Radiology Information System

Zachary Mace,
Kevin Freyre,
Andrew Davis,
Cole Ledford

Team Responsibilities

- Zachary Mace – Team Lead
  - Features Architecture
  - Front/Back End development
  - Web Hosting
  - Debugging
  - Stress Testing

- Kevin Freyre – Scribe
  - Front/Back End development
  - Debugging
  - Stress Testing
  - Documentation

- Andrew Davis
  - User Info feature owner
  - Modeling and Design
  - Stress Testing
  - Debugging

- Cole Ledford
  - Billing feature owner
  - Modeling and Design
  - Stress testing
  - Debugging

RIS Workflow

- RIS Purpose: manage information and workflow in a radiology department
  - Schedule and register patients and appointments
  - Process patients and upload images
  - Create and format diagnostic reports
  - Store and manage diagnostic reports
  - Digitally track imaging files
  - Input and manage billing information

Project Description

- Install one of the two supplied RIS systems
- Test system and identify defects
- Modify system to address the bugs discovered
- Add two additional features to the system through software engineering process
  - Requirements
  - Design
  - Implementation
  - Verification
  - Maintenance

Project Tools Used

- Development Environment
  - VisualStudio Code
    - Java Extension Pack
    - Git for VSCode
  - MySQL 8.0
    - Server
    - Workbench
  - JDK 11.0.2
  - Apache-Maven 3.6.3
- Source Control
  - GitHub
- Web Development Framework
  - SpringBoot 2.4.4
  - Spring Security 5.4.6
  - Spring Email 2.3.4
  - Hibernate JPA
- Diagram Tools
  - Diagrams.net/Draw.io
  - MySQL Workbench
- Communication and Coordination
  - Discord
  - Microsoft Teams
  - Jira

Project Agenda

- Planning 06/30/2021 – 07/07/2021
  - Analyzed team member experience
  - Outlined project requirements
  - Familiarized with project tools
- System Install/Testing 07/07/2021 – 07/14/2021
  - Installed system to maintain
  - Setup database
  - Began testing and identifying issues
  - Debugging
- Feature Implementation 07/14/2021 – 07/21/2021
  - Researched potential new features
  - Feature approvals
  - Began designing/developing new features
- Feature Testing 07/21/2021 – 07/28/2021
  - Finished developing features
  - Tested and identified issues
  - debugging
- Project Finalization 07/28/2021 – 08/01/2021
  - Finished all deliverables and gathered all documentation

New Project Features

Billing

- Cost of appointment is assigned and determined by modality
- Addition of Insurance information and price in appointment scheduling.
- Billing statement sent to patient after having checked-in for their appointment

User Info

- Ability for users to manually edit their account information such as password, email and display name

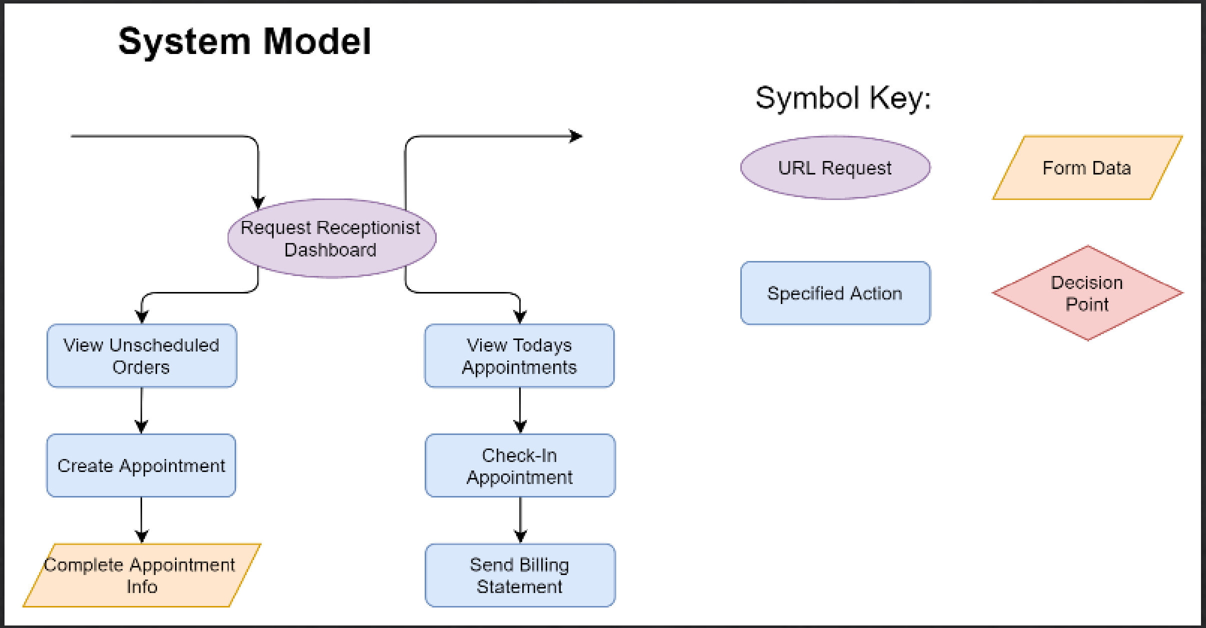
Billing Feature

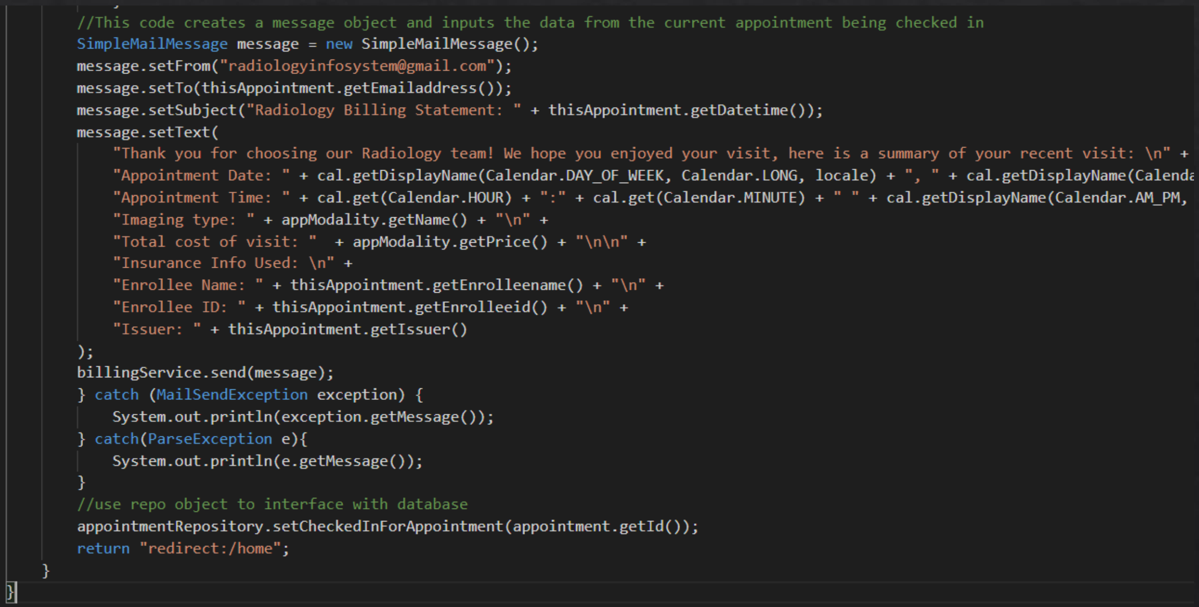
Functional Requirements

- Ability to change the price based on the choosing of a modality.
- Retrieval of Insurance information at the time of appointment scheduling.
- Total cost of modality displayed on receptionist dashboard as well as at the time of appointment scheduling
- Delivery of an email with price of modality as well as insurance information provided

Nonfunctional Requirements

- Reliably provide modality price and insurance info through email.
- Maintain Data integrity by safeguarding personal information from unauthorized/uncredentialled personnel
- Provide serviceability and maintainability to future email features





```
        //This code creates a message object and inputs the data from the current appointment being checked in
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("radiologyinfosystem@gmail.com");
        message.setTo(thisAppointment.getEmailaddress());
        message.setSubject("Radiology Billing Statement: " + thisAppointment.getDatetime());
        message.setText(
            "Thank you for choosing our Radiology team! We hope you enjoyed your visit, here is a summary of your recent visit: \n" +
            "Appointment Date: " + cal.getDisplayName(Calendar.DAY_OF_WEEK, Calendar.LONG, locale) + ", " + cal.getDisplayName(Calenda
            "Appointment Time: " + cal.get(Calendar.HOUR) + ":" + cal.get(Calendar.MINUTE) + " " + cal.getDisplayName(Calendar.AM_PM,
            "Imaging type: " + appModality.getName() + "\n" +
            "Total cost of visit: "  + appModality.getPrice() + "\n\n" +
            "Insurance Info Used: \n" +
            "Enrollee Name: " + thisAppointment.getEnrolleename() + "\n" +
            "Enrollee ID: " + thisAppointment.getEnrolleeid() + "\n" +
            "Issuer: " + thisAppointment.getIssuer()
        );
        billingService.send(message);
        } catch (MailSendException exception) {
            System.out.println(exception.getMessage());
        } catch(ParseException e){
            System.out.println(e.getMessage());
        }
        //use repo object to interface with database
        appointmentRepository.setCheckedInForAppointment(appointment.getId());
        return "redirect:/home";
    }
}
```

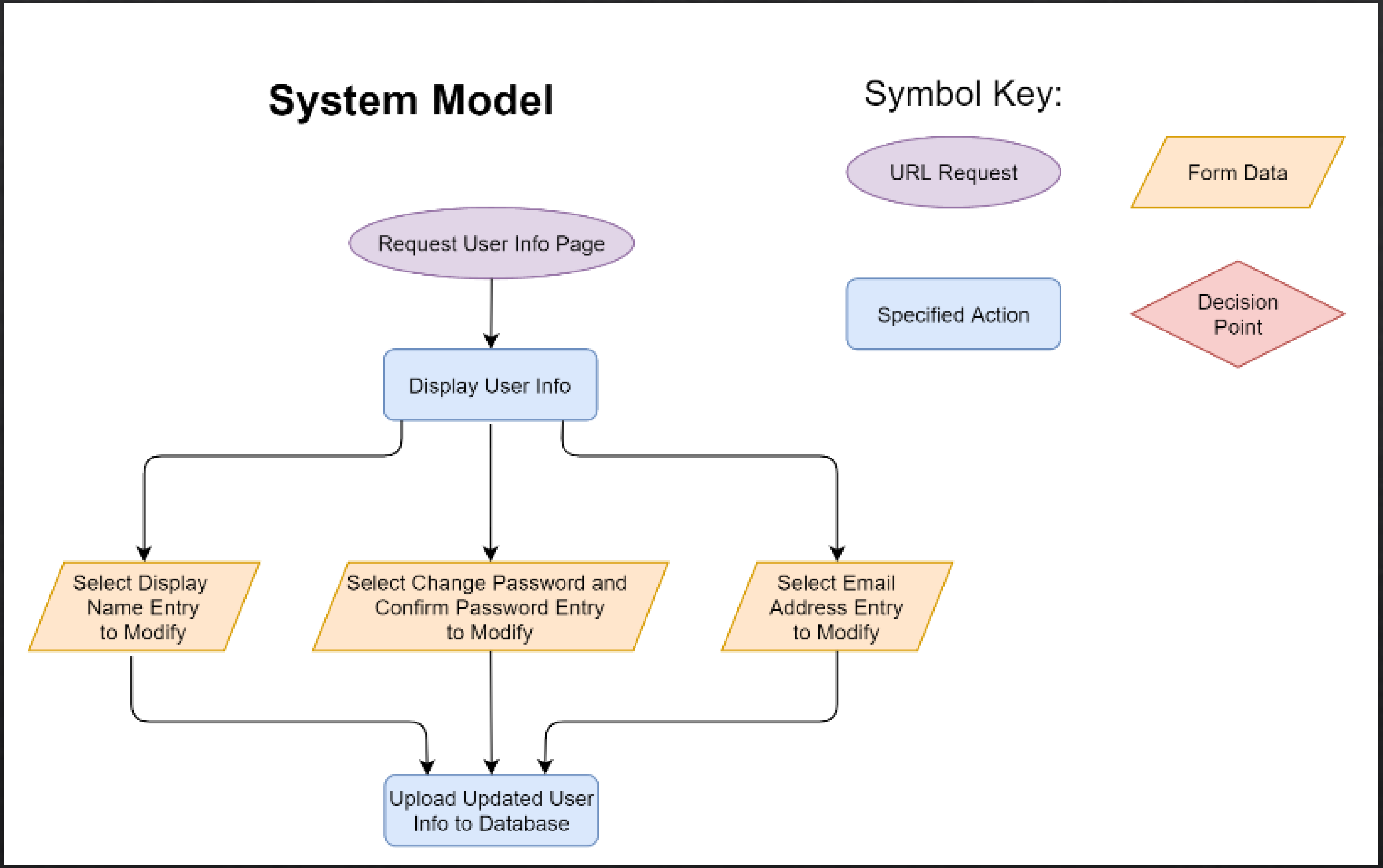
User Info Feature

Functional Requirements

- Allow user to update password, display name and email address
- Allow user to view user id and username
- Prevent user from updating user id and username

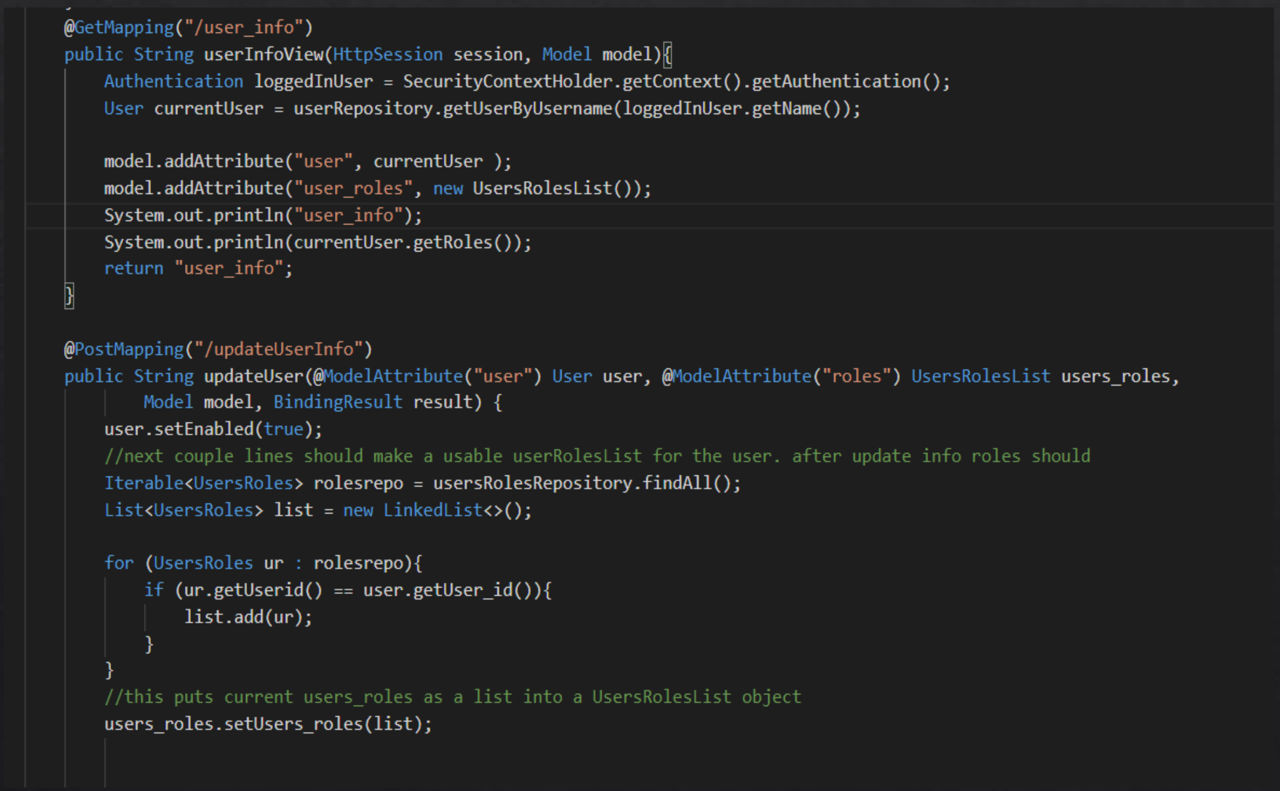
Nonfunctional Requirements

- Provide adaptability by allowing end users to change user info
- Provide accessibility by allowing user to view information that pertains to them
- Providing usability in an easy to use, simple dashboard for users


System Model
Symbol Key:
URL Request
Form Data
Specified Action
Decision Point
Request User Info Page
Display User Info
Select Display Name Entry to Modify
Select Change Password and Confirm Password Entry to Modify
Select Email Address Entry to Modify
Upload Updated User Info to Database

User Info Feature cont.

POST and GET mapping for user info page/form



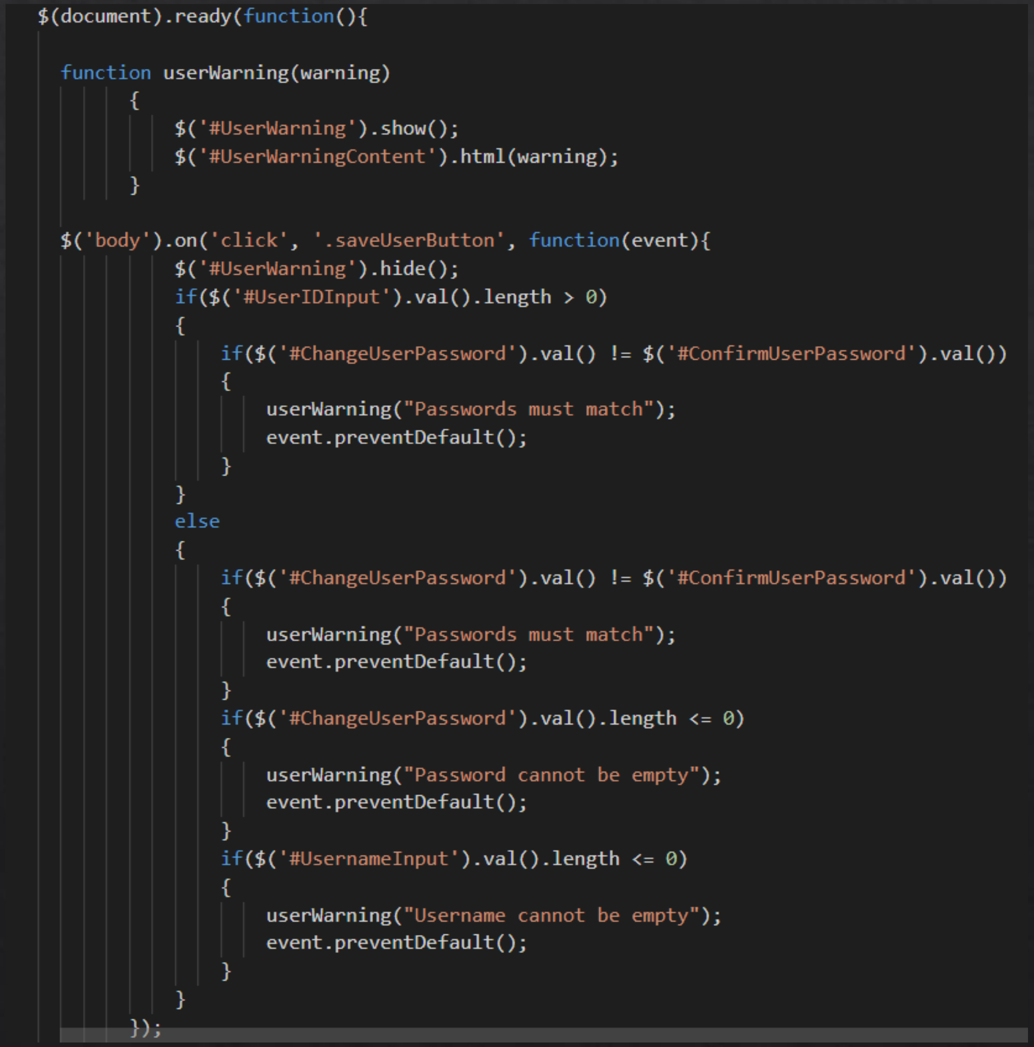
```java
@GetMapping("/user_info")
public String userInfoView(HttpSession session, Model model){
    Authentication loggedInUser = SecurityContextHolder.getContext().getAuthentication();
    User currentUser = userRepository.getUserByUsername(loggedInUser.getName());

    model.addAttribute("user", currentUser );
    model.addAttribute("user_roles", new UsersRolesList());
    System.out.println("user_info");
    System.out.println(currentUser.getRoles());
    return "user_info";
}

@PostMapping("/updateUserInfo")
public String updateUser(@ModelAttribute("user") User user, @ModelAttribute("roles") UsersRolesList users_roles,
        Model model, BindingResult result) {
    user.setEnabled(true);
    //next couple lines should make a usable userRolesList for the user. after update info roles should
    Iterable<UsersRoles> rolesrepo = usersRolesRepository.findAll();
    List<UsersRoles> list = new LinkedList<>();

    for (UsersRoles ur : rolesrepo){
        if (ur.getUserid() == user.getUser_id()){
            list.add(ur);
        }
    }
    //this puts current users_roles as a list into a UsersRolesList object
    users_roles.setUsers_roles(list);
```

Javascript to ensure password correctness in form



```javascript
$(document).ready(function(){

  function userWarning(warning)
      {
          $('#UserWarning').show();
          $('#UserWarningContent').html(warning);
      }

  $('body').on('click', '.saveUserButton', function(event){
          $('#UserWarning').hide();
          if($('#UserIDInput').val().length > 0)
          {
              if($('#ChangeUserPassword').val() != $('#ConfirmUserPassword').val())
              {
                  userWarning("Passwords must match");
                  event.preventDefault();
              }
          }
          else
          {
              if($('#ChangeUserPassword').val() != $('#ConfirmUserPassword').val())
              {
                  userWarning("Passwords must match");
                  event.preventDefault();
              }
              if($('#ChangeUserPassword').val().length <= 0)
              {
                  userWarning("Password cannot be empty");
                  event.preventDefault();
              }
              if($('#UsernameInput').val().length <= 0)
              {
                  userWarning("Username cannot be empty");
                  event.preventDefault();
              }
          }
      });
```

Lessons Learned

What went well:

- As a whole team communication was a strong point. We stayed on track and stayed connected on important, time-sensitive issues.
- As a team, we were very adaptable to changing environments and working in a distributed development team.
- Time management was not an issue for most of the project.

What needs to be improved:

- Documentation of assigned tasks, problems, and established codebase.
- Communication with the client. Several deadlines were missed due toa misunderstanding with the wants of the client.
- Interleave design and implementation steps better: Design should be done sooner to better the team's understanding of the implementation process.