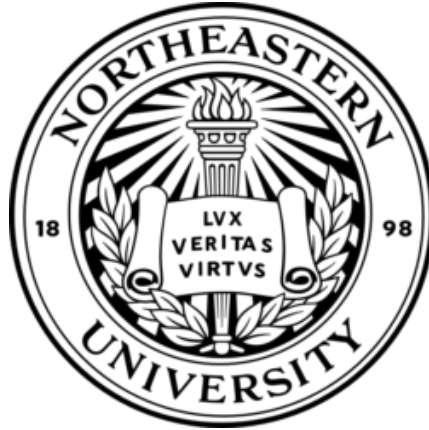


HomeToGo

Project Report



FALL 2021

WEB DEVELOPMENT TOOLS & METHODS ([INFO 6250](#))

INSTRUCTOR

Prof. YUSUF OZBEK

PRESENTED BY

NARENDRA RAJ RAMASAMY KANCHANAMALA
(NUID 001553969)

SUMMARY

HomeToGo is an online platform that's allows ease booking of properties (rooms, houses, villas etc.,) by the users of the application. HomeToGo also provides people who want to list their property for the rental business the scope to register their property which after the verifying the authenticity of the property the Admin gets to approve it. Once approved, the property is available for the end user to be booked based on it's availability. The motive is to encourage local property owners to advertise their properties and make monetary benefits.

TECHNOLOGY STACK

Front End Technologies:

- Java Server Pages (JSP)
- Java Standard Tag Library – Core tags
- Bootstrap
- HTML, CSS & JQuery

Back End Technologies:

- Spring MVC Framework – 5.2.2
- Hibernate – HQL & Criteria Queries
- MySQL

USER ROLES & TASKS PERFORMED

i) Customer

- The users can sign up using their preferred emailID and password
- Once Signed In, users can view the list of properties that are available based on their search criteria
- Ability to go through vital information about the Property pre booking such as Cost per day, Property Owner Contact Info, exact address etc.,
- Users can go ahead and rent the Property of their liking
- Once booking is confirmed, the user is notified through an email

ii)Property Owner

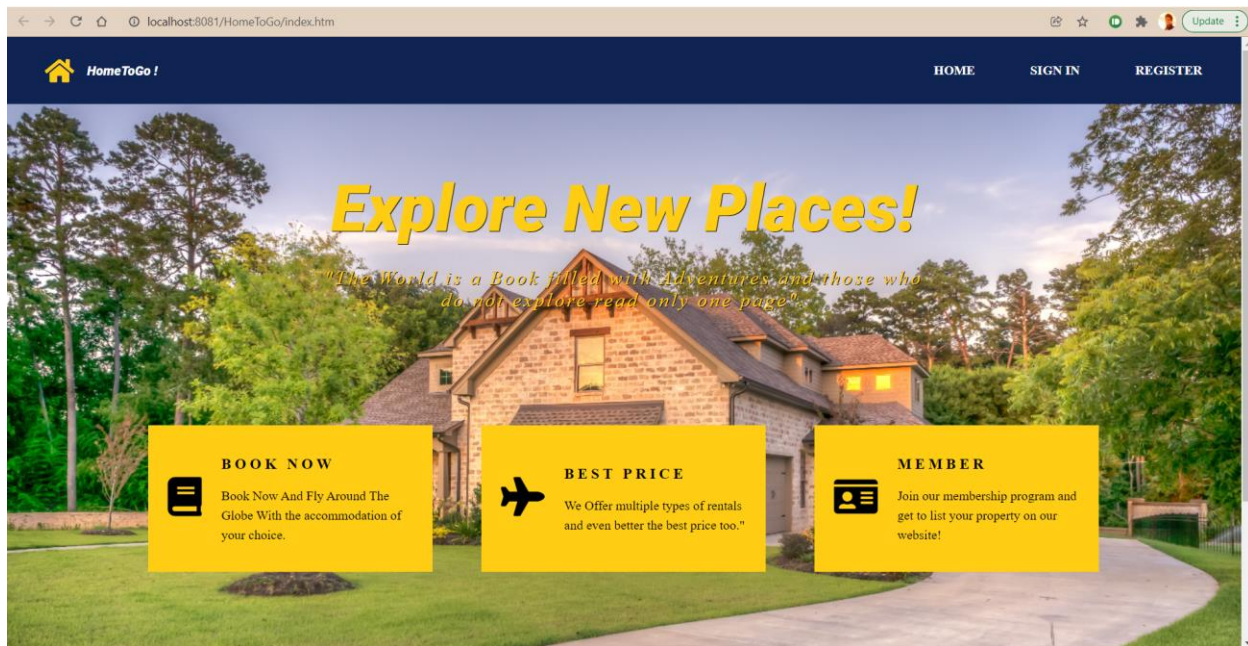
- The Property Owners can sign up also using their emailID and password
- Once Signed In, they can register their Property on the platform by filling out crucial information
- Once their Property is approved by the Admin, can update/delete their property at any time from their dashboard
- Property Owners can view the complete list of bookings(upcoming and past) of all their Properties in a single page

iii)Admin

- The Admin can sign in using the common sign in page.
- Once Signed In, the Admin can view the list of Properties that are awaiting his approval
- The Admin post verifying the crucial information of the Property may decide to Approve/Reject the Properties.
- Approved Properties are only available for booking on the platform by the users.

IMPORTANT FEATURES/PAGES SCREENSHOTS

LANDING PAGE OF THE APPLICATION:



BCRYPT – Encryption Algorithm to hash passwords

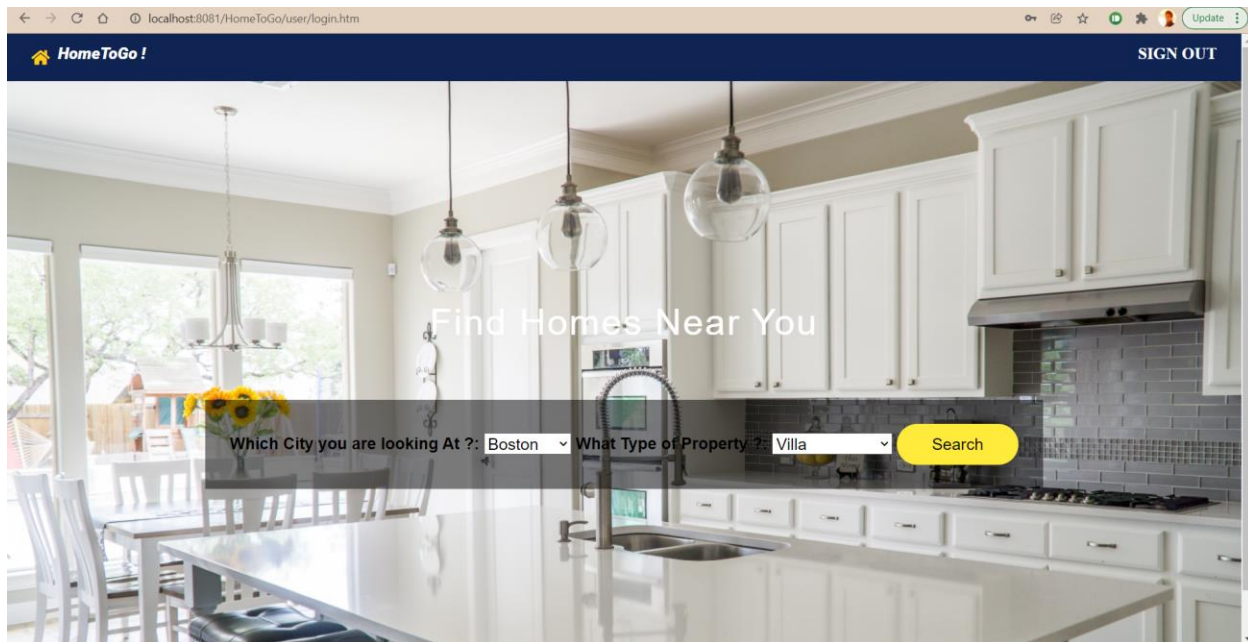
The below image shows how the user sensitive information is hashed and saved to the database. So in the event of a Database compromise, at least the user sensitive information such as passwords are not exposed to the hackers. [Click Here](#)

```
mysql> select * from user;
```

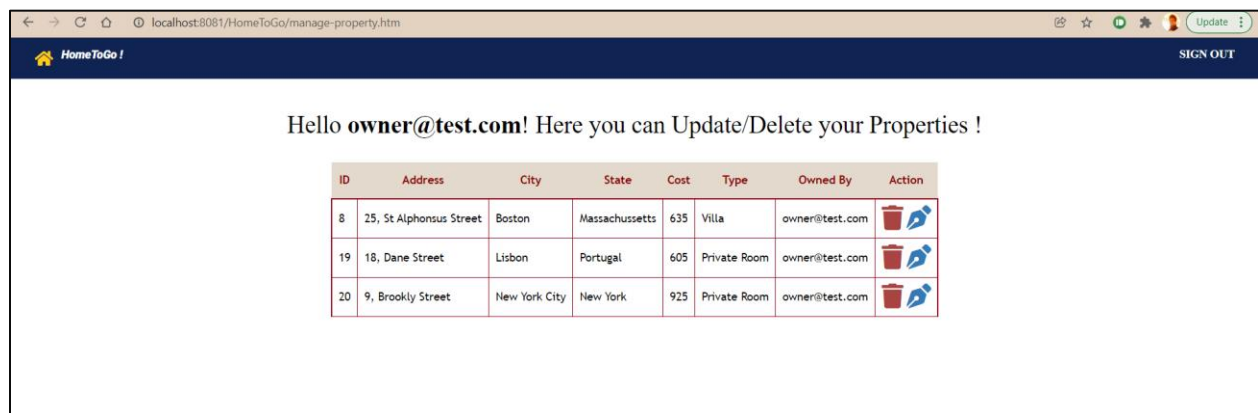
UserName	Password	Role
admin@test.com	\$2a\$10\$Z95Jh6oRtkIv3iavYyNNM0s32ARUJ5t6wK7Ymbr/qwRIQUjWwCPHi	admin
narenrj95@gmail.com	\$2a\$10\$eMYWyQj0WA9JG8UOF4.k04X0LnjLNp9soyyKKyANQHX9CLJyEvz.	customer
owner@test.com	\$2a\$10\$JjlsVX5mzhXgmPVCXwiJLeeFJDZHN0XGRfknEX48USjY0tiUe.HYG	propertyowner
owner@test5.com	\$2a\$10\$8wdhrMlvXnNk15TdXbCa5.vU0hjpsNet4wkud2d2bw4V/VKbD4Ww	propertyowner
ramasamykanchanama.n@northeastern.edu	\$2a\$10\$1hLSTHoAtmD76d0gvn5HTuJqW3thSqPSTgXMTVSgpyYbmn5awqb.C	propertyowner
random@test.com	\$2a\$10\$phkAf4HXCf1br/Gm3QQKyFHPtCryHV9J8Ser29fYMYFNKwC9YzPy	propertyowner
random@test2.com	\$2a\$10\$57lnaP304lw5t.yrc61Ts.UshABbPLNvLB3gf3nkyLabG5X8Z1sdC	propertyowner
random@test3.com	\$2a\$10\$hsDxVSzS0eLrixajy1XaulmhomMMixXoLImYK.MsIBq/vaoSKkXq	propertyowner
random@test4.com	\$2a\$10\$PQbRVOTAHBJ15aE6yDKQJenQXioJrU11dZZ1AQooIPBpyMueq8peW	propertyowner
user@demo.com	\$2a\$10\$mZbDK1etfACFNSHRCYR9q.riyhmS7wIAaPrm9Jh4UnTqm7ejNSPzu	customer
user@test.com	\$2a\$10\$1iFSDahLguIWY93s12lBv.eCU5.f484nDB6kTg/fB0ESz09FZ/3JO	customer
user@test2.com	\$2a\$10\$4PqRAxbXCfghYppnHaQ.H.Kvc86KUx6TwnM0tu/i3U7hIH4zsbCW.	customer

```
12 rows in set (0.00 sec)
```

SEARCH PAGE – POST SUCCESSFUL SIGN IN OF CUSTOMER



PROPERTY OWNER DASHBOARD – TO UPDATE/DELETE PROPERTY



ADMIN DASHBOARD – TO APPROVE/REJECT PROPERTIES



DYNAMIC DATA POPULATION

The Date Fields of a Property are updated on a real time basis

The available dates are only **20/12, 21/12 & 25/12** for check in & check out

Confirm Booking Details

Property Address:	25, St Alphonsus Street
City:	Boston
State:	Massachusetts
Cost Per Day:	635
Owned By:	owner@test.com

Which Date you are looking At Checking In ? 2021/12/20
2021/12/20
2021/12/21
2021/12/25

Which Date you are looking At Checking Out ? 2021/12/21
2021/12/25

Confirm

Confirm Booking Details

Property Address:	25, St Alphonsus Street
City:	Boston
State:	Massachusetts
Cost Per Day:	635
Owned By:	owner@test.com

Which Date you are looking At Checking In ? 2021/12/20
2021/12/20
2021/12/21
2021/12/25

Which Date you are looking At Checking Out ? 2021/12/20
2021/12/20
2021/12/21
2021/12/25

Confirm

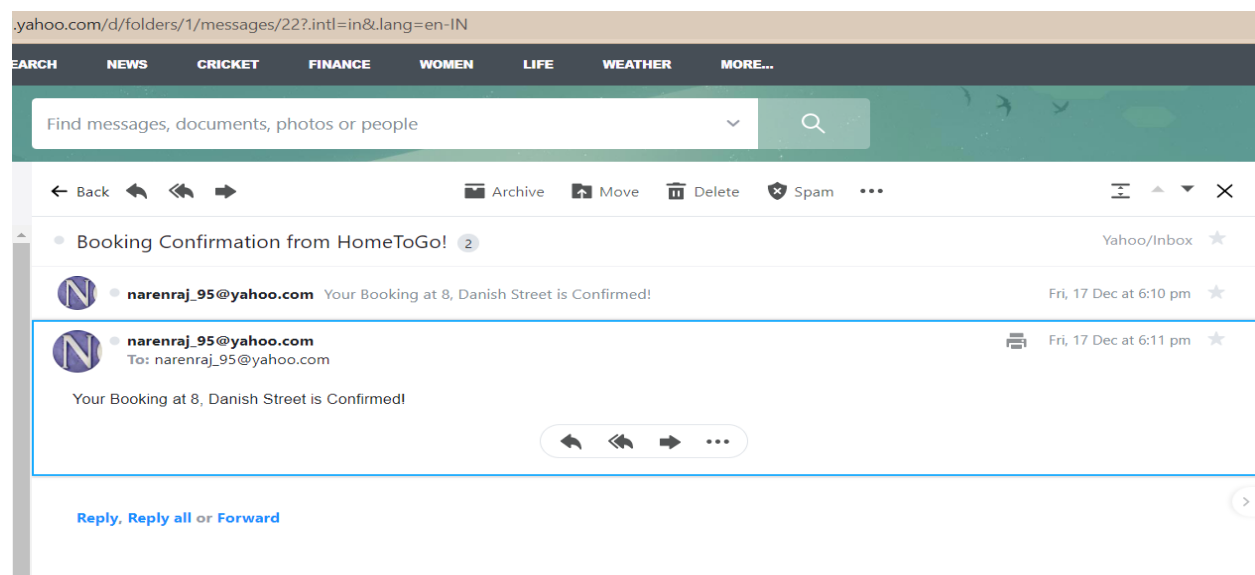
Booking for this Property

As there are already two other bookings that have been confirmed for the below dates, the available dates of this property is updated instantly to reflect the real-time status.

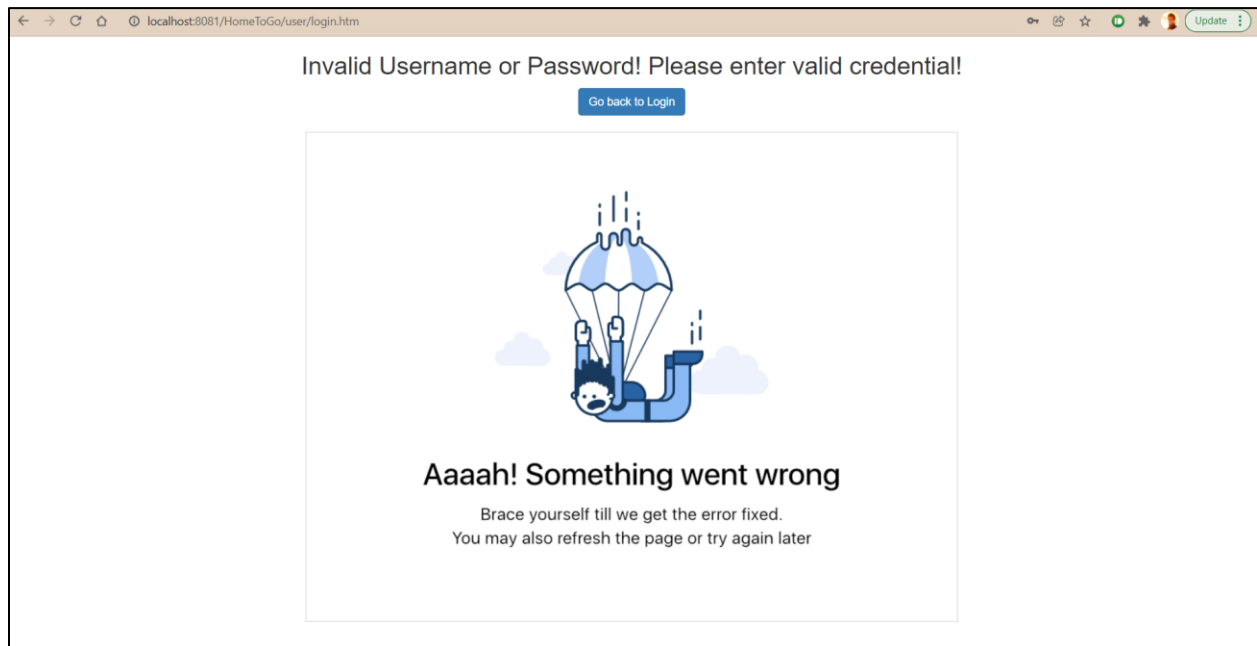
Booking_ID	Property_Booked	Booked_By_User	Property_Owner	Start_Date	End_Date
1	25, St Alphonsus Street	narenrj95@gmail.com	owner@test.com	2021/12/17	2021/12/19
11	25, St Alphonsus Street	user@test2.com	owner@test.com	2021/12/22	2021/12/24

2 rows in set (0.00 sec)

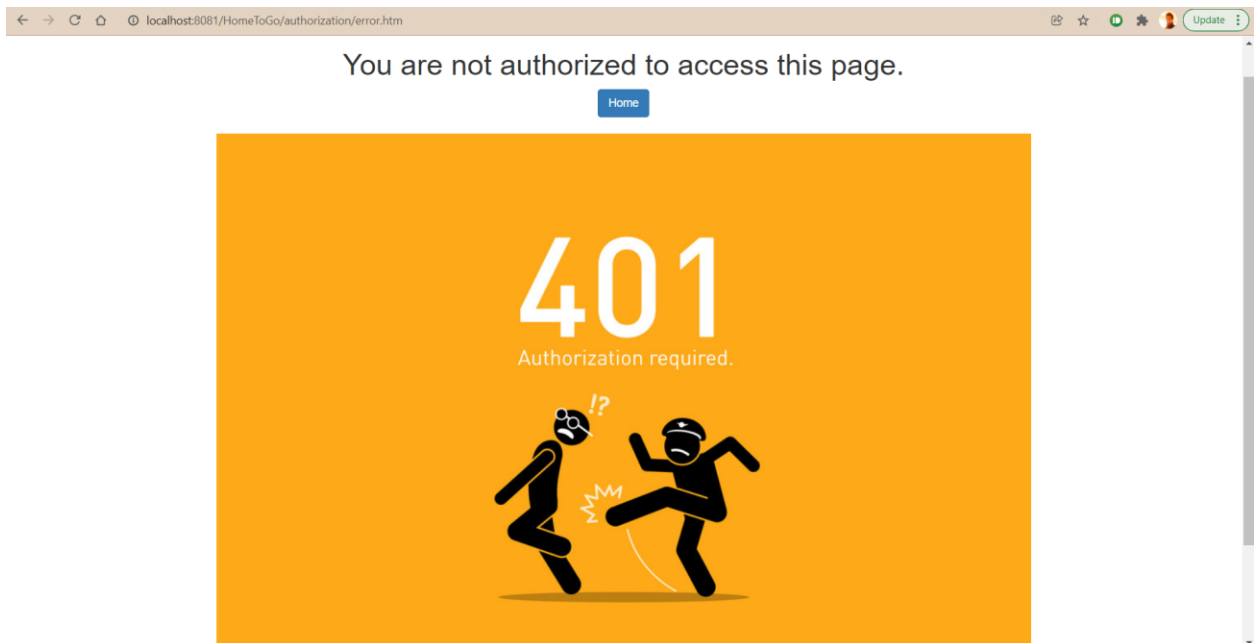
EMAIL NOTIFICATION – POST SUCCESSFUL BOOKING BY USER




EXCEPTION HANDLING – A sample page where an invalid username & password combo is handled



INTERCEPTOR – A Sample Page that demonstrates the use of interceptor



VALIDATORS – A sample page that demonstrates server side validation



User Registration Form

Note : Fields marked with an asterisk(*) are mandatory

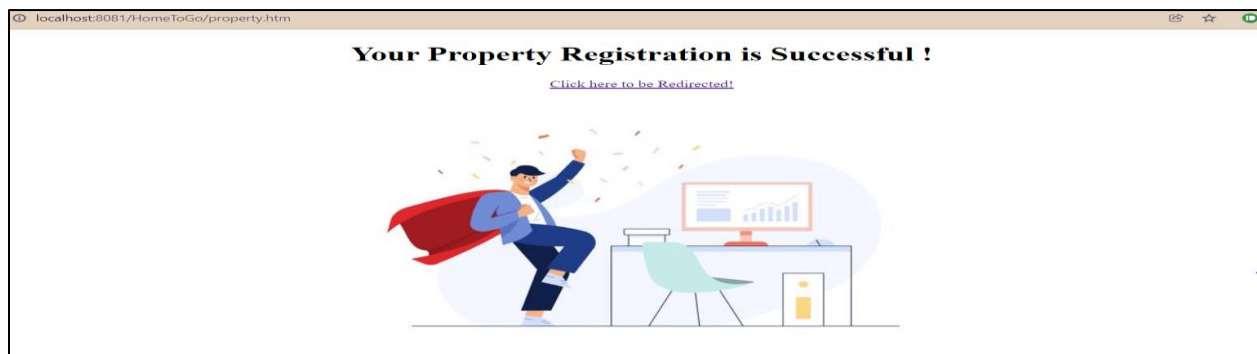
The Username field cannot be empty!

The Password field cannot be empty!

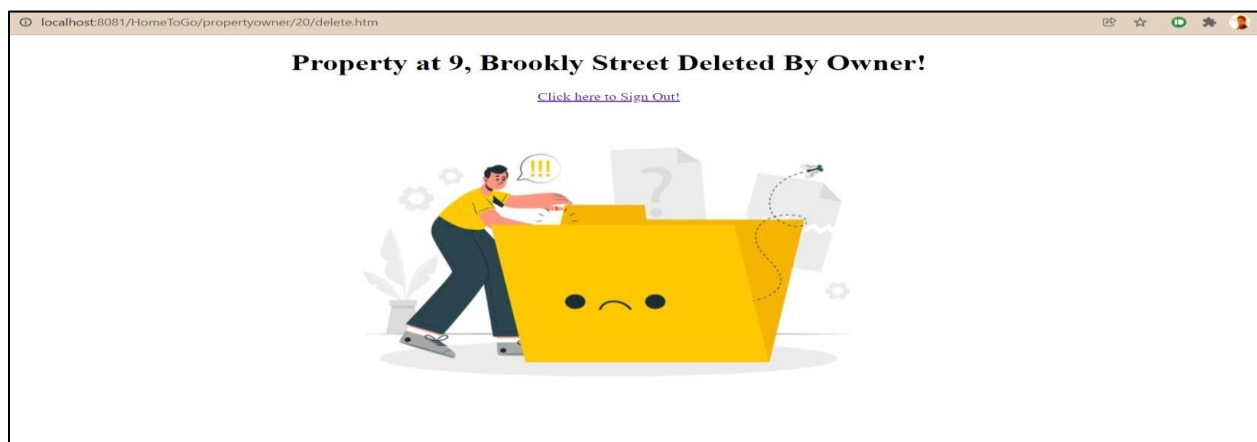
Email ID*:	<input type="text"/>
Password*:	<input type="password"/>
Registering For:	Customer <input type="button" value="v"/>

Others - Few pages to denote the Successful Capture of User Activity

i) When Property Owners successfully registers their Property



ii) When Property Owners delete their Property



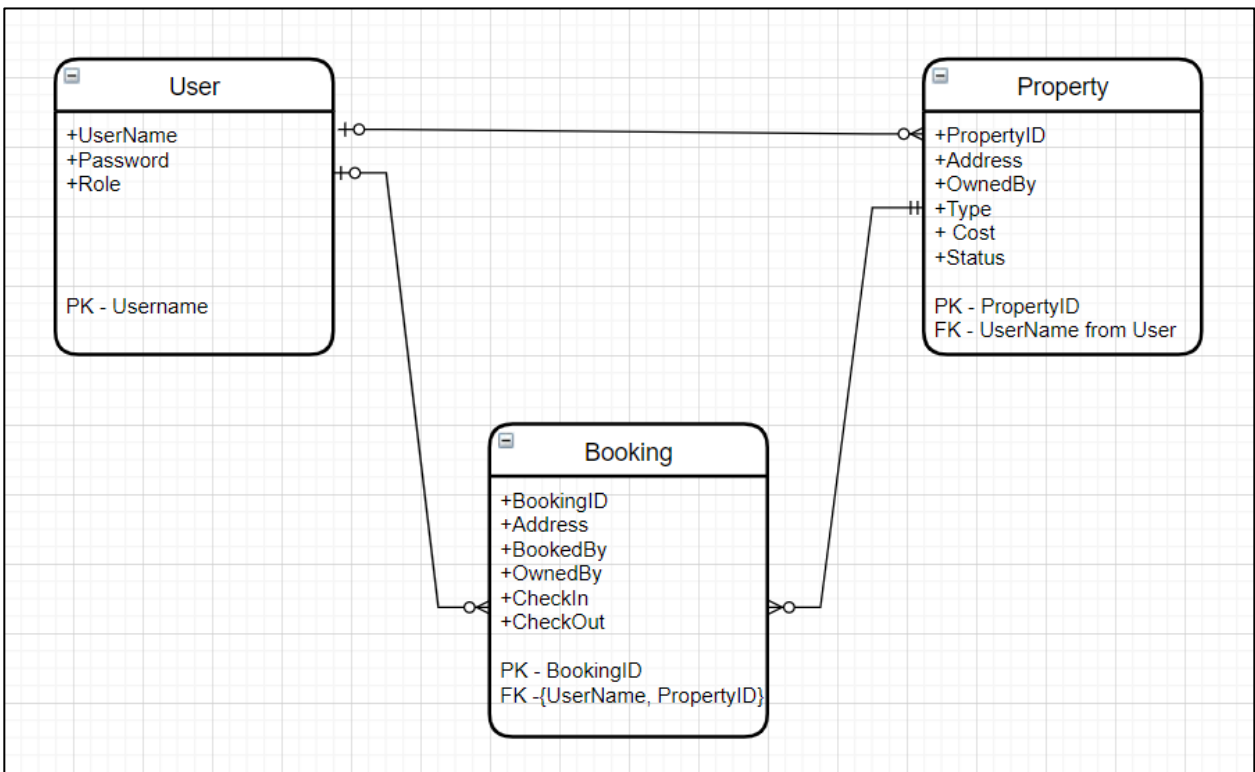
OBJECT MODEL

The Entities are **User, Property & Booking**

Relationship between User and Property is One Optional to Many Optional

Relationship between Property and Booking is Mandatory One to Many Optional

Relationship between User and Booking is One Optional to Many Optional



List Of Tables:

```
mysql> show tables;
+-----+
| Tables_in_hometogo |
+-----+
| booking              |
| property             |
| user                 |
+-----+
3 rows in set (0.01 sec)
```

CONTROLLER SOURCE CODE

ADMIN CONTROLLER:

```
@Controller
```

```
public class AdminController {
```

```
@GetMapping(value = "/admin-manage.htm")
```

```
protected ModelAndView listProperties(HttpServletRequest request,  
AdminDAO adminDao) throws Exception
```

```
{
```

```
    Map<String, Object> modelMap = new HashMap<String, Object>();
```

```
    modelMap.put("properties", adminDao.listALLProperties());
```

```
    return new ModelAndView("adminManageProperties", modelMap);
```

```
}
```

```
@RequestMapping(value = "/admin/{propertyID}/approve.htm", method =  
RequestMethod.GET)
```

```
protected ModelAndView approveProperty(@PathVariable("propertyID") int  
pid, HttpServletRequest request, AdminDAO adminDao) throws Exception
```

```
{
```

```
    Map<String, Object> modelMap = new HashMap<String, Object>();
```

```
    modelMap.put("properties", adminDao.approveProperty(pid));
```

```
    return new ModelAndView("propertyApproved", modelMap);
```

```
}
```

```
@RequestMapping(value = "/admin/{propertyID}/reject.htm", method = RequestMethod.GET)
```

```
protected ModelAndView rejectProperty(@PathVariable("propertyID") int pid,HttpServletRequest request, AdminDAO admindao) throws Exception
```

```
{  
    Map<String, Object> modelMap = new HashMap<String, Object>();  
    modelMap.put("properties", admindao.rejectProperty(pid));  
    return new ModelAndView("propertyRejected", modelMap);  
}  
}
```

PROPERTYOWNER CONTROLLER:

```
@Controller
```

```
public class PropertyOwnerFormController {
```

```
    @Autowired //Non-thread Safe and used to create instance of a class  
    PropertyOwnerValidator propertyValidator;
```

```
    @GetMapping(value = "/property.htm")
```

```
    protected ModelAndView showForm(HttpServletRequest request, User user, Property property) {
```

```
        ModelAndView mv = new ModelAndView();  
        HttpSession session = (HttpSession) request.getSession();  
        User user2 = (User) session.getAttribute("user");
```

```
String ownerName = user2.getUsername();
session.setAttribute("OwnerName", ownerName);

return new ModelAndView("addProperty", "property", ownerName);
}
```

```
@PostMapping(value = "/property.htm")
protected ModelAndView showSuccess(HttpServletRequest request,
Property property, User user, BindingResult result, SessionStatus status,
PropertyDAO propertydao) {

    HttpSession session = (HttpSession) request.getSession();

property.setPropertyAddress(request.getParameter("propertyAddress"));
    property.setPropertyCity(request.getParameter("propertyCity"));
    property.setPropertyState(request.getParameter("propertyState"));
    property.setPropertyType(request.getParameter("propertyType"));
    property.setPropertyCost(request.getParameter("propertyCost"));
    String OwnerName = (String) session.getAttribute("OwnerName");
    property.setPropertyOwner(OwnerName);
    System.out.println("Property Cost is:" + property.getPropertyCost());

    try {
        Property propReg = propertydao.saveItem(property);
        status.setComplete();

        return new ModelAndView("propertyRegistered", "property",
propReg);
    } catch (Exception e) {
        System.out.println("Inside Catch Block");
    }
}
```

```

        System.out.println("the message is " + e.getMessage());

        return new ModelAndView("error", "errorMessage", "An error
occurred while Registering the User !");
    }
}

```

```

@GetMapping(value = "/manage-property.htm")

protected ModelAndView showManageProperty(HttpServletRequest request, User user, Property property, PropertyDAO propertydao) throws
Exception {

    HttpSession session = (HttpSession) request.getSession();

    User user3 = (User) session.getAttribute("user");

    String ownerName2 = user3.getUsername();

    System.out.println("The Name of the Owner is: " + ownerName2);

    Map<String, Object> modelMap = new HashMap<String, Object>();

    modelMap.put("properties",
propertydao.getOwnerPropertyList(ownerName2));

    modelMap.put("ownerName", ownerName2);

    return new ModelAndView("manageProperty", modelMap);

}

```

```

@GetMapping(value = "/view-bookings.htm")

protected ModelAndView showBookings(HttpServletRequest request, User
user, Booking booking, BookingDAO bookingdao) throws Exception {

    HttpSession session = (HttpSession) request.getSession();

    User user4 = (User) session.getAttribute("user");

    String ownerName3 = user4.getUsername();

    System.out.println("The Name of the Owner is: " + ownerName3);

    List<Booking> bookingsList_Owner = new ArrayList<>();

```

```
        bookingsList_Owner =  
bookingdao.getBookingsOfAPropertyOwner(ownerName3);
```

```
        Map<String, Object> modelMap = new HashMap<String, Object>();  
        modelMap.put("confirmedBookings", bookingsList_Owner);  
        modelMap.put("ownerName", ownerName3);  
        return new ModelAndView("viewBookings", modelMap);  
    }
```

//Property Owner can delete his own properties from the application in the
manageProperty.jsp

```
@RequestMapping(value = "/propertyowner/{propertyID}/delete.htm",  
method = RequestMethod.GET)
```

```
protected ModelAndView deleteProperty(@PathVariable("propertyID") int  
pid, HttpServletRequest request, PropertyDAO propertydao) throws  
Exception {
```

```
        Map<String, Object> modelMap = new HashMap<String, Object>();  
  
        modelMap.put("deletedProperty",  
propertydao.getProperty(pid).getPropertyAddress());  
        propertydao.delete(propertydao.getProperty(pid));  
        return new ModelAndView("ownerDeletedProperty", modelMap);  
    }
```

```
@RequestMapping(value = "/propertyowner/{propertyID}/update.htm",  
method = RequestMethod.GET)
```

```
protected ModelAndView updateProperty(@PathVariable("propertyID") int
pid, HttpServletRequest request, PropertyDAO propertydao) throws
Exception {
```

```
    Map<String, Object> modelMap = new HashMap<String, Object>();
    modelMap.put("property", propertydao.getProperty(pid));
    modelMap.put("pid", pid);
    return new ModelAndView("ownerUpdateProperty", modelMap);
}
```

```
@RequestMapping(value = "/propertyowner/{propertyID}/update.htm",
method = RequestMethod.POST)
```

```
protected ModelAndView updatePropertyPost(@PathVariable("propertyID")
int pid, HttpServletRequest request, PropertyDAO propertydao, Property
property) throws Exception {
```

```
    Map<String, Object> modelMap = new HashMap<String, Object>();

    int result = propertydao.updatePropertyByOwner(pid,
request.getParameter("propertyAddress"),
request.getParameter("propertyType"),
request.getParameter("propertyCost"));

    modelMap.put("updatedCols", result);
    modelMap.put("pid", pid);
    return new ModelAndView("ownerUpdatePropSuccess", modelMap);
}
}
```

ROLE CONTROLLER:

```
@Controller
@RequestMapping("/authorization/*")
public class RoleController {

    @RequestMapping(value = "/authorization/error.htm", method =
RequestMethod.GET)
    protected ModelAndView postMenuError(HttpServletRequest request)
throws Exception {
        return new ModelAndView("authError");
    }
}
```

USER CONTROLLER:

```
@Controller
public class UserFormController {

    @Autowired //Non-thread Safe and used to create instance of a class
    UserValidator userValidator;

    @GetMapping(value = "/user/register.htm")
    public String showForm(ModelMap model, User user) {
```

```
//Command Class  
model.addAttribute("user", user);  
return "user-form";  
}
```

```
@PostMapping(value = "/user/register.htm")  
public ModelAndView showSuccess(HttpServletRequest request,  
@ModelAttribute("user") User user, BindingResult result, SessionStatus  
status, UserDao userDao) throws UserException {
```

```
    userValidator.validate(user, result);
```

```
    if (result.hasErrors()) {  
        return new ModelAndView("user-form");  
    }
```

```
    try {  
        System.out.println("The result values are: " + result.getModel());  
        System.out.println("The User Entered Password in Plain Text is: " +  
user.getPassword());  
        String plainPwd = user.getPassword();  
        String hashedPwd = hashPassword(plainPwd);  
        user.setPassword(hashedPwd);  
  
        User userReg = userDao.saveItem(user);  
        request.getSession().setAttribute("user", userReg);  
        status.setComplete();  
        return new ModelAndView("user-success", "user", userReg);  
    }
```

```

    } catch (Exception e) {
        return new ModelAndView("error", "errorMessage", "An error
occurred while Registering the User !");
    }

}

```

```

@RequestMapping(value = "/user/login.htm", method =
RequestMethod.GET)

protected String goToHome(HttpServletRequest request) throws
Exception {
    return "login-form";
}

```

```

@RequestMapping(value = "/user/login.htm", method =
RequestMethod.POST)

protected ModelAndView loginUser(HttpServletRequest request, UserDao
userdao) throws Exception {
    HttpSession session = (HttpSession) request.getSession();
    try {

        String authToken = request.getParameter("authToken");
        System.out.println("! ----- " + authToken);

        User user = userdao.getUser(request.getParameter("username"),
request.getParameter("password"));
        if (user == null) {
            session.setAttribute("errorMessage", "Invalid username and
password! Please try again!");

```

```

        return new ModelAndView("error");
    }
    session.setAttribute("user", user);
    String title = user.getUserRole().trim();

    if (title.equals("admin")) {
        return new ModelAndView("adminLanding");
    } else if (title.equals("customer")) {

        Map<String, Object> modelMap = new HashMap<String,
Object>();

        //For Dynamically populating the City Field
        List<Property> listProp = new ArrayList<>();
        listProp = userDao.listCities();
        List<String> distinctCities = new ArrayList<>();
        if (listProp.size() > 0) {
            System.out.println("Inside If Condition Of LoginUser
Handler");
            String city = null;
            for (Property item : listProp) {
                city = item.getPropertyCity();
                if (!distinctCities.contains(city)) {
                    distinctCities.add(city);
                }
            }
        }
    }
}

```

```

        modelMap.put("properties", distinctCities);
        return new ModelAndView("customerLanding", modelMap);
    } else if (title.equals("propertyowner")) {
        return new ModelAndView("propertyOwnerLanding");
    } else {
        return null;
    }
} catch (UserException e) {
    session.setAttribute("errorMessage", "An error occurred while
logging in");
    return new ModelAndView("error");
}
}

```

//For Signing Out Users and Invalidating Session

```

@RequestMapping(value = "/signOut.htm", method =
RequestMethod.GET)
protected String signOutUser(HttpServletRequest request) throws
Exception {

```

```

    System.out.println("Inside Sign Out Controller !");
    HttpSession session = (HttpSession) request.getSession();
    session.invalidate();

```

```

    return "Home";
}

```

```

    @RequestMapping(value = "/user/search.htm", method =
RequestMethod.POST)

    protected ModelAndView searchResults(HttpServletRequest request,
    UserDao userDao) throws Exception {

        String cityName = request.getParameter("propertyCity");
        String propertyType = request.getParameter("propertyType");

        List<Property> listProp = new ArrayList<>();
        listProp = userDao.getPropertiesForUser(cityName, propertyType);
        Map<String, Object> modelMap = new HashMap<String, Object>();
        modelMap.put("searchResults", listProp);

        return new ModelAndView("userSearchResults", modelMap);
    }

//To Allow the User to Book A Property in PropertyBookingPage.jsp

    @RequestMapping(value = "/user/{propertyID}/booking.htm", method
= RequestMethod.GET)

    protected ModelAndView showPropertyInfo(@PathVariable("propertyID")
int pid, HttpServletRequest request, UserDao userDao, PropertyDAO
propertydao, BookingDAO bookingdao, Booking booking) throws Exception
{

        List<String> datesList = new ArrayList<>();
        Map<String, Object> modelMap = new HashMap<String, Object>();
        datesList = generateDates();

```

```

        System.out.println("Size of the list of Dates: " + datesList.size());
        modelMap.put("property", propertydao.getProperty(pid));

        //For Handling Date in the dropdown based on previous bookings if
any
        List<Booking> bookingsList = new ArrayList<>();

        bookingsList =
        bookingdao.getBookingsOfAProperty(propertydao.getProperty(pid).getPrope
rtyAddress());

        if (bookingsList.size() > 0) {

            System.out.println("Size of Bookings List: " + bookingsList.size());

            booking = bookingsList.get(bookingsList.size() - 1);

            System.out.println("End Date of Bookings List: " +
            booking.getEndDate());

            String removeSD = null, removeED = null;
            int indexSD, indexED;
            for (Booking item : bookingsList) {
                removeSD = item.getStartDate();
                removeED = item.getEndDate();

                indexSD = datesList.indexOf(removeSD);
                indexED = datesList.indexOf(removeED);
                int j = indexSD;
                for (int i = indexSD; i <= indexED; i++) {
                    datesList.remove(j);
                }
            }
        }

```



```
//getBookingsOfAProperty  
modelMap.put("populatedDates", datesList);  
return new ModelAndView("propertyBookingPage", modelMap);  
}
```

```
@RequestMapping(value =  
"/user/{propertyID}/{userName}/bookingSuccess.htm", method =  
RequestMethod.POST)
```

```
protected ModelAndView bookProperty(@PathVariable("propertyID") int pid,  
@PathVariable("userName") String userName, HttpServletRequest request,  
UserDAO userdao, PropertyDAO propertydao, BookingDAO bookingdao,  
Booking booking, Property property) throws Exception {
```

```
Map<String, Object> modelMap = new HashMap<String, Object>();
```

```
String startDate = request.getParameter("startDate");
```

```
String endDate = request.getParameter("endDate");
```

```
System.out.println("The Start & End Dates are: " + startDate + " " +  
endDate);
```

```
//Retrieve the Property Details using pid to populate the booking table
```

```
property = propertydao.getProperty(pid);

System.out.println("The Property Address is: " +
property.getPropertyAddress() + " UserName is: " + userName);


booking.setPropertyBooked(property.getPropertyAddress());

booking.setPropertyOwner(property.getPropertyOwner());

booking.setStartDate(startDate);

booking.setEndDate(endDate);

booking.setBookedByUser(userName);

bookingdao.saveItem(booking);

sendMail(userName, booking.getPropertyBooked());

return new ModelAndView("bookingSuccessful", modelMap);

}
```

//To Generate Hashed Password

```
private String hashPassword(String plainTextPassword) {

    return BCrypt.hashpw(plainTextPassword, BCrypt.gensalt());

}
```

//To Generate Dates

```
private List<String> generateDates() {
```

```

List<String> dates = new ArrayList<>();
SimpleDateFormat sdf = new SimpleDateFormat("YYYY/MM/dd");
//String strDate = sdf.format(date);
Calendar cal = new GregorianCalendar(2021, 11, 16);
int j = 1;
for (int i = 1; i < 10; i++) {
    cal.add(Calendar.DAY_OF_MONTH, j);
    dates.add(sdf.format(cal.getTime()));
}
return dates;
}

//To Send Email To Customer on Successful Booking

public static void sendMail(String recipient, String messageBody) {

    Properties properties = new Properties();

    properties.put("mail.smtp.port", "587");

    properties.put("mail.smtp.starttls.enable", "true");

    properties.put("mail.smtp.auth", "true");

    properties.put("mail.smtp.host", "smtp.mail.yahoo.com");


    final String myEmailID = "narenraj_95@yahoo.com";
    final String password = "tzahwcxzojsjblvl";


    Session session = Session.getInstance(properties, new
Authenticator() {

```

```
@Override
protected PasswordAuthentication getPasswordAuthentication() {
    return new PasswordAuthentication(myEmailID, password);
}

});
System.out.println("Preparing to Send Email");
session.setDebug(true);
try {
    MimeMessage message = new MimeMessage(session);

    message.setFrom(new InternetAddress(myEmailID));
    message.addRecipient(Message.RecipientType.TO, new
InternetAddress(recipient));

    message.setSubject("Booking Confirmation from HomeToGo!");
    message.setText("Your Booking at " + messageBody + " is
Confirmed!");

    Transport.send(message);
} catch (MessagingException mex) {
    mex.printStackTrace();
}
}
}
```

THANK YOU!