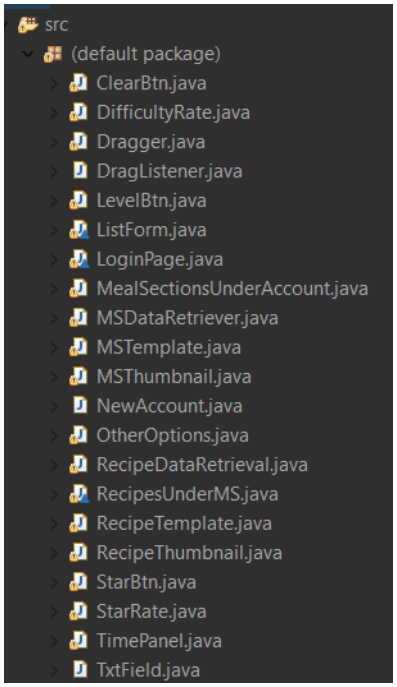
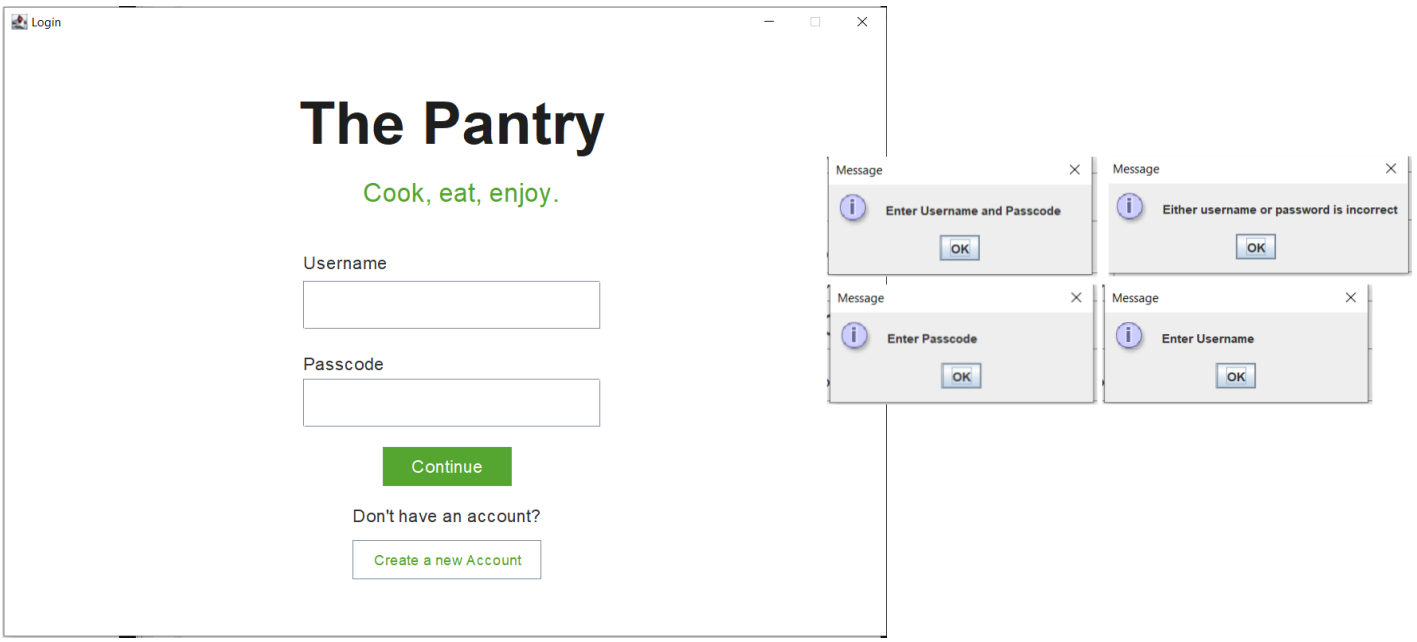
Criterion C: Development

This recipe manager is written in Java using the Eclipse IDE, and allows for users to write and store recipes through an account that they create. All account data is stored in a local sqlite database.

Program Files



Main window (LoginPage.java)



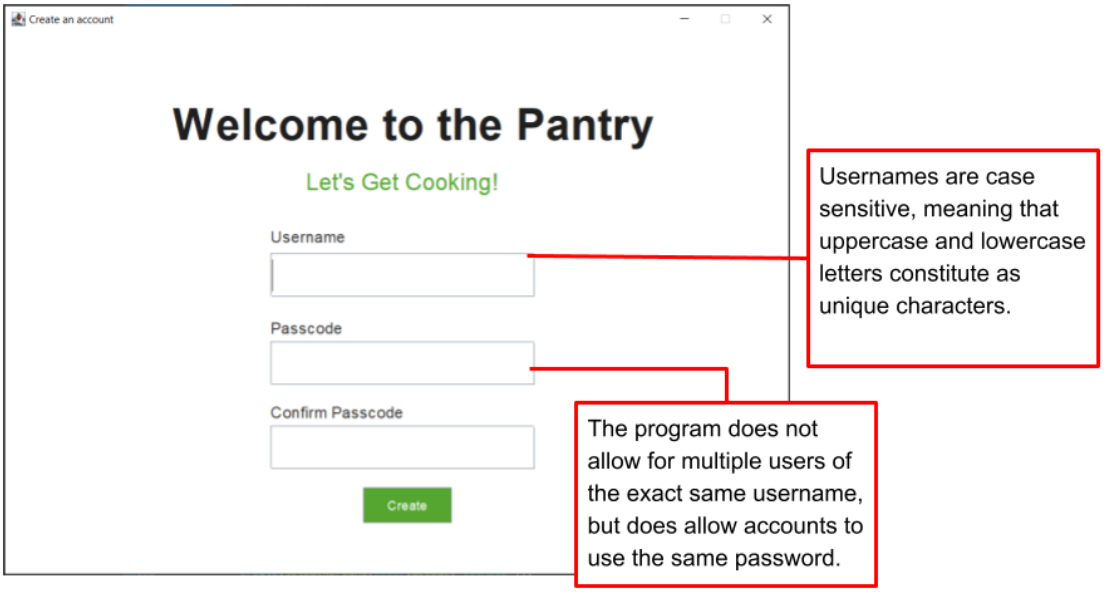
This is the first frame the user sees upon running the program. The JOptionPane messages displayed on the right are shown when the user has provided insufficient

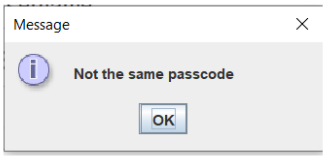
login information and clicked the continue button. For this frame, I used TextFields, Label, and Buttons and formatted the layout according to my preference as shown below:

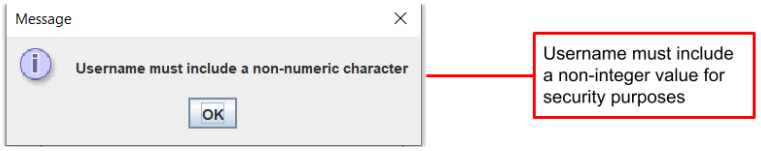
```java
desc = new JLabel("Cook, eat, enjoy.");
desc.setFont(new Font("Arial", Font.PLAIN, 27));
desc.setForeground(Color.decode("#55A630"));
desc.setSize(400, 100);
desc.setLocation(360, 110);
c.add(desc);

userLab = new JLabel("Username");
userLab.setFont(new Font("Arial", Font.PLAIN, 18));
userLab.setSize(100, 15);
userLab.setLocation(300, 225);
c.add(userLab);

txt1 = new JTextField();
txt1.setFont(new Font("Arial", Font.PLAIN, 30));
txt1.setSize(300, 50);
txt1.setLocation(300, 250);

c.add(txt1);

passLab = new JLabel("Passcode");
passLab.setFont(new Font("Arial", Font.PLAIN, 18));
passLab.setSize(150, 20);
passLab.setLocation(300, 325);
c.add(passLab);

txt2 = new JPasswordField();
txt2.setFont(new Font("Arial", Font.PLAIN, 30));
txt2.setSize(300, 50);
txt2.setLocation(300, 350);
c.add(txt2);
```

```java
// Login option display
if (s.equals("Login")) {

    cont = new JButton("Continue");
    cont.setFont(new Font("Arial", Font.PLAIN, 18));
    cont.setSize(130, 40);
    cont.setForeground(Color.white);
    cont.setBackground(Color.decode("#55A630"));
    cont.setLocation(380, 420);
    c.add(cont);

    noAcc = new JLabel("Don't have an account?");
    noAcc.setFont(new Font("Arial", Font.PLAIN, 18));
    noAcc.setSize(300, 100);
    noAcc.setLocation(350, 440);
    c.add(noAcc);

    newAcc = new JButton("Create a new Account");
    newAcc.setFont(new Font("Arial", Font.PLAIN, 15));
    newAcc.setSize(190, 40);
    newAcc.setLocation(350, 515);
    newAcc.setBackground(Color.WHITE);
    newAcc.setForeground(Color.decode("#55A630"));
    c.add(newAcc);

    cont.addActionListener(this);
    newAcc.addActionListener(this);

}
```

Create new account function







```java
// when creating a new acc
if (e.getSource() == newAcc) {
    CreateLoginForm createAccForm = new CreateLoginForm("New acc");
} else if (e.getSource() == create) {
    String username = txt1.getText();
    String passcode = txt2.getText();
    String passcodeCheck = txt3.getText();
    Boolean validName = false;

    // if atleast 1 character in the username is non-numeric,
    for (int i = 0; i < username.length(); i++) {
        if (Character.isLetter(username.charAt(i))) {
            validName = true;
            break;
        }
    }

    // empty username
    if (username.equals("")) {
        JOptionPane.showMessageDialog(this, "Enter Username");
    }
```

Loops through characters of string to confirm at least 1 alphabetic char

More JOptionPane message dialogues…

```java
// valid passcode and username
// open up the new account which sends passcode and username to the database
else {
    NewAccount newKitchen = new NewAccount(txt1.getText(), txt2.getText());
    dispose();
}
```

NewAccount is the class which creates the query and sends new account data to database

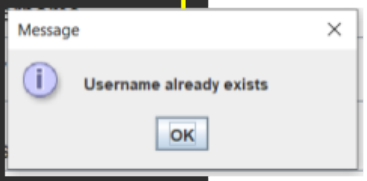
NewAccount.java is the class which sends the new account information to the database and also checks if that username exists and presents the JOptionPane error message. A repeated username entry will cause an exception, since the sqllite database will not accept duplicate entries. So this error can be caught with the JOptionPane error message.

In the database, a **users** table holds all information about users. **Userscol** refers to the username and **userpasskey** is the combined passcode and owner string to guarantee a 100% unique key identifier that is used to refer to each user and their kitchen going forward.

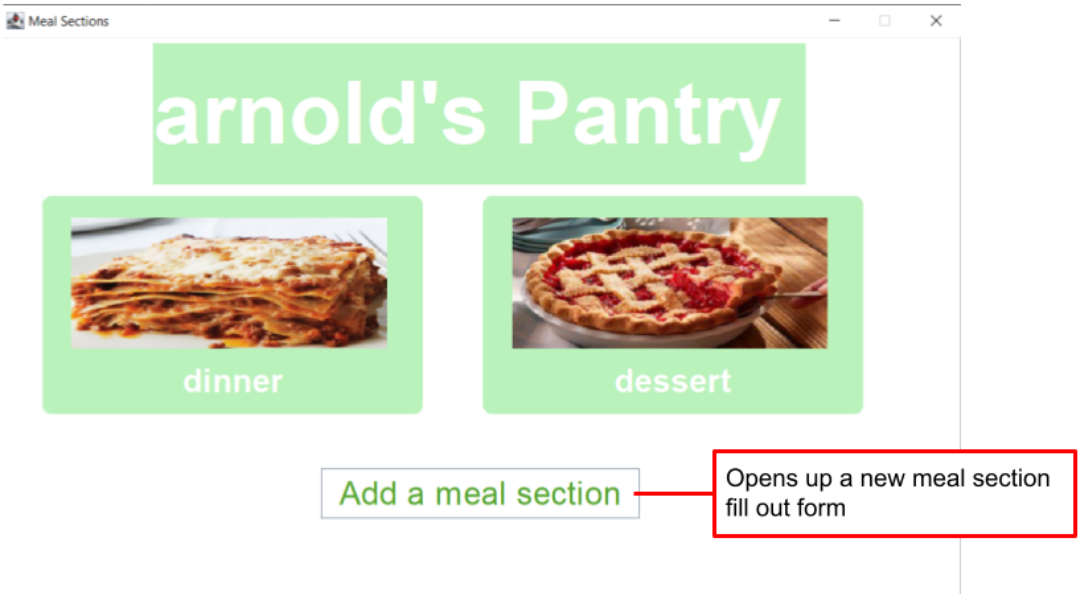
Table: users

	passcode	userscol	userpasskey
	Filter	Filter	Filter
1	1	ree	1ree
2	1	qw	1qw
3	1	na	1na
4	1	wq	1wq
5	hullo	arnold	hulloarnold
6	hello	jello	hellojello
7	1	FDSAFDSA	1FDSAFDSA

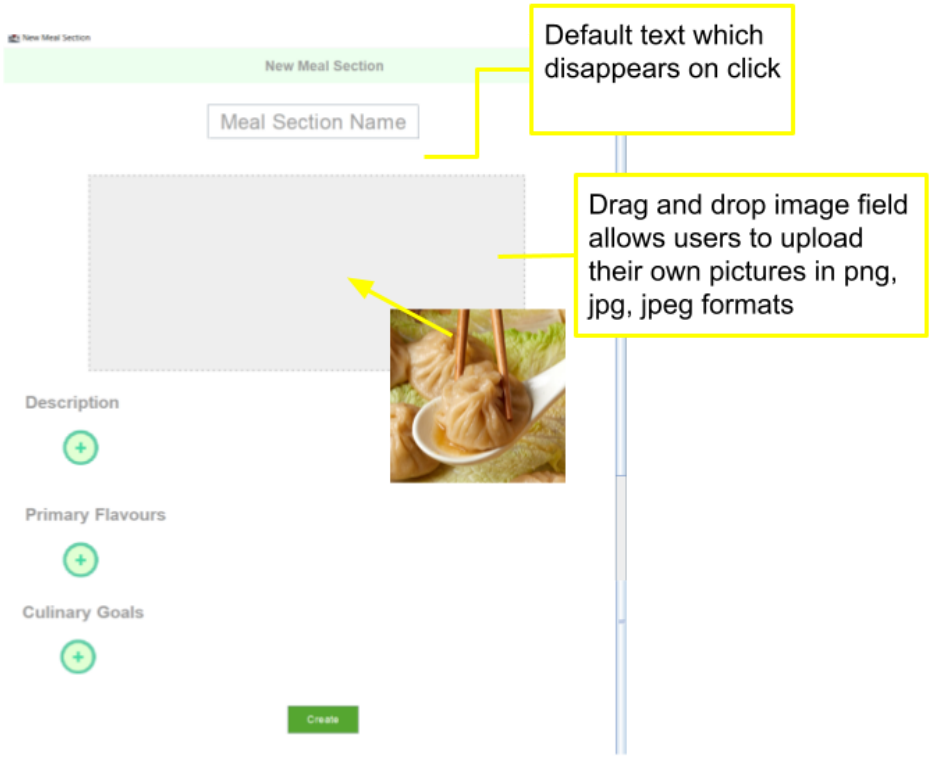
```java
public class NewAccount {

    public NewAccount(String owner, String passcode) {
        try {
            // Connect to database
            String url = "jdbc:sqlite:db.db";
            Class.forName("org.sqlite.JDBC");
            Connection con = DriverManager.getConnection(url);

            Statement stmt = con.createStatement();
            String s = "INSERT INTO USERS VALUES ('" + passcode + "','" + owner + "','" + passcode + owner + "')";
            stmt.execute(s);
            s = "CREATE TABLE IF NOT EXISTS '" + passcode + owner + "' (\n"

                    + " MealSection text DEFAULT NULL\n" + ");";

            stmt.execute(s);
            JOptionPane.showMessageDialog(null, "Success! Account created.");
            con.close();
        }
        // error message for showing if username already exists in database
        catch (SQLException e1) {
            e1.printStackTrace();
            JOptionPane.showMessageDialog(null, "Username already exists");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```



Add a meal section function (MealSectionsUnderAccount.java)
Upon successful login, the user is taken into their main pantry where meal sections are displayed.



Filling out a meal section form (MSTemplate.java)



Disappearing default text

To create the disappearing on click default text, FocusListener was used as shown:

```java
name.addFocusListener(new FocusListener() {
    // when focus is gained, and the field is either empty or set to default text,
    // clear the field
    public void focusGained(FocusEvent e) {
        if (name.getText().equals("") || name.getText().equals("Meal Section Name")) {
            name.setText("");
        }
    }

    public void focusLost(FocusEvent e) {

        // when focus is lost and there is no text, set to default text
        if (name.getText().equals("")) {
            name.setText("Meal Section Name");
        }

        // if there is text, set the meal section name to be that text after the mouse
        // leaves the field
        msName = name.getText();
    }

});
```

Image drag and drop field

Two different classes were required to support drag and drop. A **Dragger.java** class and **DragListener.java**. **Dragger.java** creates the physical instance as a JPanel and specifies the size of the field and connects to **DragListener.java** which implements the DropTargetListener interface.

DragListener.java

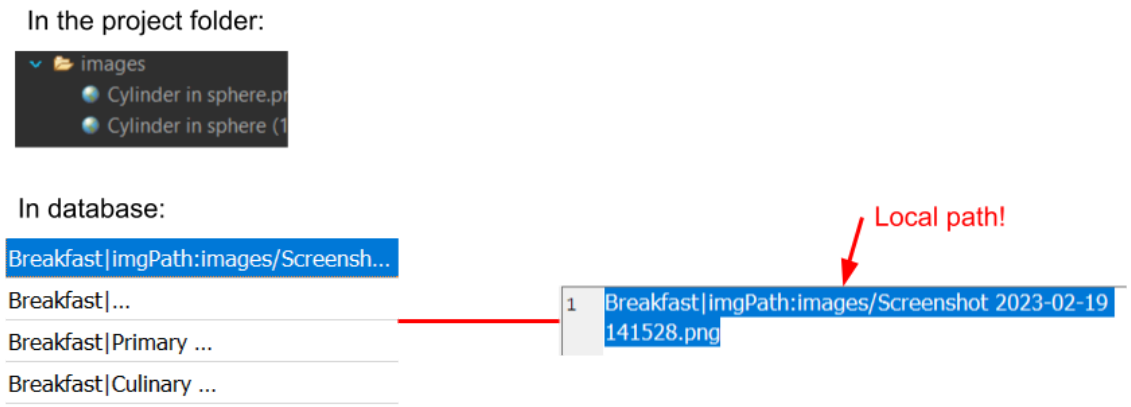
```java
public void drop(DropTargetDropEvent ev) {
    ev.acceptDrop(DnDConstants.ACTION_COPY);
    Transferable t = ev.getTransferable();

    DataFlavor[] df = t.getTransferDataFlavors();

    for (DataFlavor f : df) {
        try {
            if (f.isFlavorJavaFileListType()) {

                @SuppressWarnings("unchecked")
                List<File> files = (List<File>) t.getTransferData(f);

                for (File file : files) {

                    // copying file to local images folder using file input and output streams

                    FileInputStream in = new FileInputStream(file.getAbsolutePath());
                    Path path = Paths.get(file.getPath());
                    Path fileName = path.getFileName();
                    String s = fileName.toString();

                    // change path name
                    FileOutputStream ou = new FileOutputStream("images\\" + s);

                    BufferedInputStream bin = new BufferedInputStream(in);

                    BufferedOutputStream bou = new BufferedOutputStream(ou);
                    int b = 0;
                    while (b != -1) {
                        b = bin.read();
                        bou.write(b);
                    }
                    bin.close();
                    bou.close();

                    // update string name
                    newImagePath = "images\\" + s;

                    displayImage(newImagePath);
```

Using DataFlavour to get metadata of image. Checking if the image is of a supported type to be saved and accepted.

Buffered Input and Output streams were used to save a copy of the image in a local folder. The image path had to be first changed to the directory of this file.

Writing the image to the file

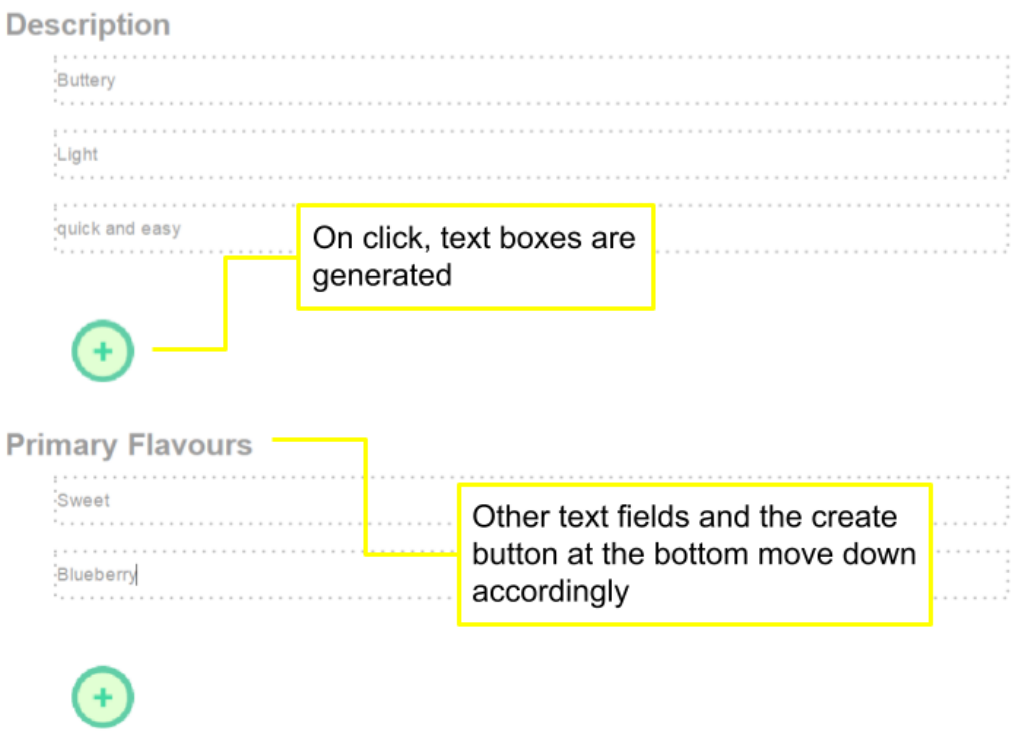
Custom method to display image after dragged



Originally, I did not use the Buffered input and output streams and simply saved a copy of the image path to the database. This worked fine when the user uploaded their own image, since the absolute path was from their own file system. However, upon further thought, I realised that in the event that the user deleted the image from their computer, accessed their account from another device, or moved it, the path saved in the database would no longer be applicable. So I found that copying the image to a local folder and saving the path from the local folder in the database to be the best solution, since the image would be available at all times no matter the location on the user's computer.

List form (ListForm.java)

I wanted a unique user experience where the user clicked on the green add button, and a text block appeared, to feel as though they were creating a list instead of just writing a regular text document. This format allowed for clear organisation, and for the user to leave some steps and descriptions blank and return to them at a later time. So, I custom developed my own format called List Form.



Before fully developing my own formatting, I did lengthy experiments with built in layout and list managers, but I heavily disliked the look and feel of them. Developing my own format proved to be one of the most difficult parts of the coding process. My solution ended up counting how many fields were created, and updating a variable responsible for spacing, and then redrawing all other elements lower according to the spacing variable.

This format was used on both the meal section and recipe forms. I had to take into account how many other sections were below the section which was adding elements, because only the ones directly below it had to move down along with the create button. The same code was used in the process of populating meal sections or recipe templates, but it was a method which accepted the string to be added as an element.

```
// For listForms on Meal Section template
else if (e.getSource() == elmntAdd && track == 2) {
    int tfLength = 770;
    int tfWidth = 900;

    TxtField field = new TxtField(tfLength);
    field.getTF().setLocation(50, 50 + spacing);
    elmntList.add(field.getTF());
    add(field.getTF());

    spacing += 60;

    // with the addition of another textfield, increase vertical spacing
    setSize(tfWidth, 150 + spacing);

    // increase overall spacing
    totalSpacingMS += 60;

    // move create button down
    int yBtn = MSTemplate.yCreate + totalSpacingMS;
    scrollCountMS = 1150 + totalSpacingMS;
    MSTemplate.container.setPreferredSize(new Dimension(0, scrollCountMS));
    MSTemplate.create.setLocation(405, yBtn);
    elmntAdd.setLocation(60, 80 + spacing);

    // if first list form is extended, move both of the following sections down
    if (formID == 0) {
        int primFY = MSTemplate.primFY + MSTemplate.descForm.getSpacing();
        MSTemplate.allForms.get(1).setLocation(20, primFY);

        int culnGY = MSTemplate.culnGY + MSTemplate.primFForm.getSpacing()
                + MSTemplate.descForm.getSpacing();
        MSTemplate.allForms.get(2).setLocation(20, culnGY);
    }

    // if second list form is extended, only move the one directly under it down
    else if (formID == 1) {
        int culnGY = MSTemplate.culnGY + MSTemplate.primFForm.getSpacing()
                + MSTemplate.descForm.getSpacing();
        MSTemplate.allForms.get(2).setLocation(20, culnGY);
    }
}
```

Checking when the green button is clicked and the type is a meal section form

Generate new TextField and add it to an arraylist of elements

Update sizing of container

```java
public void populateMSListForm(String addStr) {
    int tfLength = 770;
    int tfWidth = 900;

    TxtField field = new TxtField(tfLength);

    // add inputted text
    field.getTF().setText(addStr);
    field.getTF().setLocation(50, 50 + spacing);
    elmntList.add(field.getTF());
    add(field.getTF());
```

When recalling and displaying meal sections from the database, this method takes in the string which has been called from the database and treats it as an added element. The same code above is used to populate the rest of the page.

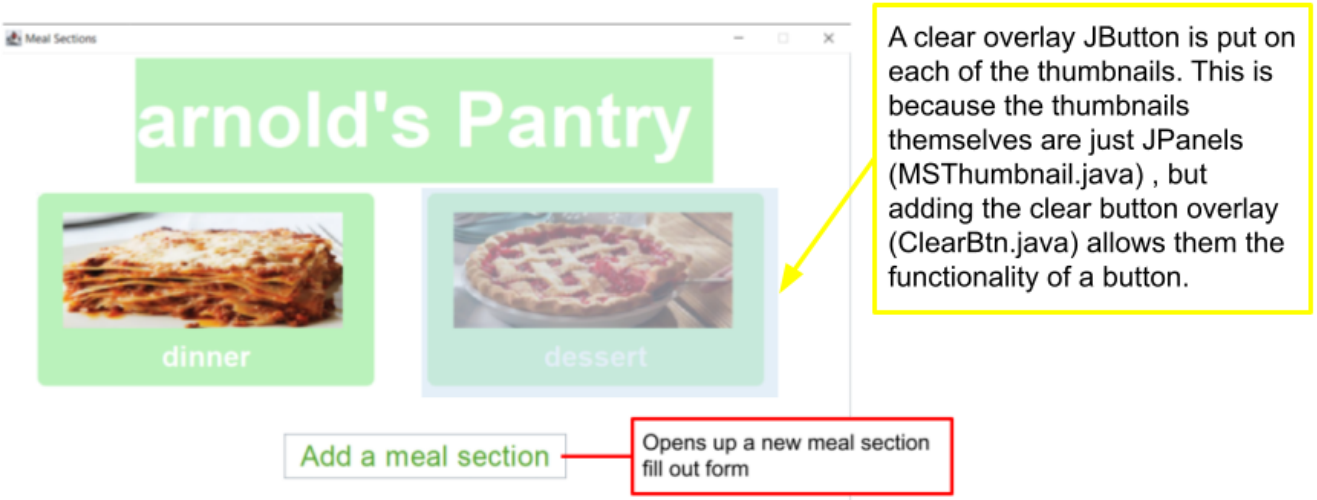
This is how the data is formatted in the database upon creation of a meal section: The code to add image and List Form data to the database will be discussed for the Recipe template shortly, as the process is very similar.

MealSection
Filter
dessert|imgPath:images/Screenshot ...
dessert|Description:For birthdays, ...
dessert|Primary Flavours:vanilla,empty
dessert|Culinary Goals:Whipped ...

MealSection is the first column in the users table. Empty listform fields are set to enter as “empty”. The cells are organized such that they begin with the meal section name, followed by a ‘|’

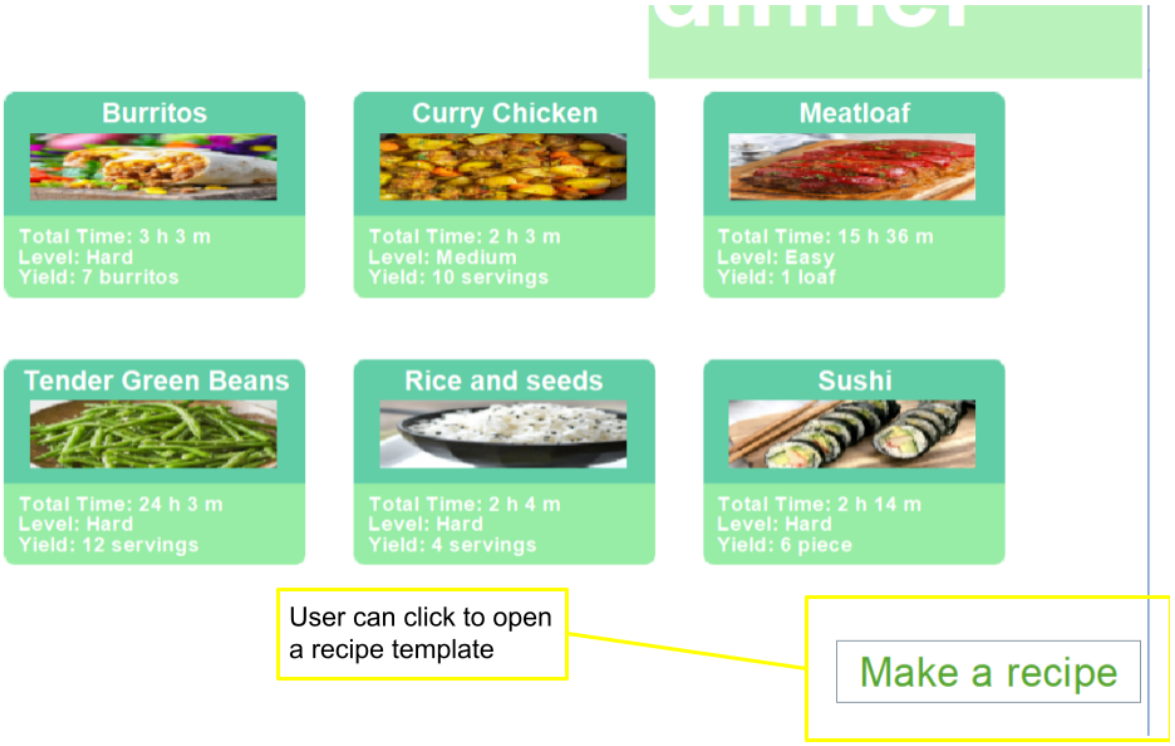
[MealSection Name] | [Label] : [Entry], [Entry]

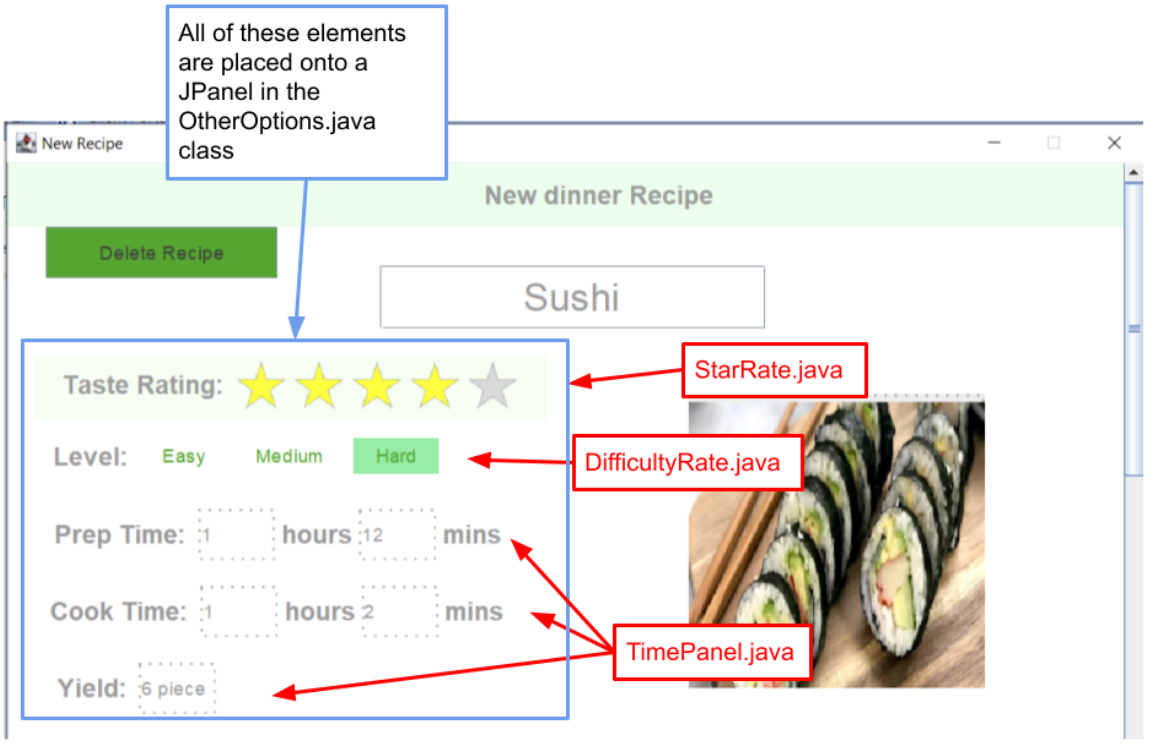
Writing recipes



A clear overlay JButton is put on each of the thumbnails. This is because the thumbnails themselves are just JPanels (MSThumbnail.java) , but adding the clear button overlay (ClearBtn.java) allows them the functionality of a button.

On click of the thumbnail, the user can view recipes under that section. And is able to write recipes with the Make a Recipe button on the bottom right of the screen.





Not pictured: Ingredients, substitutions, instructions, notes

Taste rating

The user can click stars and the stars up to the selected star will turn yellow, while the stars after will remain grey. The stars themselves are made in **StarBtn.java** which extends JButton and the bar is made in **StarRate.java** which extends from JPanel**.**

StarRate.java

```java
// generate 5 StarBtns and add to starArr
for (int i = 0; i < 5; i++) {
    StarBtn star = new StarBtn();
    starArr.add(star);
    add(star);
}
```

An arraylist **starArr** holds the 5 StarBtn objects.

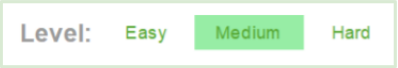
StarBtn.java

```java
public void mouseClicked(MouseEvent e) {

    btn = new ImageIcon("star.png");
    selectedStar = starID;
    for (int i = 0; i <= starID; i++) {
        // colour all stars up to the selected star as yellow
        StarRate.starArr.get(i).setIcon(btn);
    }
    btn = new ImageIcon("emptyStar.png");
    for (int i = starID + 1; i < 5; i++) {
        // colour all stars following the selected stars grey
        StarRate.starArr.get(i).setIcon(btn);
    }
}
```

Each star is given a starID which is an integer from 0 to 4. Everytime a StarBtn is clicked, the clicked star is identified and a loop up until that star will make the stars have a yellow icon and all those after are changed to the grey star image.

Level bar

The level bar makes the difficult buttons in **LevelBtn.java** and the bar is assembled as a JPanel in **DifficultyRate.java.**



DifficultyRate.java

```java
private LevelBtn easy;
private LevelBtn med;
private LevelBtn hard;
public static ArrayList<LevelBtn> btnList;
```

LevelBtn is a class which extends JButton. It takes the name to display on the button as an argument

Buttons are stored in an arraylist

Constructor…

```java
easy = new LevelBtn("Easy");
med = new LevelBtn("Medium");
hard = new LevelBtn("Hard");
```

LevelBtn.java

```java
public void mouseClicked(MouseEvent e) {
    // ensures colour persists only on selected button
    selectedBtn = lvlID;
    setBackground(Color.decode("#97EDA5"));
    for (int i = 0; i < 3; i++) {
        if (i != lvlID) {
            DifficultyRate.btnList.get(i).setBackground(Color.white);
        }
    }
    repaint();
}
```

Sets the clicked button to have a green background (selected mode)

Loop through all other buttons and force them to have a white background (deselected)

Prep and cook time and yield panel

These panels come from the same class called TimePanel.java. There is a unique constructor for the yield panel since it operates slightly differently than the two time panels.

```java
// label specifies cook or prep time
public TimePanel(String label) {

    setBackground(Color.white);
    setLayout(new FlowLayout());
    lab = new JLabel(label);
    lab.setFont(new Font("Arial", Font.BOLD, 20));
    lab.setForeground(Color.decode("#9B9B9B"));
    add(lab);

    fHours = new TxtField(60);
    this.fHours.getTF().setPreferredSize(new Dimension(60, 40));
    add(fHours.getTF());

    hours = new JLabel("hours");
    hours.setFont(new Font("Arial", Font.BOLD, 20));
    hours.setForeground(Color.decode("#9B9B9B"));

    add(hours);

    fMins = new TxtField(60);
    this.fMins.getTF().setPreferredSize(new Dimension(60, 40));
    add(fMins.getTF());

    mins = new JLabel("mins");
    mins.setFont(new Font("Arial", Font.BOLD, 20));
    mins.setForeground(Color.decode("#9B9B9B"));

    add(mins);
}
```

Input either “Cook” or “Prep” time when creating an object of TimePanel.java

Creates instance of custom TxtField. Java class. Writes the label of hours and mins. For yield constructor, there is only one TxtField and it omits the hours/mins label.

```java
// setter methods for time and yield
public void setTime(String hours, String mins) {
    fHours.getTF().setText(hours);
    fMins.getTF().setText(mins);
}
```

When the create button is pressed, setTime() is called on the TimePanel object and a method in TxtField called getTF() is called which returns the java TextField which then gets set to the times given.

These components, including the image drag and drop field, are placed onto a single JPanel and formatted in the **OtherOptions.java** class. In this class, there are methods for populating the fields with previously selected data.

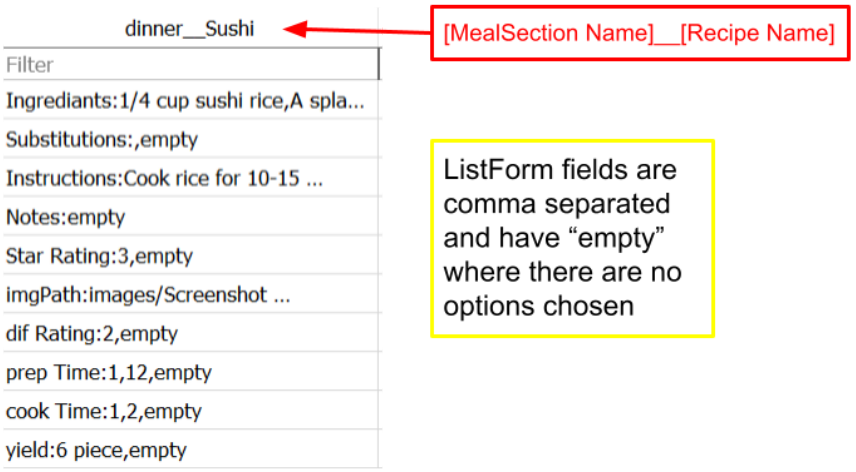
OtherOptions.java

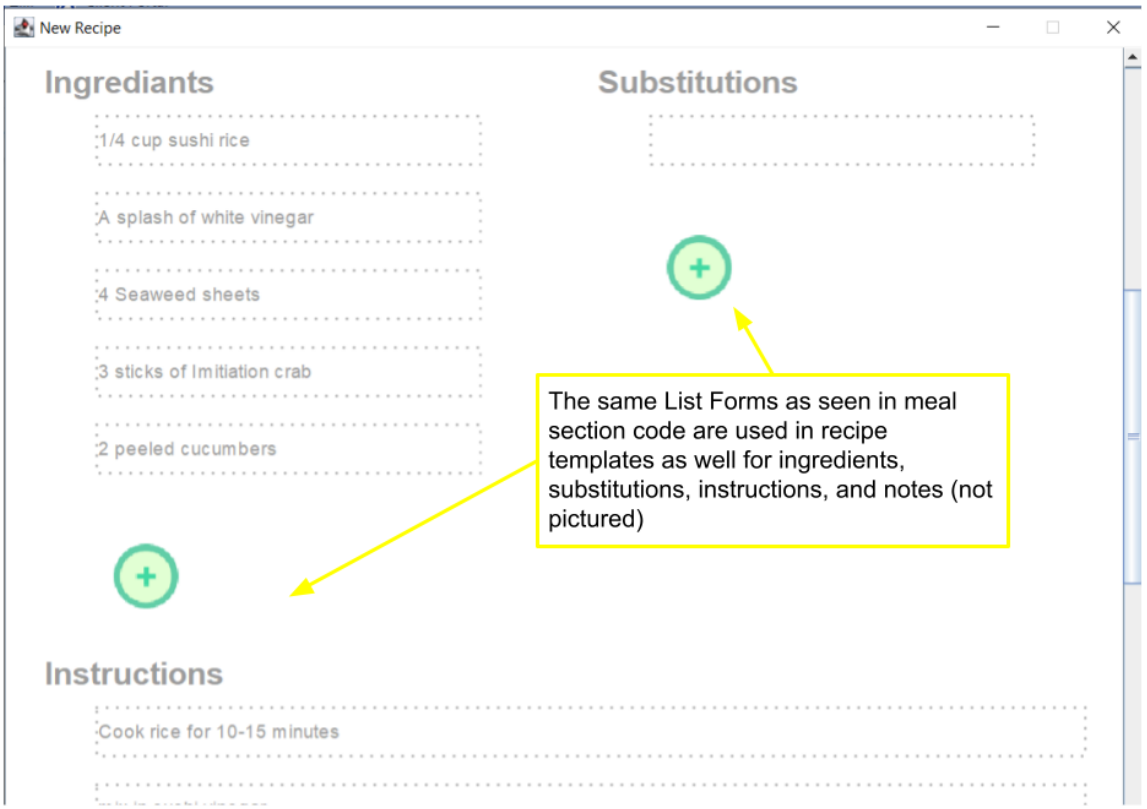
This method is called when a recipe is opened. Data is read from the database and passed as arguments into this method.

```java
// method for populating a recipe template with existing recipe option
// selections
public void populateRecipeListForm(String starRate, String imgPath, String difRate, String pTimeHours,
        String pTimeMins, String cTimeHours, String cTimeMins, String yield) {
    StarBtn.selectedStar = Integer.parseInt(starRate);
    rateBar.repaintStarBar();

    imgField.loadImg(imgPath);

    LevelBtn.selectedBtn = Integer.parseInt(difRate);
    difBar.repaintSelectedDiff();

    prep.setTime(pTimeHours, pTimeMins);
    cook.setTime(cTimeHours, cTimeMins);
    this.yield.setYield(yield);
}
```

Values are set and elements are updated accordingly

When the user clicks the “create” button at the bottom of the page, the data is loaded into the database in the following manner:





When the create button is clicked, methods on the ListForms are called which format the queries and send the data to the database. Learning how to use a database was by far the most difficult and time consuming piece of this entire IA, and was a large reason as to why other tasks could not be completed in the given time period.

RecipeTemplate.java

Checks if only numbers are entered into prep and cook time. A valid recipe name does not have certain characters which are not allowed in the database, and it must be a unique name to that meal section.

```
// if all numeric required values are correct and
// proceed by sending data to db
if (numCheck && validRecipName) {
    try {

        String url = "jdbc:sqlite:db.db";
        try {
            Class.forName("org.sqlite.JDBC");
        } catch (ClassNotFoundException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        Connection con = DriverManager.getConnection(url);
        Statement stmt = con.createStatement();
        // __ distinguishes meal section name from recipe name

        if (type == 1) {
            colName = msName.replaceAll(" ", "_") + "__" + RecipeTemplate.recipName.replaceAll(" ", "_");
            String query = "ALTER TABLE '" + CreateLoginForm.currUser + "' ADD '" + colName + "' text NULL";
            stmt.execute(query);

            ingForm.addInfoToDB();
            subForm.addInfoToDB();
            instructForm.addInfoToDB();
            noteForm.addInfoToDB();
            ListForm.addOptsInfo();

            // when a new recipe is created after one has already been created in the same
            // session,
            // static counter totalRecip is to be reset and the arrayList is cleared so at
            // to preserve the spacing
            // and reformatting system
            ListForm.totalRecip = 0;

            ClearBtn.ks.deleteKS();

            ClearBtn.ks = new RecipesUnderMS(msName);
            frame.dispose();
```

Connect to database

Formats recipe name as [MealSection Name]__[Recipe Name] and adds as a column to user table

Methods in ListForm that add data to database

ListForm.java

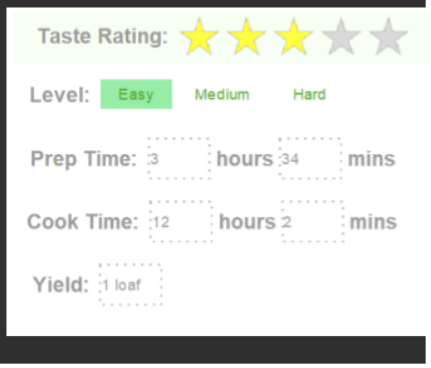
```java
// method for formatting data and adding to the db
// I want data to enter the db as such:
// [MealSectionName]: ing1,ing2,ing3
public void addInfoToDB() {
    String s = header + ":";
    if (elmntList.size() > 0) {
        for (JTextField t : elmntList) {
            if (t.getText() != "") {
                s += t.getText() + ",";
            }

        }
    } else {
        s += "empty";
    }
    elmntList.clear();

    try {
        String url = "jdbc:sqlite:db.db";
        try {
            Class.forName("org.sqlite.JDBC");
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        Connection con = DriverManager.getConnection(url);
        Statement stmt = con.createStatement();
        stmt.execute(formatQuery(s));
        con.close();
    } catch (SQLException e1) {
        JOptionPane.showMessageDialog(null, "Recipe already defined");
        e1.printStackTrace();
    }
}
```

Formats the string which will be entered into the database

Method which writes the query

ListForm.java

```java
// method for adding data from the block of star rating, image, difficulty
// rating, times, and yield
public static void addOptsInfo() {
    String starRate = "Star Rating:" + ((StarRate) OtherOptions.optArr.get(0)).getSelectedStar() + ",";

    String imgPath = "imgPath:" + ((Dragger) OtherOptions.optArr.get(1)).getImagePath() + ",";

    // replace \\ with /
    imgPath = imgPath.replaceAll((char) 92 + "" + (char) 92, (char) 47 + "");

    String difRate = "dif Rating:" + ((DifficultyRate) OtherOptions.optArr.get(2)).getSelectedDiff() + ",";

    String prepTime = "prep Time:" + ((TimePanel) OtherOptions.optArr.get(3)).getHours() + ","
            + ((TimePanel) OtherOptions.optArr.get(3)).getMins() + ",";

    String cookTime = "cook Time:" + ((TimePanel) OtherOptions.optArr.get(4)).getHours() + ","
            + ((TimePanel) OtherOptions.optArr.get(4)).getMins() + ",";

    // make so that yield and all other textfields cannot add commas (unnecessary
    // and messes up retrieval from db)

    String yield = "yield:" + ((TimePanel) OtherOptions.optArr.get(5)).getYield() + ",";

    // add data to db
    try {
        String url = "jdbc:sqlite:db.db";
        try {
            Class.forName("org.sqlite.JDBC");
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        Connection con = DriverManager.getConnection(url);
        Statement stmt = con.createStatement();

        stmt.execute(formatQuery(starRate));
        stmt.execute(formatQuery(imgPath));
        stmt.execute(formatQuery(difRate));
        stmt.execute(formatQuery(prepTime));
        stmt.execute(formatQuery(cookTime));
        stmt.execute(formatQuery(yield));
```

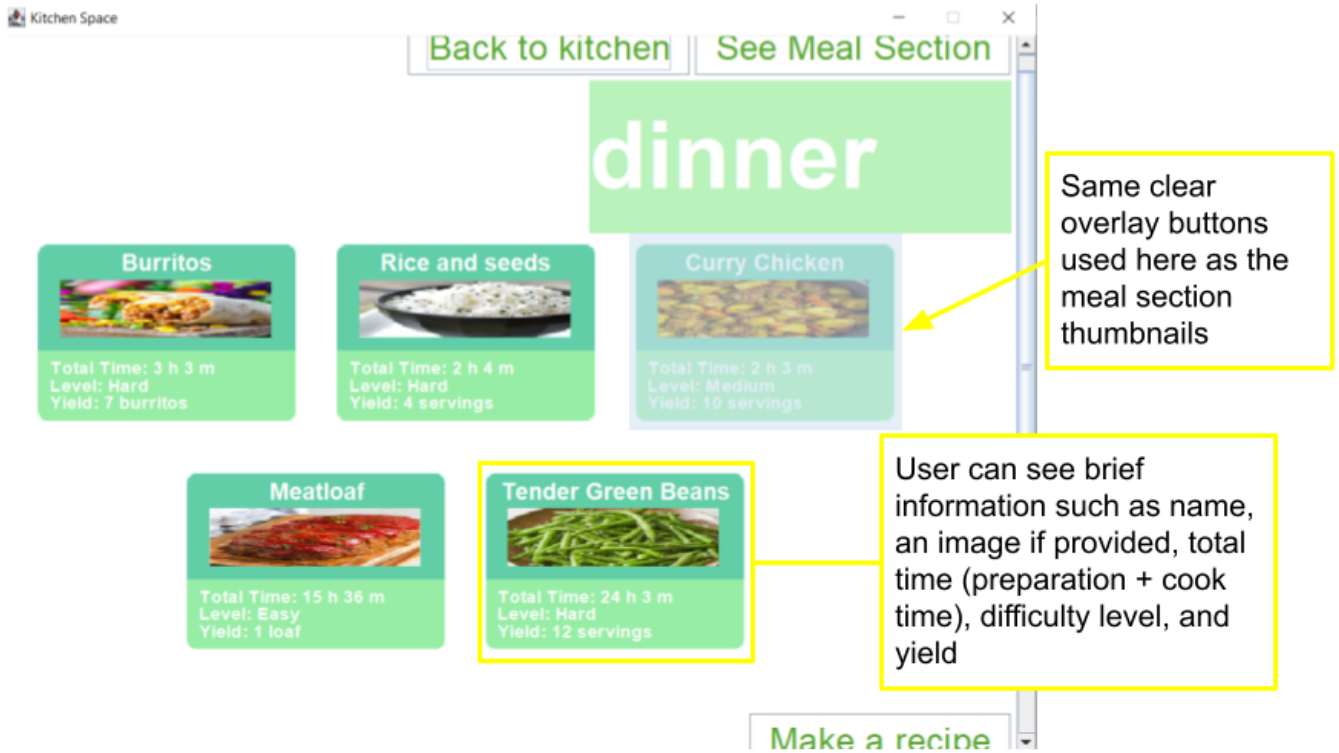


ListForm.java

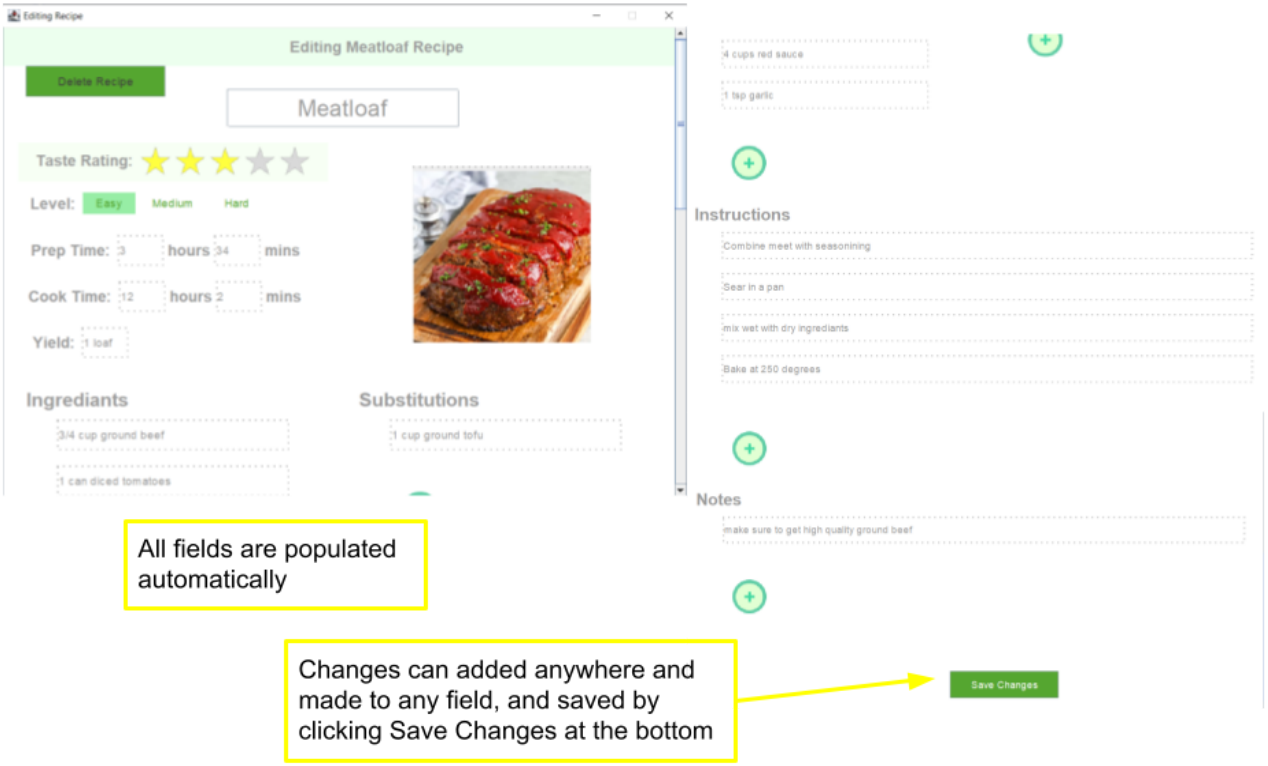
```java
// method for preparing the query that sends info to db
public static String formatQuery(String s) {

    // for empty values
    if (s.charAt(s.length() - 1) == ':' || s.charAt(s.length() - 1) == ',') {
        s.substring(0, s.length() - 2);
        s += "empty";
    }
    // making updates to a recipe, just insert the data into the same recipe
    if (RecipeTemplate.editingRecip) {
        return "INSERT INTO '" + CreateLoginForm.currUser + "' ('" + RecipeTemplate.newColName + "') VALUES ('" + s + "')";

    }

    // if making a new recipe,
    if (track == 1) {
        return "INSERT INTO '" + CreateLoginForm.currUser + "' ('" + RecipeTemplate.colName + "') VALUES ('" + s + "')";
    }
    // if making a new meal section
    String temp = MSTemplate.msName + "|";
    temp += s;
    s = temp;
    return "INSERT INTO '" + CreateLoginForm.currUser + "' (MealSection) VALUES ('" + s + "')";

}
```

Viewing and editing existing recipes

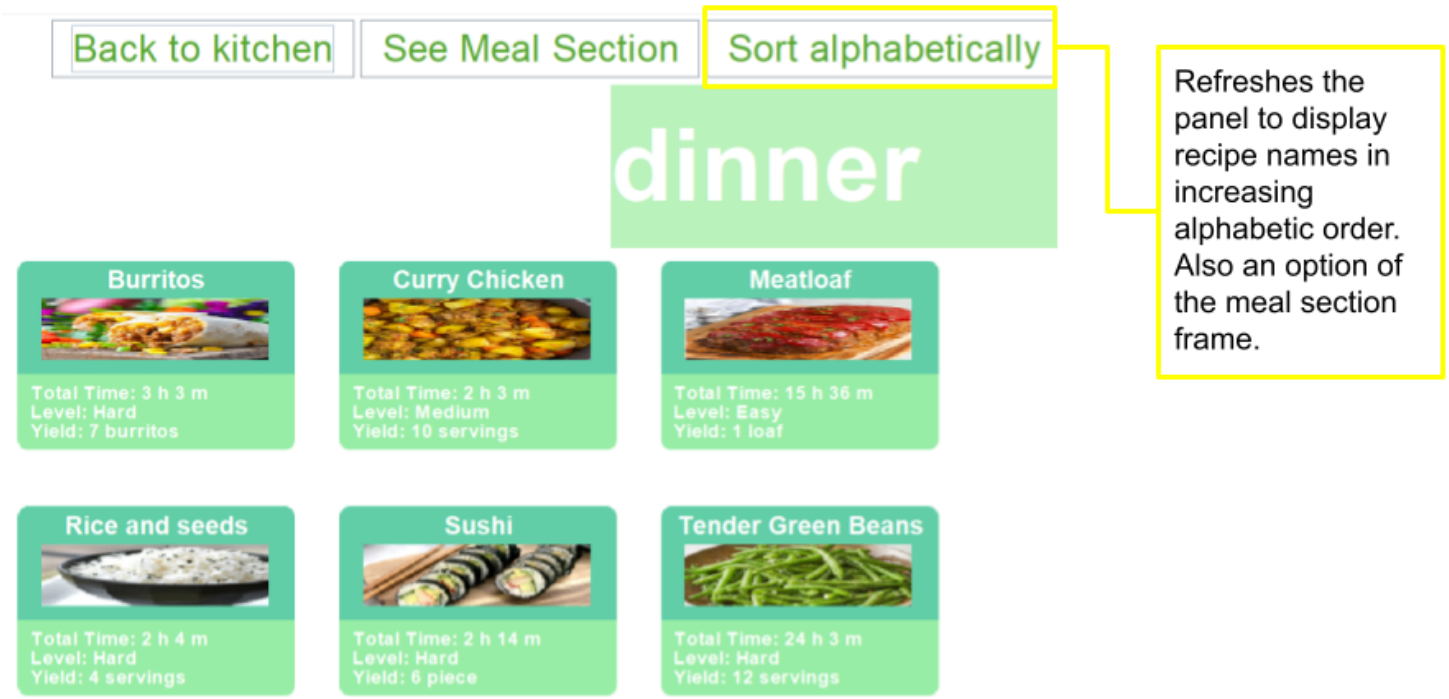
The user can click on the thumbnails of the meals to view them. This opens a recipe template, and methods are called which populate them with data retrieved from the database.



Saving changes follows the same logic as adding new recipe data to the database, except it does not create another column. It instead uses the existing column, but drops all its data and inserts everything again.



Sorting meal sections and recipes by alphabetical order



RecipeDataRetrieval.java

```
RecipeThumbnail[] arr = new RecipeThumbnail[recipArr.size()];
// sort alphabetically

for (int i = 0; i<recipArr.size(); i++) {
    arr[i] = recipArr.get(i);
}
```

After retrieving data from database, the thumbnails of each recipe are made in RecipeThumbnail.java. An arraylist of thumbnails is collected and added to an array to be sorted before displaying them on the panel.

Insertion sort

```
if (RecipesUnderMS.sortSelected) {
    for (int i = 1; i<recipArr.size(); i++) {
        RecipeThumbnail var = arr[i];
        int j = i-1;
        while (j>=0 && var.getName().compareToIgnoreCase(arr[j].getName())<0) {
            arr[j+1] = arr[j];
            j--;

        }
        arr[j+1] = var;

    }
    RecipesUnderMS.sortSelected = false;
}

// add each thumbnail to container
for (int i = 0; i<recipArr.size(); i++) {
    add(arr[i]);
}
```

Insertion sort was used since it runs in O(N) in the best case.

Word count: 1000

Techniques used:

- Inheritance
- Interface
- Sorting
- Arraylist
- Database