( codicevolo )
      ON UPDATE CASCADE
      ON DELETE CASCADE,
  documento VARCHAR(50) NOT NULL
      REFERENCES passeggero( documento )
      ON UPDATE CASCADE
      ON DELETE CASCADE,
  id_prenotazione INTEGER NOT NULL
        REFERENCES prenotazione( id )
        ON UPDATE CASCADE
        ON DELETE CASCADE,
  dataemissione DATE NOT NULL,
  prezzo     NUMERIC(10,2) NOT NULL,

  PRIMARY KEY( codicevolo, documento ),
  UNIQUE ( codicevolo, documento, id )
);

CREATE TABLE posto(
  lettera    CHAR NOT NULL,
  numero     INTEGER DEFAULT 0,
  PRIMARY KEY ( lettera, numero ),
  UNIQUE ( lettera, numero )
);
CREATE TABLE imbarco(
```

```
  lettera    CHAR NOT NULL,
  numero     INTEGER DEFAULT 0,
  imbarcato BOOLEAN DEFAULT FALSE,
  -- PRIMARY KEY DEL BIGLIETTO
  codicevolo  VARCHAR(10) NOT NULL,
  documento VARCHAR(50) NOT NULL,
  -- IDENTIFICATO DAL BIGLIETTO E DAL POSTO
  PRIMARY KEY ( codicevolo, documento, lettera, numero),

  FOREIGN KEY( lettera, numero )
      REFERENCES posto( lettera,numero )
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY( codicevolo, documento )
      REFERENCES biglietto( codicevolo,documento )
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Abbiamo aggiunto delle funzioni scritte in *plpgsql* per eseguire azioni di routine sul base di dati e per inserire automaticamente la data e l'ora di inserimento di determinate tuple. Si riporta di seguito il codice delle funzioni create.

```
CREATE OR REPLACE FUNCTION update_time_richiesta()
RETURNS TRIGGER AS '
BEGIN
  IF NEW.datarichiesta IS NULL THEN
    NEW.datarichiesta := current_date;
  END IF;
  IF NEW.orarichiesta IS NULL THEN
    NEW.orarichiesta := current_time;
    END IF;
  RETURN NEW;
END' LANGUAGE 'plpgsql';



CREATE OR REPLACE FUNCTION update_time_biglietto()
RETURNS TRIGGER AS '
BEGIN
  IF NEW.dataemissione IS NULL THEN
    NEW.dataemissione := current_date;
  END IF;
  RETURN NEW;
END' LANGUAGE 'plpgsql';



CREATE OR REPLACE FUNCTION update_numvoli()
RETURNS TRIGGER AS '
BEGIN
  IF NEW.dataemissione IS NULL THEN
    NEW.dataemissione := current_date;
  END IF;
  RETURN NEW;
END' LANGUAGE 'plpgsql';

CREATE OR REPLACE FUNCTION update_tessera() RETURNS trigger AS $$
BEGIN
  UPDATE passeggero
  SET numvoli = (SELECT count(*) FROM volo JOIN biglietto ON
    biglietto.codicevolo = volo.codicevolo
```

```
39                  JOIN tratta ON (volo.partenza = tratta.partenza
    AND volo.arrivo = tratta.arrivo )
40          WHERE passeggero.tessera=true AND passeggero.documento =
    biglietto.documento
41          GROUP BY passeggero.documento ),
42    miglia = (SELECT sum(distanza) FROM volo JOIN biglietto ON
      biglietto.codicevolo = volo.codicevolo
43                  JOIN tratta ON (volo.partenza = tratta.partenza
    AND volo.arrivo = tratta.arrivo )
44          WHERE passeggero.tessera=true AND passeggero.documento =
    biglietto.documento
45          GROUP BY passeggero.documento )
46    FROM biglietto
47    WHERE biglietto.dataemissione >= (SELECT date('now') - interval '
      3 year') AND NEW.documento = passeggero.documento;
48    RETURN NEW;
49
50  END;
51  $$ LANGUAGE 'plpgsql';
```

Per automatizzare l'esecuzione delle funzioni si sono scritti dei trigger, azioni eseguite al verificarsi di certe condizioni. I trigger creati sono i seguenti:

```
1  CREATE TRIGGER update_time_richiesta
2  BEFORE INSERT ON prenotazione
3    FOR EACH ROW EXECUTE PROCEDURE update_time_richiesta();
4
5  CREATE TRIGGER update_tessera
6  AFTER INSERT OR DELETE ON biglietto
7      FOR EACH ROW EXECUTE PROCEDURE update_tessera();
8
9  CREATE TRIGGER update_time_biglietto
10 BEFORE INSERT ON biglietto
11   FOR EACH ROW EXECUTE PROCEDURE update_time_biglietto();
```

3 Popolamento della base di dati

Per il popolamento della base di dati sono stati scritti dei programmi per la generazione automatica di query. In particolare sono stati creati dei programmi in Java per popolare le tabelle *Passeggero*, *Tratta* e *Volo* , in grado di gestire le chiavi esportate per ciascuna delle precendenti tabelle. I files .sql prodotti sono pronti per essere eseguiti senza alcuna necessità di modificare i files. I files per il popolamento si trovato nella cartella *scriptsql* e sono *popola_tratta.sql*, *popola_volo.sql*, *popola_passeggero*, *popola_prenotazione.sql* e *popola_biglietto.sql*.

Part II
Progetto del sito web

4 Progettazione logica

Si riportano di seguiti gli schemi di pagina seguiti durante la creazione del sito.

```
page-schema index unique (
  ricerca_volo:link(ricercavolo, *RicercaVolo)
  contatti_:link(contatti, *Contatti)
  chisiamo_:link(chisiamo, *ChiSiamo)
  -- Se il passeggero    loggato allora mostro
  area_personale:link(areapersonale, *BigliettiPage)
  -- Altrimenti
  area_personale:link(areapersonale, *Login)
)
```

```
page schema BigliettiPage(

  info_passeggero:list_of(
    nome: text;
    cognome: text;
    nazione: text;
    documento: text;
    username: text;
    password: password;
    // solo se il passeggero ha la tessera
    miglia : integer;
    num_voli : integer;
  );
  elenco_prenotazioni:list_of(
    codiceVolo: link( codiceVolo: text, *EmettiBigliettoPage );
    dataPartenza: date;
    oraPartenza: time;
    aeroportoDiPartenza: text;
    aeroportoDiArrivo: text;
    documento: text;
    dataRichiesta: date;
    oraRichiesta: time;
  );
  elenco_biglietti:list_of(
    codiceVolo: text;
    dataPartenza: date;
    oraPartenza: time;
    aeroportoDiPartenza: text;
    aeroportoDiArrivo: text;
    prezzo: text;
    dataEmissione: date;
  );
  picture:form(
    image:file;
    submit(*BigliettiPage);
  );
)
DB to page schema BigliettiPage
parameter(documento)
(
  bigliettiPage.elenco_prenotazioni: SELECT *
              FROM prenotazione
              WHERE documento=?documento?
```

```
            AND NOT EXISTS ( SELECT *
                FROM biglietto
                WHERE prenotazione.id=biglietto.id_prenotazione);
  bigliettiPage.elenco_biglietti: SELECT *
          FROM biglietto b
          WHERE b.documento=?documento?;
  info_passggero: SELECT *
      FROM passeggero
      WHERE documento = ?documento?
)
DB from page schema BigliettiPage(
  -- recupero il documento dalla sessione
  picture: UPDATE passeggero
     SET picture=?image?
     WHERE documento=?documento?

)
```

```
page-schema VoliPage (
  elencoVoli: list_of(
    codicevolo: link( codiceVolo, *PrenotazionePage );
    data: date;
    ora: time;
    partenza: text;
    arrivo: text;
    durataVolo: text;
    aereo: text;
  )
)
DB to page schema VoliPage
parameter( dataPartenza, partenza, arrivo )
(
      elencoVoli: SELECT volo.*
     FROM tratta JOIN volo on ( tratta.partenza = volo.partenza AND
     tratta.arrivo = volo.arrivo )
     WHERE datapartenza=?datapartenza?
       AND tratta.partenza ilike ?partenza?
                   AND tratta.arrivo ilike ?arrivo?
     ORDER BY orapartenza;
)
```

```
page-schema PrenotazionePage (
  dettagliovolo: list_of(
    codicevolo: link( codiceVolo, *PrenotazionePage );
    data: date;
    ora: time;
    partenza: text;
    arrivo: text;
    durataVolo: text;
    aereo: text;
  )
  prenotazione: form(
      nome: text;
      cognome: text;
      nazionalita: text;
      documento: text;
      username: text;
      password: text;
      Login    : submit();
      );
)
DB from page schema PrenotazionePage
```

```
(
  if( SELECT * FROM passeggero WHERE documento=?documento? )
  then
    INSERT INTO passeggero( nome, cognome,nazionalita,documento,
    login,password) VALUES
      (?nome?,?cognome?,?nazionalita?,?documento?,?username?,?
    password?);
  end;
  INSERT INTO prenotazione(codicevolo,documento) VALUES
      (?codicevolo?, ?documento?);
)
DB to page schema PrenotazionePage
parameter(codicevolo,documento)
(
  -- Recupera queste informazioni solo se l'utente    loggato
  form: SELECT * FROM passeggero
        WHERE documento=?documento?;
  dettagliovolo: SELECT * FROM volo
      WHERE codicevolo=?codicevolo?;
)
```

```

// Vedere cambiare la pagina o meno
page-schema EmettiBigliettoPage (
  numeroprenotazione: text;
  emissione: form(
      emetti:submit(*BigliettiPage);
    );
  annulla: form(
      annullaazione:submit(*BigliettiPage);
    );
)
DB from page schema EmettiBigliettoPage
(
  -- codicevolo e documento vengono recuperati dalla prenotazione
  -- il prezzo viene calcolato al momento dell'inserimento
    attraverso un algoritmo
  emissione: INSERT INTO biglietto(codicevolo, documento,
    id_prenotazione, prezzo )
      VALUES (?codicevolo?,?documento?,?id_prenotazione?,?prezzo?);
)
```

```
page-schema RicercaVolo unique(
  ricerca: form(
      partenza: text;
      arrivo: text;
      dataPartenza: date;
      Login    : submit();
      );
)
DB from page schema RicercaVolo
(
  ricerca: if(SELECT volo.*
     FROM tratta JOIN volo on ( tratta.partenza = volo.partenza AND
     tratta.arrivo = volo.arrivo )
     WHERE datapartenza=?datapartenza?
       AND tratta.partenza ilike ?partenza?
                    AND tratta.arrivo ilike ?arrivo?
     ORDER BY orapartenza )
    then *VoliPage else *RicercaVolo end;
)
DB to page schema RicercaVolo
```

```
parameter(partenza)
(
  partenza: SELECT DISTINCT partenza
            FROM tratta;
  arrivo: SELECT DISTINCT arrivo
    FROM tratta t
    WHERE t.partenza=?partenza? ORDER BY arrivo
)
```

```
page-schema Login (
  datiLogin: form(
      Username: text;
      Password: password;
      Login    : submit();
      );
)
DB to page schema Login
parameter(Username)
(
  if( SELECT login
      FROM passeggero
      WHERE login=?Username?;)
  then *BigliettiPage else *Login end;
)
DB from page schema Login
(
  form: if( SELECT login
      FROM passeggero
      WHERE login=?Username? AND password=?Password?;)
        then *BigliettiPage else *Login end;
)
```

5 Struttura dell'applicazione web

Architettura MVC2 L'elaborato è stato sviluppato seguendo il design pattern MVC2 e seguendo l'approccio servlet-centric. Questo pattern è composto da tre moduli:

- MODEL - Comprende la classe DBMS.java e i Java Data Beans. I beans sono contenuti nel package *bean*, mentre la classe DBMS.java è contenuta nel package *database*. DBMS.java si occupa di interrogare il DB e di manipolare i dati al suo interno. La comunicazione con main.java e picture.java avviene tramite i Java Data Beans in ambo le direzioni ;
- VIEW - Comprende tutte le JSPs, i javascript, i css e le foto profilo salvate all'interno della base di dati;
- CONTROLLER - Comprende le classi *main.java*, *picture.java*. Entrambe sono servlet con compiti differenti: *picture.java* gestisce le richieste di upload e download delle immagini, *main.java* controlla il flusso e gestisce il resto delle rihieste **GET** e **POST**. Utilizza *DBMS*.java per l'iterazione con la base di dati ed inoltre gli eventuali dati alla JSP appropriata ;

La servlet *picture.java* gestisce la parte multimediale dell'applicazione, caricando e scaricando dalla base di dati le foto profilo dei passeggeri. La servlet

identifica e gestisce le richieste HTTP sulla base che esse siano **GET** o **POST** e dal valore del parametro **ps** passato nella richiesta.

Per le richieste di tipo **GET**, **ps** può assumere questi valori:

- **downloadimage** - Scrive direttamente sullo stream output l'immagine relativa al passeggero specificato. I parametri richiesti sono:
  - **documento** Il documento del passeggero per il quale bisogna caricare l'immagine
- **Parametro assente o stringa vuota** - Viene passato il controllo a *index.jsp*

Per le richieste di tipo **POST**, **ps** può assumere questi valori:

- **uploadimage** - Recupera i dati del passeggero loggato e carica l'immagine passata nella base di dati. In caso di errore passa il controllo a *error.jsp* specificando un messagio di errore, altrimenti passa il controllo a *bigliettiPage.jsp* dopo aver caricato i relativi dati. I parametri richiesti sono:
  - **image** L'immagine da caricare come foto profilo
- **Parametro assente o stringa vuota** - Viene passato il controllo a *index.jsp*

La servlet *main.java* controlla le richieste da parte delle JSPs. La servlet identifica e gestisce le richieste HTTP sulla base che esse siano **GET** o **POST** e dal valore del parametro **ps** passato nella richiesta.

Per le richieste di tipo **GET**, **ps** può assumere questi valori:

- **areapersonale** - Vengono caricati dalla base di dati i voli e le prenotazioni dell'utente specificato nel parametro *pass*. Viene passato il controllo a *areapersonale.jsp*. Non sono richiesti ulteriori parametri in quanto i dati del passeggero vengono recuperati dall'attributo di sessione *pass*.
- **ricercavolo** - Vengono caricati gli aeroporti di partenza e viene passato il controllo a *ricercavolo.jsp*.
- **prenotazione** - Recupera le informazioni del volo e del passeggero. Infine passa il controllo a *prenotazionePage.jsp*. I parametri richiesti sono:
  - **codiceVolo** Codice del volo per recuperare le informazioni

  Non serve specificare un parametro per il passeggero in quanto le informazioni vengo recuperate dall'attributo della sessione.
- **logout** - Viene rimosso il passeggero loggato e passa il controllo a *ricercavolo.jsp* insieme agli aeroporti di partenza.
- **contatti** - Viene passato il controllo a *contatti.jsp*
- **emettibiglietto** - Passa il numero della prenotazionen a *emettiBigliettoPage.jsp* e gli passa il controllo. I parametri richiesti sono:
  - **numPrenotazione** Numero della prenotazione per la quale emettere il biglietto

- **chisiamo** - Passa il controllo a *chisiamo.jsp*
- **Parametro assente o stringa vuota** - Viene passato il controllo a *index.jsp*

Per le richieste di tipo **POST**, **ps** può assumere questi valori:

- **newbiglietto** - Viene richiesto di emettere un biglietto a partire dal numero della prenotazione. Una volta inserito il biglietto passa il controllo a *bigliettiPage.jsp* con i dati relativi al passeggero loggato, alle sue prenotazioni e ai suoi biglietti. I parametri richiesti sono:
  - **numPrenotazione** Numero della prenotazione per la quale emettere il biglietto
- **login** - Vengono ricevuti la coppia login e password dalla form *authentication* di *login.jsp*. Se la coppia è valida allora passa il controllo a *bigliettiPage.jsp* insieme ai dati del passeggero, delle prenotazioni e dei voli. Altrimenti imposta un messaggio di errore e passa il controllo a *login.jsp* I parametri richiesti sono:
  - **username** Username del passeggero
  - **password** Password del passeggero
- **nuovaprenotazione** - Vengono inviati i dati dalla form *prenotazione* di *prenotazionePage.jsp*, se i dati sono validi allora si procede ad aggiungere una nuova prenotazione del database. Se il passeggero non esiste ed è la prima volta che effettua una prenotazione allora sarà creato un nuovo passeggero a partire dai dati passati alla servlet. Il controllo poi passa a *esitoPage.jsp* con il relativo messaggio di stato. I parametri richiesti sono:
  - **nome** Nome del passeggero
  - **cognome** Cognome del passeggero
  - **documento** Numero del documento, può essere sia il numero di una carta d'identità sia di un passaporto
  - **nazionalita** Nazionalità del passeggero
  - **username** Username scelto dal passeggero
  - **password** Password scelta dal passeggero
  - **codicevolo** Codice del volo da prenotare
  - **tessera** Richiesta della tessera
- **volipage** - Ricerca un volo a partire dai valori specificati nei parametri passati e passa il controllo a *voliPage.jsp* se esiste almeno una corrispondeza, altrimenti viene passato il controllo a *ricercavolo.jsp*. I parametri richiesti sono:
  - **partenza** Aeroporto di partenza
  - **arrivo** Aeroporto di arrivo
  - **date** Data di partenza del volo

- **Parametro assente o stringa vuota** - Viene passato il controllo a *index.jsp*

La servlet *main.java* è in grado di gestire anche richieste AJAX sempre con lo stesso parametro **ps**. Il parametro può assumere i seguenti valori:

- **ajaxricercavolo** - Data un aeroporto di partenza passato come parametro, ricava tutti i possibili aeroporti di arrivo. Dopo aver recuperato le informazioni invia un messaggio JSON alla JSP chiamante. I parametri da passare sono:
  - **part** Indica l'aeroporto di partenza
- **checkusername** - Dato il nome utente di un passeggero interroga la classe *DBMS.java* se l'username esiste all'interno della base di dati. Se esiste setta il campo **isFree** all'interno del messaggio JSON di risposta a **false**, altrimenti a **true**. I parametri da passare sono:
  - **username** Indica il nome utente da controllare
- **checkdocumento** - Dato il numero di documento interroga la classe *DBMS.java* se il numero di documento esiste all'interno della base di dati. Se esiste setta il campo **isFree** all'interno del messaggio JSON di risposta a **false**, altrimenti a **true**. I parametri da passare sono:
  - **documento** Indica il documento da controllare

Gestione di errori Se si verifica un errore su una qualunque pagina JSP, allora viene caricata, automaticamente dal sistema, la pagina *error.jsp*. La pagina riporta la natura dell'errore. Se viene passato il parametro **msg** allora stamperà il messaggio di errore specificato.

Pagine JSPs e parametri passati Vengono descritte ora le JSPs utilizzate con i relativi parametri.

- *index.jsp* - Home page del sito. Viene mostrato un menù con il quale è possibile muoversi. La pagina è composta da un iframe a cui interno vengono caricate le singole JSPs.
- *error.jsp* - Se si verifica un errore su una qualunque pagina JSP, allora viene caricata, automaticamente dal sistema, la pagina *error.jsp*. La pagina riporta la natura dell'errore. Se viene passato il parametro **msg** allora stamperà il messaggio di errore specificato. Parametri passati alla pagina:
  - **msg** Messaggio di errore da stampare
- *bigliettiPage.jsp* - Pagina personale del passeggero dove vengono mostrati i dati personali, i biglietti e le prenotazioni. È possibile emettere un biglietto cliccando sul link del codice del volo nella tabella inerente alle prenotazioni. Il codice è un link per la pagina *emettiBigliettoPage.jsp*. È presente una form per caricare un'immagine personale come immagine profilo. Se non è specificata un'immagine all'interno della base di dati allora la JSP caricherà un'immagine di default. Parametri passati alla pagina:

  - **prenotazioni** Tutte le prenotazioni del passeggero che si è logggato
  - **biglietti** Tutti i biglietti emessi del passeggero loggato
  - **pass** Oggetto di tipo PasseggeroBean con le informazioni del passeggero

- *chisiamo.jsp* - Pagina riportante una breve descrizione della società e del loro operato;

- *contatti.jsp* - Pagina dove si possono trovare i contatti dell'azienda;

- *emettiBigliettoPage.jsp* - Pagina dove si emette il biglietto a partire da una prenotazione. Parametri passati alla pagina:

  - **numPrenotazione** Numero della prenotazione per la quale si vuole immettere il biglietto.

- *esitoPage.jsp* - Pagina di esito per la prenotazione. Se la prenotazione è andata a buon fine verrà visualizzato un messaggio di conferma, altrimenti di errore. Parametri passati alla pagina:

  - **status** Esito della prenotazione.

- *login.jsp* - Pagina per il login del passeggero. Prima di effettuare il login il passeggero deve aver prenotato almeno un volo. Parametri passati alla pagina:

  - **auth** Messaggio di errore se l'autenticazione precedente non è andata a buon fine

- *prenotazionePage.jsp* - Pagina dove vengono mostrati i dettagli del volo scelto ed una form per la prenotazione del volo. I dati inseriti nella form vengono controllati da una funzione JQUERY. Ogni campo ha collegato una funzione per la verifica della correttezza dei dati inseriti. I campi **documento** e **username** vengono validati real-time tramite due funzioni AJAX, una per validare il documento e l'altra per l'username. Entrambe le funzioni richiedono una conferma alla servlet sulla disponibilità del nome utente e del documento. Per entrambi si controlla che non esistano già all'interno della base di dati. Per gli altri campi viene eseguito un controllo sulla lunghezza e sul contenuto. Parametri passati alla pagina:

  - **volo** Oggetto di tipo VoloBean con all'interno tutte le informazioni sul volo scelto
  - **pass** Oggetto di tipo PasseggeroBean con le informazioni del passeggero. Questo parametro è diverso da *null* solo se il passeggero ha effettuato il login in precedenza.

- *ricercavolo.jsp* - Pagina per la ricerca di un volo. Mediante la form si devono scegliere un aeroporto di partenza, un aeroporto di arrivo e la data di partenza. Gli aeroporti di arrivo vengono caricati in un secondo momento, appena l'utente sceglie l'aeroporto di partenza, una funzione AJAX richiede alla servlet tutti gli aeroporti collegati all'aeroporto di partenza scelto. Il campo per la scelta della data di partenza presenta un "datepicker" di default che si disabilita quando riconosce come browser

"Chrome". Questo perché "Chrome" ha già incorporato un datepicker quando rileva, all'interno di una form, un campo di type *date*. Parametri passati alla pagina:

  - **partenze** Sono tutti gli aeroporti dai quali è possibile partire
  - **status** Messaggio usato se non vengono trovati dei voli a partire dai criteri scelti.

- *voliPage.jsp* - Pagina dove vengono mostrati i voli trovati dopo la selezione attraverso la form di *ricercavolo.jsp*. Il codice è un link per la pagina *prenotazionePage.jsp*. Parametri passati alla pagina:

  - **voli** ArrayList dei voli richiesti

Path e Context Il context del sito web è *basi*, mentre il path relativo per raggiungere la servlet è *servlet/main*. La porta del server è la *8080*. Quindi l'url per accedere al sito web è: `http://localhost:8080/basi/servlet/main`

Login e sessioni Quando un passeggero esegue il login viene associata una sessione al client relativo, nella quale viene mantenuto l'attributo **pass**. L'attributo contiene l'oggetto relativo al passeggero loggato. L'utilizzo di sessioni implica che il browser del client debba avere l'accettazione di COOKIE attiva per poter navigare sul sito. Ogni sessione dura 600 secondi(10 minuti), dopo di che essa verrà "eliminata" e se avviene un altra richiesta da parte di un host a cui è scaduta la sessione, allora esso verrà reindirizzato alla *index.jsp*.

Part III
Scelte progettuali

6 Ipotesi e strategie adotattate

- La relazione *passeggero* ha **documento** e**login** come possibili identificatori. È stato scelto di usare il documento come chiave primaria, mantenendo comuqnue l'attributo login univoco sulla tabella.
- Per la relazione *biglietto* non è stato utilizzato l'identificatore univoco come chiave primaria per poter avere una consistenza nella base di dati.
- L'attributo **data_emissione** nell'entità *biglietto* viene aggiunto automaticamente da una funzione trigger, spiegata nella relazione.
- Gli attributi **data_richiesta** e **ora_richiesta** nell'entità *prenotazione* viene aggiunto automaticamente da una funzione trigger, spiegata nella relazione.
- Si è aggiunto l'attributo **imbarcato** nella relazione *imbarco* in quanto il biglietto viene sempre assegnato ad un posto, però il passeggero può non imbarcarsi sull'aereo.
- All'interno delle servlet sono state utilizzate le classi Command per la gestione delle azioni da intraprendere quando si riceve una richiesta. Tutte le classe estendo un'interfaccia comune di nome *Command.java*.

7 Hibernate

Hibernate è una piattaforma middleware open source per lo sviluppo di applicazioni Java, attraverso l'appoggio al relativo framework, che fornisce un servizio di Object-relational mapping (ORM) ovvero gestisce la persistenza dei dati sul database attraverso la rappresentazione e il mantenimento su database relazionale di un sistema di oggetti Java.

Per la creazione dei Java Data Beans si è utilizzato un plugin di Eclipse di nome JBoss Tools. Questo pacchetto permette di creare i Java Data Beans relativi alle tabelle all'interno di un database.

8 AJAX

AJAX è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive. Consiste nel creare una servlet Java che riceve parametri da parte di una pagina web (nel nostro caso una JSP); questa servlet elabora i dati e li invia alla pagina chiamante, in modo che possa ricevere dinamicamente dati e quindi modificare il suo aspetto o comportamento.

Si è voluta utilizzare questa tecnologia per rendere dinamica la selezione in un campo del form di tipo select; poiché però questi campi devono essere "popolati" con dati provenienti dal database, è stato necessario introdurre la tecnologia JQuery e JSON.

JQuery consiste in un insieme di librerie per semplificare la programmazione web, nel nostro caso è stato utilizzato internamente ad AJAX per eseguire una query sul database e mappare in un LinkedHashMap la risposta. Per inviare poi questa risposta alla pagina web, è stato utilizzato JSON, una teconogia che consente lo scambio di dati tra architetture client-server;