

A Collection Of Resolved Issues

File locations of changes provided along with description of issues and fixes

Issue #14 Customer Feedback (Out of Range)

Description: This problem is a result of a feedback rating which is outside of the given range (0-5). When entering feedback outside of the given range, it is accepted as a 5/5 rating when it should be rejected.

Fix: The Jspinner component has a limit of 0 to 5. So, any numbers outside of this range will be automatically set to the closest bound. No code changes are necessary since our edge cases do not cause issues with the system but rather a temporary glitch on the UI. Because once you step out of the Jspinner, it automatically resets.

Issue #15 Customer Reservations (Future Reservations) Issue

Description: This problem allows the customer to set a reservation for any time in the future, meaning 100 years in the future. It can cause problems for keeping track of reservations.

Fix: Previously, customers could book as far into the future as wanted. This has been changed to a 60 day from the day of the current day.



```
4 PlatePlan/src/services/ReservationServiceImpl.java
@@ -69,7 +69,9 @@ public Reservation createCustomerReservation(Customer customer, LocalDate date,
String specialNotes) {
70
71     List<Table> tablesAvailable = tablesService.getTablesMatchingResReq(cap);
72
73     if (date.isAfter(LocalDate.now().plusDays(60L))) {
74         return null;
75     }
76     if (tablesAvailable.isEmpty()) {
77         return null;
78     }
79 }
```

Issue #16 Customer Reservations (Reservation Limit) Issue

Description: The customer can add any amount of reservations

Fix: Previously, customers can make as many reservations on different days as wanted. This has been changed to a limit of 5 reservations to avoid spamming.

```

PlatePlan/src/services/ReservationServiceImpl.java
@@ -77,7 +77,15 @@ public Reservation createCustomerReservation(Customer customer, LocalDate date,
77 77      Reservation reservation = new Reservation(UUID.randomUUID().toString(), customer.getEmail(), date, slot,
78 78      specialNotes, serviceUtils.getAllServersMap().get(tablesAvailable.get(0).getServer()),
79 79      tablesAvailable.get(0).getId(), cap);
80 -
80 +      try {
81 +          List<Reservation> customerReservations = db.getCustomerReservations(customer.getEmail());
82 +          if (customerReservations.size() >= 5)
83 +          {
84 +              return null;
85 +          }
86 +      } catch (AccountNotFoundException e1) {
87 +          e1.printStackTrace();
88 +      }
81 89      if (customer.getReservations() != null) {
82 90          try {
83 91              for (Reservation cusReservation : db.getCustomerReservations(customer.getEmail())) {

```

Issue #17 Table Management (0 - 100000 Capacity) Issue

Description: The business is able to allow for a number of capacity that is physically impossible. Also allows for table to be capacity of 0, can cause issues with consistency.

Fix: Added so that capacity of table can not be above 50 or be less than or equal to 0. In addition, added the ability to check table ID's to check duplicates do not exists.

```

PlatePlan/src/businessPanels/BusinessTableManageView.java
@@ -19,6 +19,7 @@
19 19  import javax.swing.JButton;
20 20  import javax.swing.JComboBox;
21 21  import javax.swing.JLabel;
22 + import javax.swing.JOptionPane;
22 23  import javax.swing.JPanel;
23 24  import javax.swing.JScrollPane;
24 25  import javax.swing.JTable;
@@ -281,7 +282,19 @@ public void actionPerformed(ActionEvent e) {
281 282
282 283      private void addTable() {
283 284          String tableId = textID.getText();
285 +
286 +          if (tablesService.isTableValid(tableId)) {
287 +              JOptionPane.showMessageDialog(null, "The table ID entered already exists!", "Error",
288 +              JOptionPane.ERROR_MESSAGE);
289 +          }
290 +      }
291 +
284 292          int cap = Integer.valueOf(textCapacity.getText());
293 +
294 +          if (cap <= 0 || cap >= 50) {
295 +              JOptionPane.showMessageDialog(null, "The capacity entered is invalid", "Error", JOptionPane.ERROR_MESSAGE);
296 +          }
297 +
285 298          String serverId = (String) serverBox.getSelectedItem();
286 299
287 300          for (String id : serverMap.keySet()) {

```

Issue #18 Code Smells (Reservation Class)

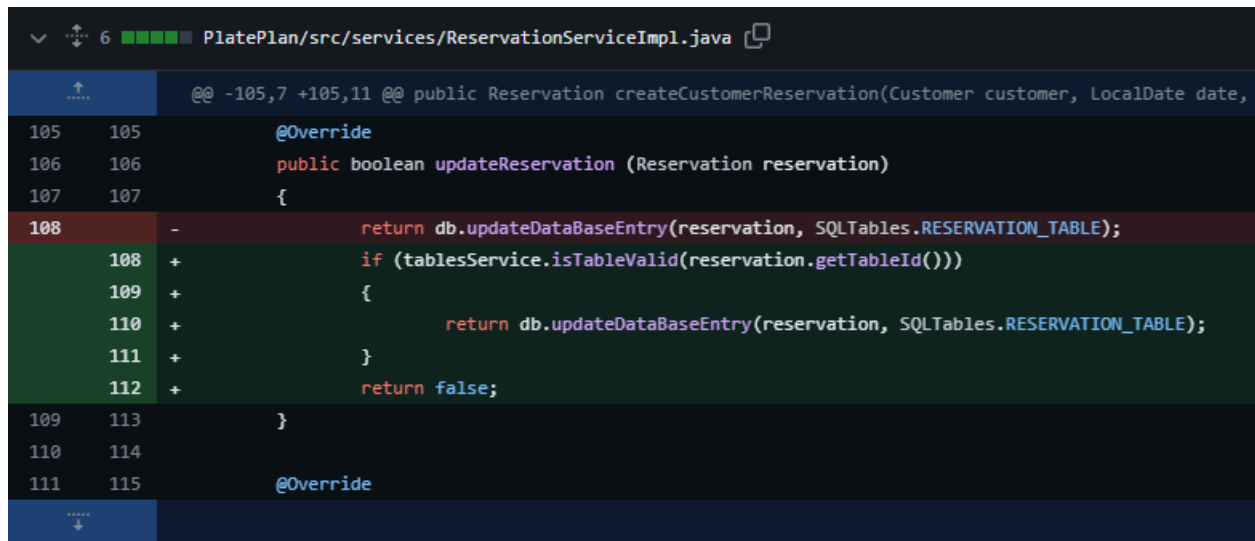
Description: A constructor with no real use, class already has a complete constructor with all necessary parameters for a reservation within reservation.java lines 33-43.

Fix: Removed unnecessary constructor which caused issues with unit tests, and therefore the unit tests have been switched from the old constructor to the new constructor.

Issue #20 Program does not perform validity check on the table id

Description: Program does not perform validity check on the table id

Fix: Fixed the issue by adding a method to table service that returns a boolean and validates the table id and returns error to user if there are issues.



```
@@ -105,7 +105,11 @@ public Reservation createCustomerReservation(Customer customer, LocalDate date,
105 105      @Override
106 106      public boolean updateReservation (Reservation reservation)
107 107      {
108  -      return db.updateDataBaseEntry(reservation, SQLTables.RESERVATION_TABLE);
108  +      if (tablesService.isTableValid(reservation.getTableId()))
109  +      {
110  +          return db.updateDataBaseEntry(reservation, SQLTables.RESERVATION_TABLE);
111  +      }
112  +      return false;
109 113      }
110 114
111 115      @Override
```

Issue #21 ClassCastException due to Improper Type Casting from TableModel to LocalDate

Description: In the BusinessReservations.java file, specifically at line 109, the code attempts to cast an object retrieved from a table model directly to LocalDate without prior validation or conversion from its original type. This practice is problematic since table models often store data in more generic formats such as String or java.util.Date. This direct casting approach can result in a ClassCastException if the object is not already a LocalDate, which is a common scenario when handling data from table models.

Fix: Fixed the issue by adding the correct date parser from String to Local Date

```
PlatePlan/src/businessPanels/BusinessReservations.java
@@ -104,7 +104,7 @@ public void actionPerformed(ActionEvent e) {
104 104         String id = tableModel.getValueAt(i, 0).toString();
105 105
106 106         // Cast the Date and Time objects to their respective types
107 -         LocalDate date = (LocalDate) tableModel.getValueAt(i, 1);
107 +         LocalDate date = LocalDate.parse(tableModel.getValueAt(i, 1).toString());
108 108         String time = tableModel.getValueAt(i, 2).toString();
109 109         String[] parts = time.split(" - ");
110 110         TimeSlot timeSlot = new TimeSlot(LocalDate.parse(parts[0]), LocalTime.parse(parts[1]));
```

Issue #22 SQL Related Code Smells in Direct Database Operations

Description: The BusinessReservations class, particularly at line 134, employs a direct database operation (db.getAllReservations()) that indicates potential SQL-related code smells, affecting the application's security and maintainability. The use of hard-coded SQL queries without proper parameterization is inherently vulnerable to SQL injection attacks. Such attacks occur when malicious inputs are able to alter the query, potentially leading to unauthorized data access or modification.

Fix: During itr3 development, this issue was caught and changed so that the component no longer makes direct calls to the database, the change can be seen in the following github commit: cf8b72f

Issue #25 Invalid Details for Account Registration

Description: When a user is registering a new account, they can enter empty fields

Fix: In issue number 26 and 27 because we already checked for only valid entries on the first name, last name and email, we don't have to worry about spaces in these fields because it is caught by the validator.

Issue #26 Invalid Emails for User Account Registration

Description: When a user is registering a new account, they can enter an invalid email.

Fix: To fix this issue we added string regex to check to make sure the email is the format

```
PlatePlan/src/customerPanels/CustomerSignUp.java
@@ -125,8 +125,11 @@ public void actionPerformed(ActionEvent e) {
125 125     }
126 126
127 127     private boolean signUpCustomer() {
128 +         String emailRegex = "^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$";
129 +
128 130         if (txtEmail.getText().isEmpty() || txtFirstName.getText().isEmpty() || txtLastName.getText().isEmpty()
129 -         || txtPass.getText().isEmpty() || !txtFirstName.getText().matches("[a-zA-Z ]+$") || !txtLastName.getText().matches("[a-zA-Z ]+$")) {
131 +         || txtPass.getText().isEmpty() || !txtFirstName.getText().matches("[a-zA-Z ]+$")
132 +         || !txtLastName.getText().matches("[a-zA-Z ]+$") || !txtEmail.getText().matches(emailRegex)) {
130 133             JOptionPane.showMessageDialog(null, "Invalid Field Entry, Please Try again", // Message to display
131 134                 "Error", // Title of the dialog
132 135             JOptionPane.ERROR_MESSAGE);
```

Issue #27 Invalid First and Last Names for Account Registration

Description: When a user is registering a new account, they can enter numbers and symbols into the first and last name fields.

Fix: Fixed the first and last name validation by using regex to see if the characters typed are part of lower and upper case alphabet if not then an error shows up

```
PlatePlan/src/customerPanels/CustomerSignUp.java
@@ -126,8 +126,8 @@ public void actionPerformed(ActionEvent e) {
126 126
127 127     private boolean signUpCustomer() {
128 128         if (txtEmail.getText().isEmpty() || txtFirstName.getText().isEmpty() || txtLastName.getText().isEmpty()
129 -         || txtPass.getText().isEmpty()) {
130 -             JOptionPane.showMessageDialog(null, "Empty fields detected", // Message to display
129 +         || txtPass.getText().isEmpty() || !txtFirstName.getText().matches("[a-zA-Z ]+$") || !txtLastName.getText().matches("[a-zA-Z ]+$")) {
130 +             JOptionPane.showMessageDialog(null, "Invalid Field Entry, Please Try again", // Message to display
131 131                 "Error", // Title of the dialog
132 132                 JOptionPane.ERROR_MESSAGE);
133 133         return false;
}
```

Issue #28 Menu Item Price Not Resetting on Invalid Input

Description: The application fails to reset the price of a menu item to its original value when encountering invalid characters in the input field. Specifically, when a user navigates to the menu page on a business account and attempts to modify the price of any menu item by entering a mix of numbers and characters, the system improperly retains the numerical portion of the input rather than reverting to the original price. This behavior could lead to unintended price updates and potential confusion for the users.

Fix: Fixed the issue by adding a try and catch around the parser

```
PlatePlan/src/componentPanels/BusinessMenuComponent.java
@@ -101,8 +101,15 @@ public void run() {
101 101     public void keyTyped(KeyEvent e) {
102 102         SwingUtilities.invokeLater(new Runnable() {
103 103             public void run() {
104 -             BusinessMenuComponent.this.menuItem.setPrice(Float.valueOf(txtPriceField.getText()));
105 -             updateMenuItem();
104 +             try {
105 +                 float newPrice = Float.parseFloat(txtPriceField.getText());
106 +                 BusinessMenuComponent.this.menuItem.setPrice(newPrice);
107 +                 updateMenuItem();
108 +             }catch (Exception e) {
109 +                 JOptionPane.showMessageDialog(null, "Price provided is not a number", "ERROR",
110 +                 JOptionPane.ERROR_MESSAGE);
111 +             }
112 +         }
106 113     }
107 114 }
108 115 }
```

Issue #29 Violation of Separation of Concerns in BusinessMenuManagement Class

The BusinessMenuManagement.java class demonstrates a significant code smell by directly invoking database operations within a Swing JPanel subclass. This practice tightly couples the user interface (UI) layer with the data access layer, contravening the principle of separation of concerns. This principle is pivotal in software design, advocating for the modularization of a

computer program into distinct sections, with each addressing a separate concern. The method calls to `DataBaseFactory.getDatabase().publishCustomerMenu()` within an action listener exemplify this issue, merging UI logic with database manipulation logic. Such a conflation complicates maintenance, testing, and limits the reusability of database access logic. It also makes it challenging to change or replace database implementations without impacting the UI code.

Fix: Adding separate method to the interface of menu service and performing the same operation

```
14 PlatePlan/src/businessPanels/BusinessMenuMangement.java
@@ -102,10 +102,16 @@ public void actionPerformed(ActionEvent e) {
102 102         JButton btnPublishMenu = new JButton("Publish Menu");
103 103         btnPublishMenu.addActionListener(new ActionListener() {
104 104             public void actionPerformed(ActionEvent e) {
105 -                 DataBase db = DataBaseFactory.getDatabase();
106 -                 db.publishCustomerMenu();
107 -                 JOptionPane.showMessageDialog(BusinessMenuMangement.this, "Menu successfully published", "Success",
108 -                     JOptionPane.INFORMATION_MESSAGE);
105 +                 boolean result = menuService.publishCustomerMenu();
106 +                 if (result)
107 +                 {
108 +                     JOptionPane.showMessageDialog(BusinessMenuMangement.this, "Menu successfully published", "Success",
109 +                         JOptionPane.INFORMATION_MESSAGE);
110 +                 }else {
111 +                     JOptionPane.showMessageDialog(BusinessMenuMangement.this, "Menu could be published", "ERROR",
112 +                         JOptionPane.ERROR_MESSAGE);
113 +                 }

```