# Agenda

- Planning
- Requirements
- Design
- Application Development
- Testing
- Summary

# Planning

- Our first major step was to code a working calculator since the functions are the same as a cash register
- The application should show the numbers that was clicked
- The application should perform calculation with the information that was provided to it
- The application should display the result

# Planning

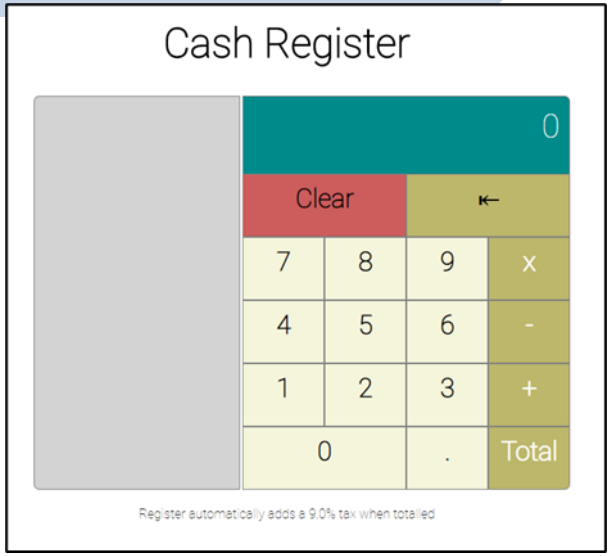

Image of generic cash register application

# Requirements

- User Interface
  - Number pads (0 - 9)
  - Operator buttons (-, +, x)
  - Equation buttons that calculates (Total, Cash, Credit Card)
  - Key Butttons (Clear, backspace)
  - Display screens (Top, Bottom, Left)

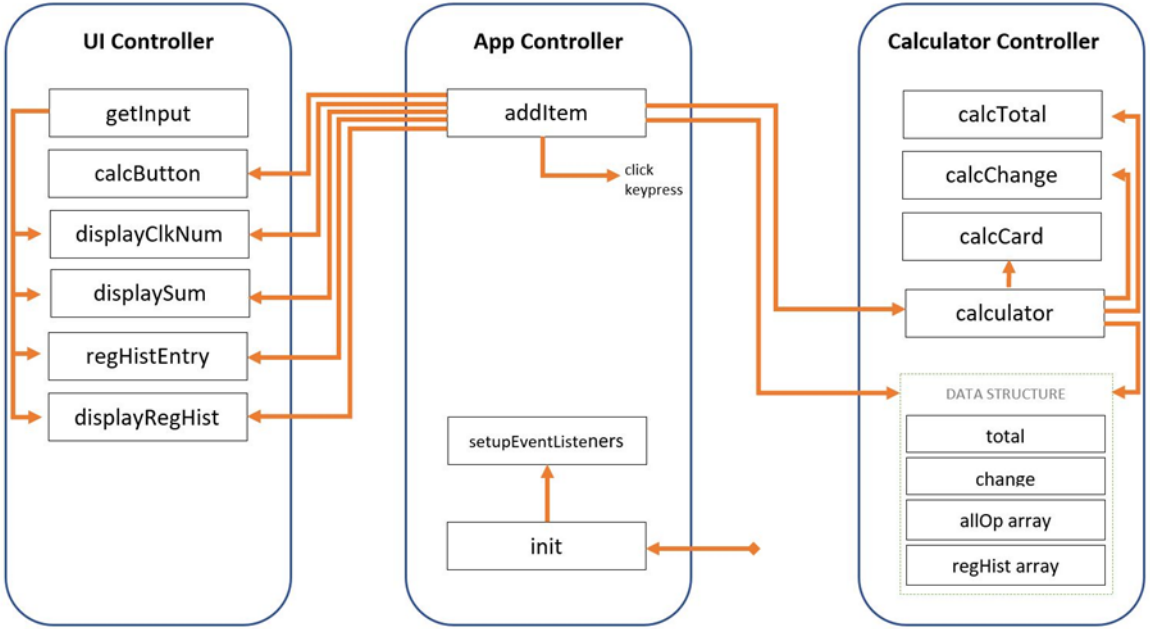
- Functions
  - Total
  - Change
  - Credit Card
  - Calculator
  - Display Clicked Number
  - Display total and subtotal
  - Transaction Entry
  - Display Transaction History

# Design




Screenshot of user interface design

# Design




Low Level Architecture Design

# Application Development

- This was the longest phase of our SDLC model
- The first thing we did was create the user interface to build a simple calculator image with div tags as buttons separated by rows
- Next, to test the capabilities of the team, we first started with a simple calculator
- After that, as planned, we separated all codes into three modules: Interface Module, Data Module and Controller Module
- Controller Module is a delegator that tells Data Module (calcCtrl Module) to execute some function and update Data Structure. It also tells Interface Module (uiCtrl Module) to display information on the screens

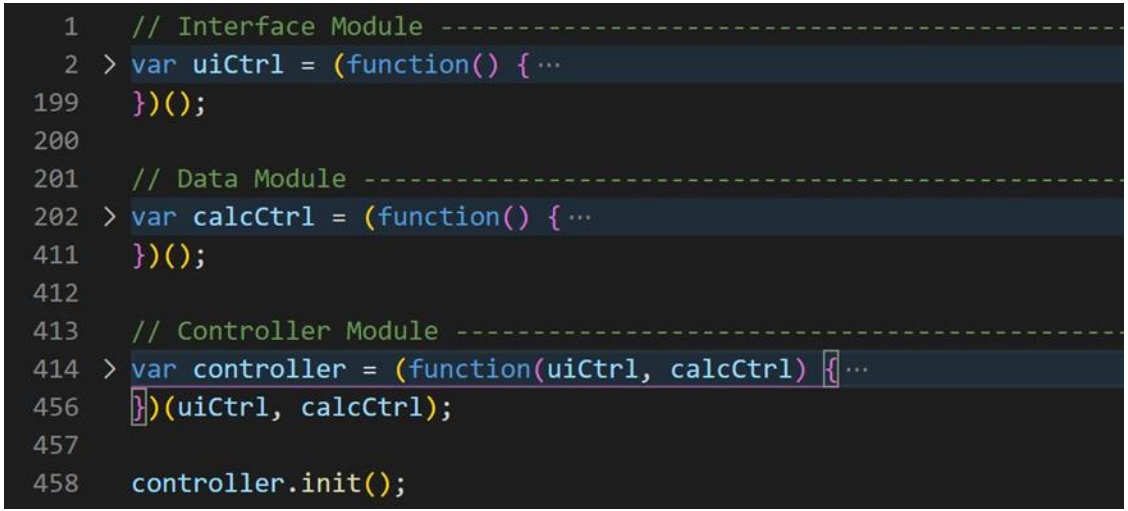
# Application Development

```
</div>

<div class="reg-parent">
    <div class="row">
        <div class="column" id="reg-display-val">0</div>
    </div>

    <div class="row">
        <div class="reg-btn column" id="reg-clear">Clear</div>
        <div class="reg-btn column" id="reg-backspace"> &#8676; </div>
    </div>

    <div class="row">
        <div class="reg-btn reg-btn-num column" id="reg-seven">7</div>
        <div class="reg-btn reg-btn-num column" id="reg-eight">8</div>
        <div class="reg-btn reg-btn-num column" id="reg-nine">9</div>
        <div class="reg-btn reg-btn-operator column" id="reg-times">x</div>
    </div>

    <div class="row">
        <div class="reg-btn reg-btn-num column" id="reg-four">4</div>
        <div class="reg-btn reg-btn-num column" id="reg-five">5</div>
        <div class="reg-btn reg-btn-num column" id="reg-six">6</div>
        <div class="reg-btn reg-btn-operator column" id="reg-minus">-</div>
    </div>

    <div class="row">
        <div class="reg-btn reg-btn-num column" id="reg-one">1</div>
        <div class="reg-btn reg-btn-num column" id="reg-two">2</div>
        <div class="reg-btn reg-btn-num column" id="reg-three">3</div>
        <div class="reg-btn reg-btn-operator column" id="reg-plus">+</div>
    </div>

    <div class="row">
        <div class="reg-btn reg-btn-num column" id="reg-zero">0</div>
        <div class="reg-btn column" id="reg-decimal">.</div>
        <div class="reg-btn reg-btn-operator column" id="reg-equals">Total</div>
    </div>
</div>

<div>
    <p>Register automatically adds a 9.0% tax when totalled</p>
</div>
```

```
.main_container {
    margin-top: 10px;
    width: 100%;
    height: calc(90vh - 30px);
    display: flex;
    flex-direction: row;

}

.reg-history{
    width: 30%;
    padding: 10px;
    border-radius: 1.5rem;
    border-top: 4px solid rgb(124, 124, 124);
    border-left: 00.5px solid rgb(124, 124, 124);
    border-right: 00.5px solid rgb(124, 124, 124);
    background: linear-gradient(180deg, rgb(145, 145, 145) 0%, rgb(209, 209, 209) 35%, rgb(209, 209, 209) 100%);
    text-align: right;
    overflow: hidden;
}

.reg-history .regInfoContainer {
    height: 100%;
}

.reg-history span {
    font-size: clamp(1rem, 2vw, 2rem);
}

.reg-parent{
    margin-top: -0.5rem; /* negate border-spacing: 0.5rem;*/
    margin-left: 10px;
    width: calc(70% + 1rem);
    height: calc(100% + 1rem); /* negate border-spacing: 0.5rem;*/
}
```

Screenshot of HTML and CSS

# Application Development




Screenshot shows how Interface Module and Data Module communicates through Controller Module

# Application Development

Controller Module

- Controller Module contains two functions:
- addItem
  - Designed to call Data Module and tell the module to take user input calculate them and update the data structure
  - It also calls Interface Module, supplies the module with data structure and tells the module to display the information on the screen
- setupEventListerners
  - Contains the event listener method that tracks clicked button and provides that information to function addItem

# Application Development

Data Module

- It is designed for two main purpose, host a data structure, and do all the heavy calculation of a calculator
- Functions
  - Calculate Total
  - Calculate Change
  - Calculate Credit Card
  - Calculator: Execute a certain task depending on the user input types.
  - There are four different types and they are number, operator, equation and key.

# Application Development

Interface Module

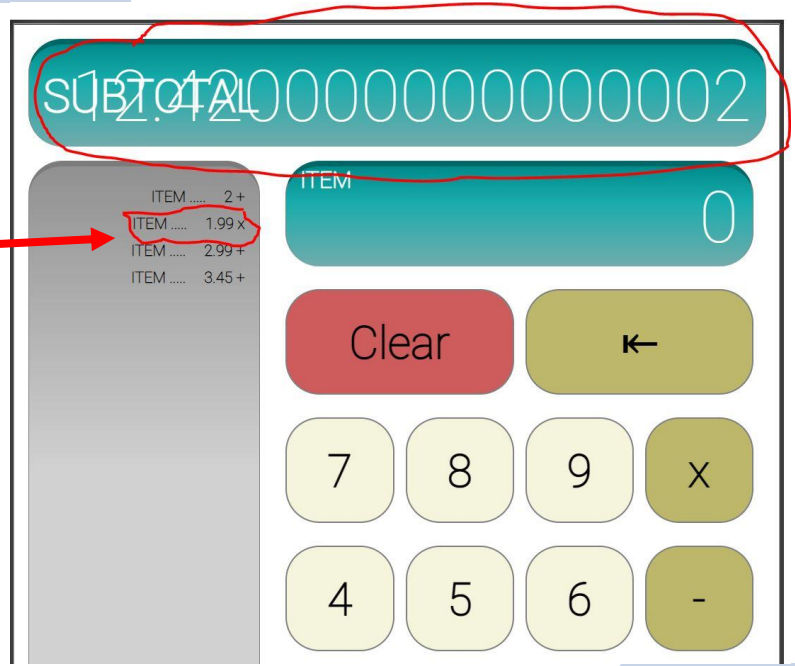
- Interface Module (uiCtrl Module) deals with getting the current value and updating the interface
- It contains functions and objects
- Objects
  - calcButton contains properties like name, type and formula
  - getInput grabs DOM elements which is used to manipulate HTML content
- Functions
  - Display clicked Number
  - Display sum
  - Push all transaction into array
  - Display register History

# Testing

- Testing was performed at various phases of the application development lifecycle
- Unit testing verifies the functionality of a unit of code
- Integration testing assesses the interface between application components
- System testing is the overall testing of the application system
- Regression testing assures that each new fix does not break anything that was previously working
- Acceptance testing is often performed by the customers themselves but in our case team member tested each others code

# Testing

- Unit Testing
- Integration Testing




- Unit testing Solution
  - toFixed() method was used
- Integration testing Solution
  - Correct information was fed to Interface Module

# Summary

Our team used HTML to create the static page, added CSS to produce a great visual design, and finally added the JavaScript to bring it all to life