Criterion C: Development

Introduction:

The integrated Development Environment (IDE) used to develop this application for both Android and IOS is Visual Studio Code. The framework used is Google's Flutter SDK. Which is based upon Dart Programming Language. The GUI was also created within Visual Studio Code through code. External Data is stored within a SQL database, provided by Google's Firebase.

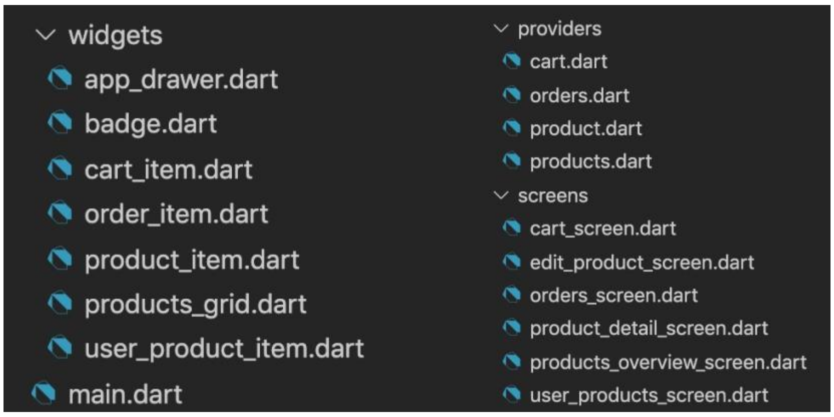
Techniques used to create Product:

- Object Oriented Programming
- Encapsulation & Inheritance
- Reading and Writing to JSON database
- Data Validation
- Animation
- Authentication & Verification
- Algorithmic Thinking
- Programming GUI
- Data Structures
- Libraries [1]

Object Oriented Programming:

Dart is an Object-Oriented Programming Language; the reason Dart was utilized is that it allows to generate a GUI based on custom and pre-existing widgets. Object oriented programming allows me to separate different areas of my application into its own category, this makes the program more organized which is easier for maintenance.

[1] Check Appendix for Libraries used.



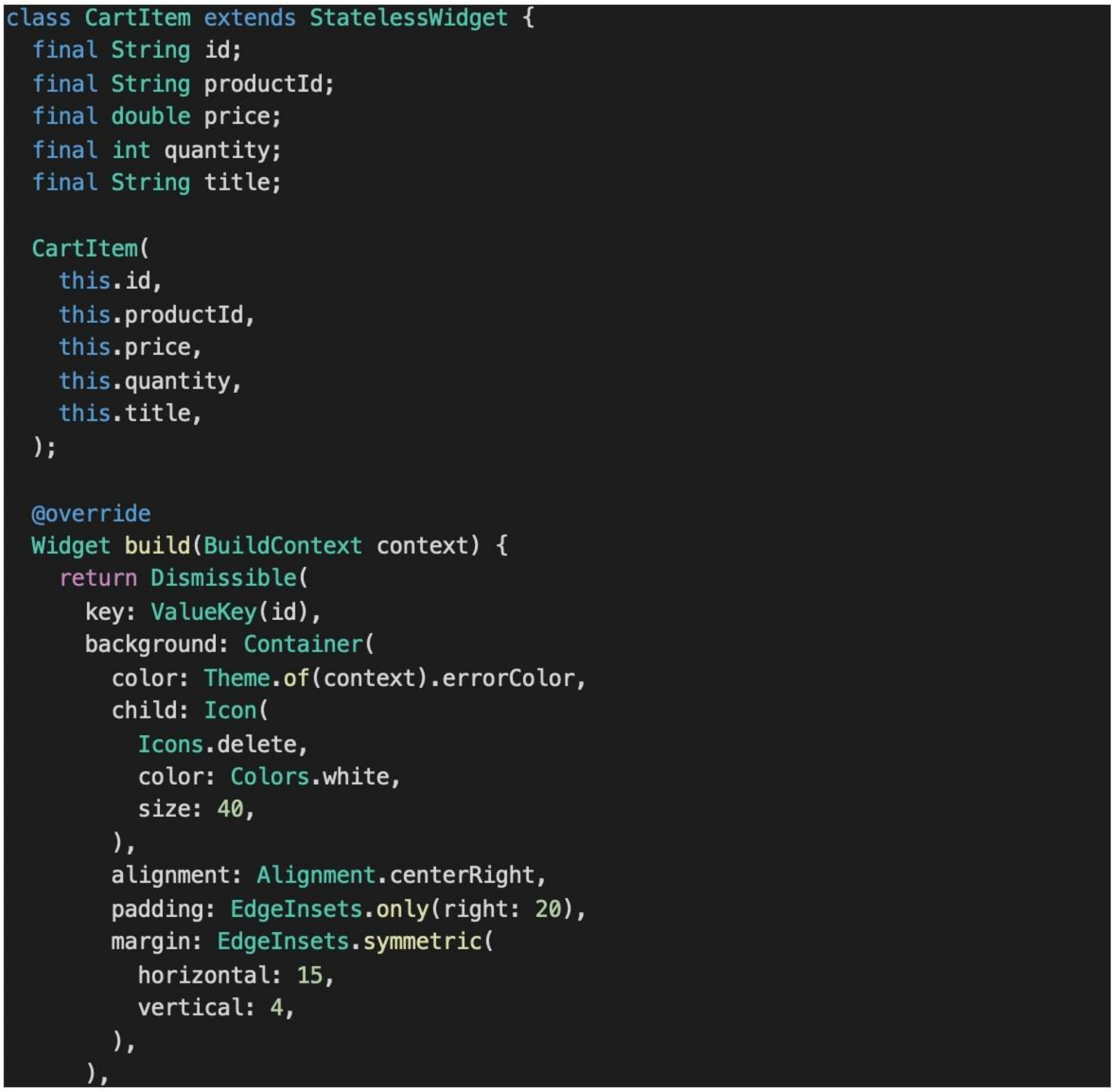
```
widgets
    app_drawer.dart
    badge.dart
    cart_item.dart
    order_item.dart
    product_item.dart
    products_grid.dart
    user_product_item.dart
main.dart
```

```
providers
    cart.dart
    orders.dart
    product.dart
    products.dart
screens
    cart_screen.dart
    edit_product_screen.dart
    orders_screen.dart
    product_detail_screen.dart
    products_overview_screen.dart
    user_products_screen.dart
```

Showcasing file organization)

(Picture 1:

Encapsulation & Inheritance:



```
class CartItem extends StatelessWidget {
  final String id;
  final String productId;
  final double price;
  final int quantity;
  final String title;

  CartItem(
    this.id,
    this.productId,
    this.price,
    this.quantity,
    this.title,
  );

  @override
  Widget build(BuildContext context) {
    return Dismissible(
      key: ValueKey(id),
      background: Container(
        color: Theme.of(context).errorColor,
        child: Icon(
          Icons.delete,
          color: Colors.white,
          size: 40,
        ),
        alignment: Alignment.centerRight,
        padding: EdgeInsets.only(right: 20),
        margin: EdgeInsets.symmetric(
          horizontal: 15,
          vertical: 4,
        ),
      ),
```

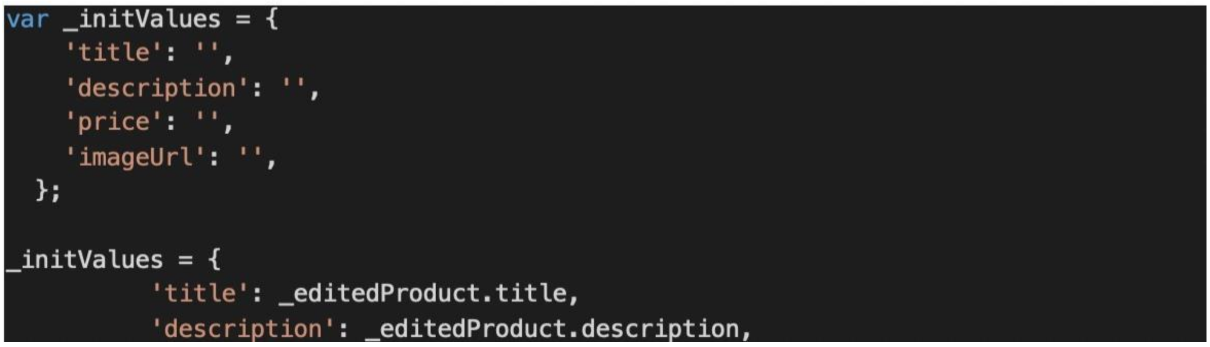
(Picture 2: Showcasing Parent and Child widgets)

Encapsulation is shown through the code in @override. This is required to be used within flutter as it allows me to change the default properties of the widget to add desired characteristics, allowing me to meet the client's needs.

There is also a presence of public and private values, values presented as "this." Represent private values that are only available within that file. Final state is also used in variables that shouldn't be changed and are initialized in that way.

Inheritance is used to create a new class from an existing class, this is occurring when the class CartItem is create based on a parent class called StatelessWidget, this is an internal widget provided by the SDK.

Initialization

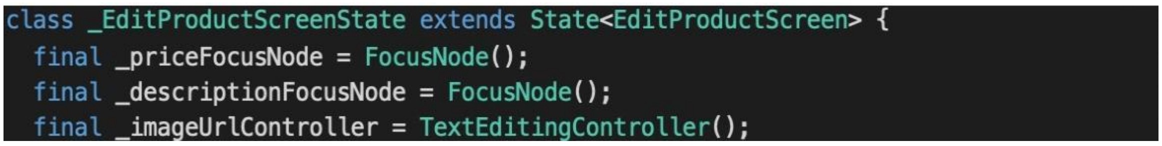


```
var _initValues = {
    'title': '',
    'description': '',
    'price': '',
    'imageUrl': '',
  };

_initValues = {
          'title': _editedProduct.title,
          'description': _editedProduct.description,
```

(Picture 3: Initialization of values for items)

Initialization was utilized in the system to be able to create placeholders for the information required in order to add an item. Default initial values are added when the program is initialized. When the _initValues is called a new item can be added by filling all the placeholders created.

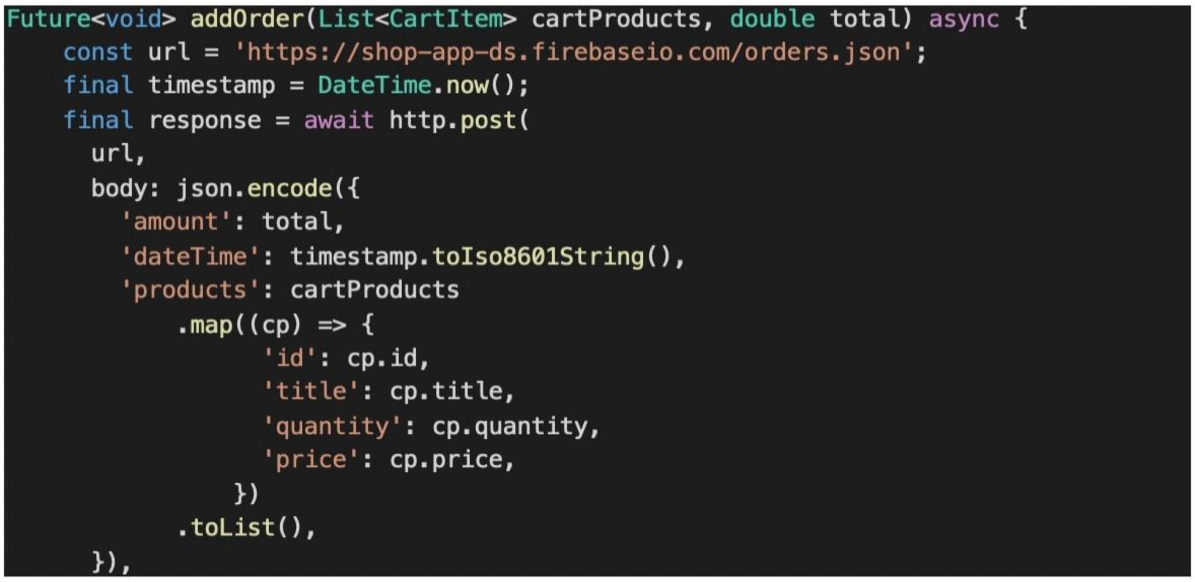


```
class _EditProductScreenState extends State<EditProductScreen> {
  final _priceFocusNode = FocusNode();
  final _descriptionFocusNode = FocusNode();
  final _imageUrlController = TextEditingController();
```

(Picture 4: Extending State Widget)

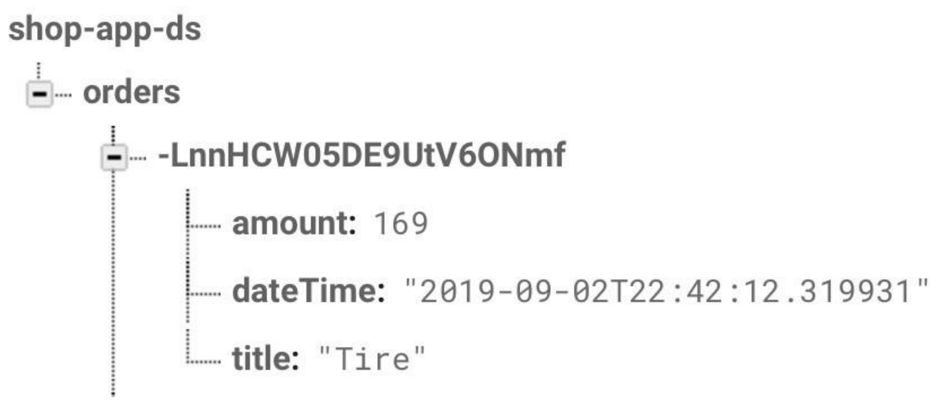
The other form of initialization is by setting the initial properties of a variable. Final means that the content of the variable or the type of variable cannot be changed.

Reading and Writing to JSON Database:



```
Future<void> addOrder(List<CartItem> cartProducts, double total) async {
    const url = 'https://shop-app-ds.firebaseio.com/orders.json';
    final timestamp = DateTime.now();
    final response = await http.post(
      url,
      body: json.encode({
        'amount': total,
        'dateTime': timestamp.toIso8601String(),
        'products': cartProducts
            .map((cp) => {
                  'id': cp.id,
                  'title': cp.title,
                  'quantity': cp.quantity,
                  'price': cp.price,
                })
            .toList(),
      }),
```

(Picture 5: Establishing connection with database)

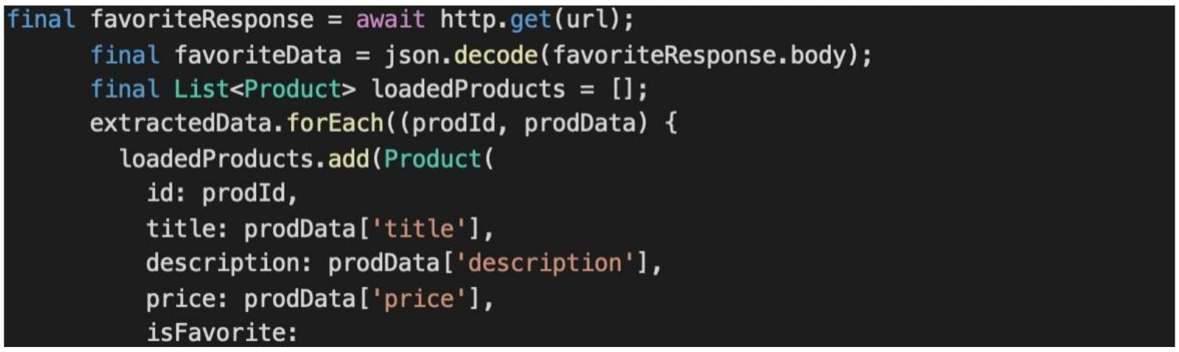


```
shop-app-ds
  orders
    -LnnHCW05DE9UtV6ONmf
        amount: 169
        dateTime: "2019-09-02T22:42:12.319931"
        title: "Tire"
```

(Picture 6: Showcasing data input into database)

The database used to store data such as; product title, description, price, imageURL. This is stored on a JSTOR-database that is provided my Google's Firebase. This approach was chosen since it is a fast, reliable database stored online which is a requirement for the application.

Data is added to the database using the HTML library which allows to send post requests, then it uses a JSON encoder to communicate with the database. The screenshot shoes the orders database with the Primary Key, and the data added successfully. The reason the screenshot is missing the quantity is because it Is a single item.



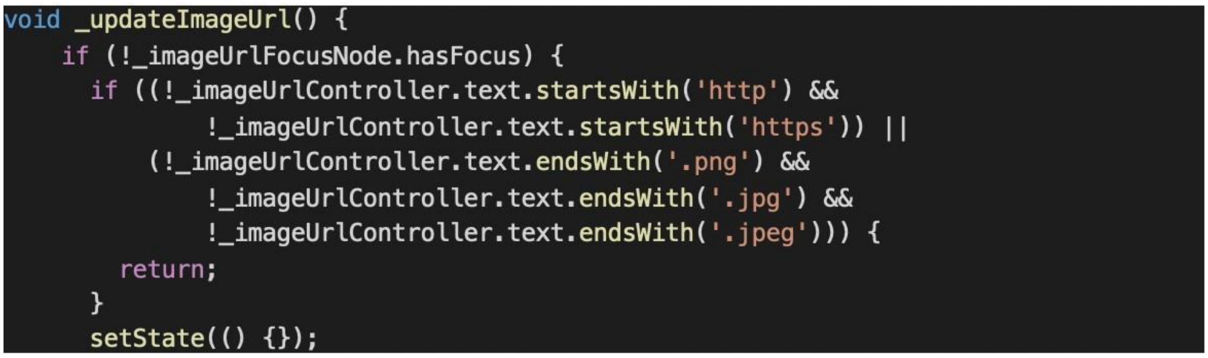
```
final favoriteResponse = await http.get(url);
      final favoriteData = json.decode(favoriteResponse.body);
      final List<Product> loadedProducts = [];
      extractedData.forEach((prodId, prodData) {
        loadedProducts.add(Product(
          id: prodId,
          title: prodData['title'],
          description: prodData['description'],
          price: prodData['price'],
          isFavorite:
```

(Picture 7: GET used to receive information from database)

Above shows how the http request changed from POST to GET to be able to retrieve the information from the database. We are using the Json decoder again. We are fetching the data we want based on the title of the column on the database.

Data Validation:

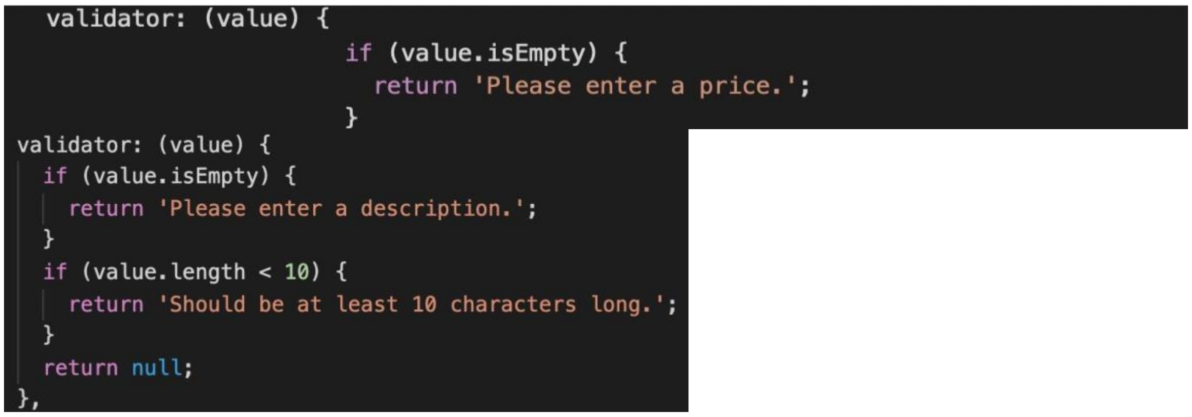
Data validation is used to ensure that data redundancy is minimized in terms of duplicate accounts. It is also to ensure that data enters the database correctly.



```
void _updateImageUrl() {
    if (!_imageUrlFocusNode.hasFocus) {
      if ((!_imageUrlController.text.startsWith('http') &&
              !_imageUrlController.text.startsWith('https')) ||
          (!_imageUrlController.text.endsWith('.png') &&
              !_imageUrlController.text.endsWith('.jpg') &&
              !_imageUrlController.text.endsWith('.jpeg'))) {
        return;
      }
      setState(() {});
```

(Picture 8: Validate URL format)

This is an example of data validation occurring, it is used to check the image URL the user entered. This also means that the website only accepts only 2 formats of images, JPG and PNG. If the input doesn't meet these requirements, then it won't allow the change/addition to be made to the database. This is to ensure that the URL inputted by the user is functional.

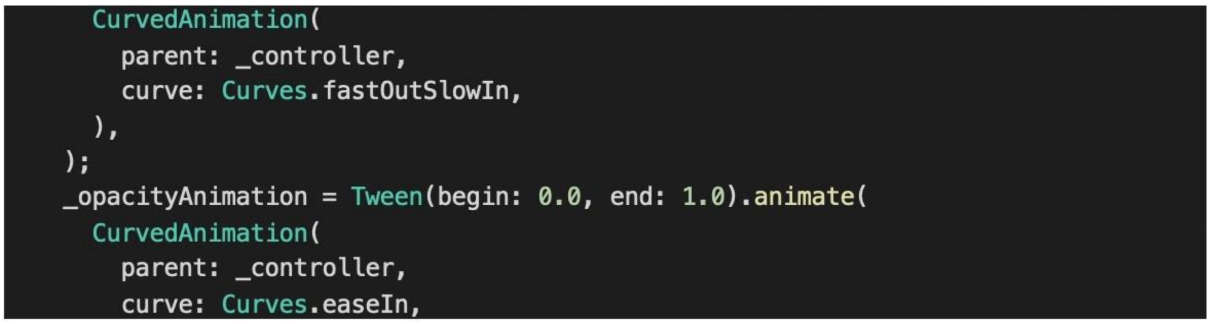


```
  validator: (value) {
                        if (value.isEmpty) {
                          return 'Please enter a price.';
                        }
validator: (value) {
  if (value.isEmpty) {
    return 'Please enter a description.';
  }
  if (value.length < 10) {
    return 'Should be at least 10 characters long.';
  }
  return null;
},
```

(Picture 9: Set character limit for user input)

This is another example where it checks the checkbox and to ensure that all fields are filled in before submitting an entry to the database. This is useful since all columns in the database cannot be NULL.

Animations:

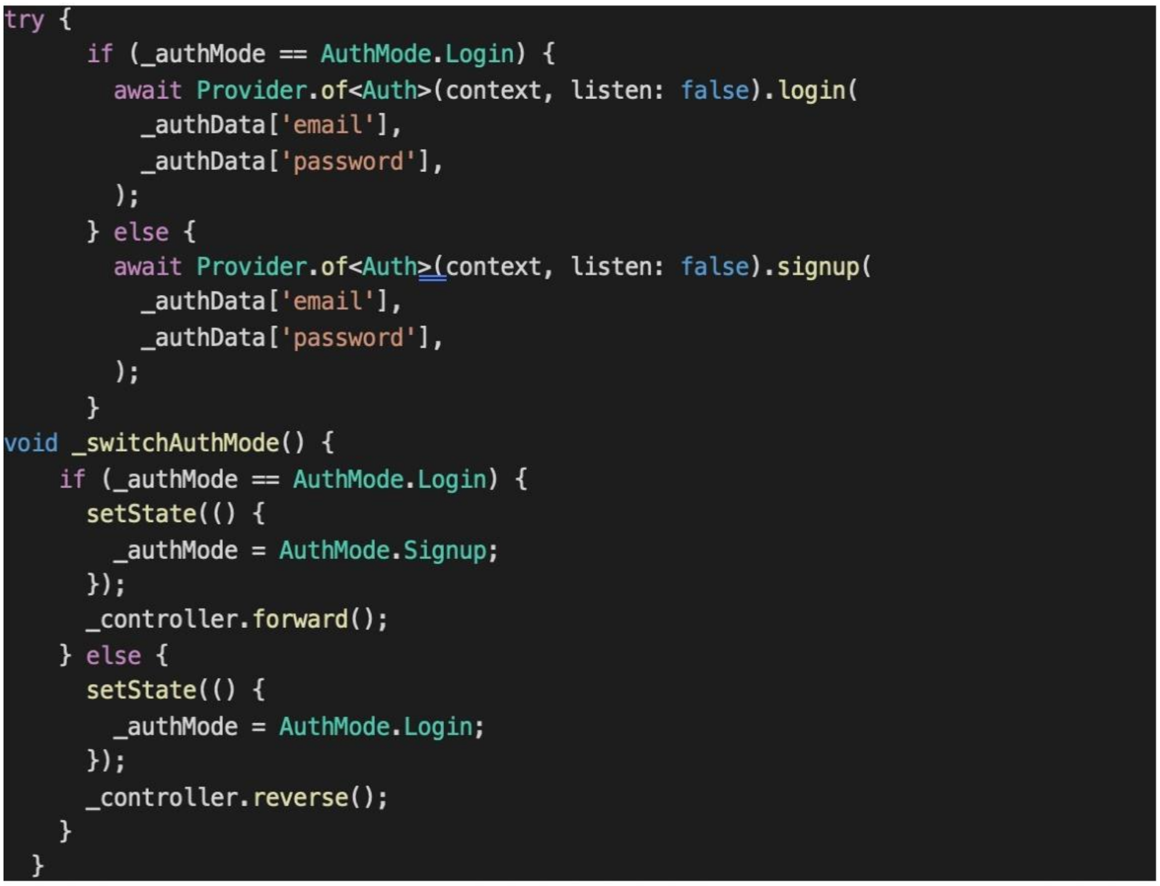


```
      CurvedAnimation(
        parent: _controller,
        curve: Curves.fastOutSlowIn,
      ),
    );
    _opacityAnimation = Tween(begin: 0.0, end: 1.0).animate(
      CurvedAnimation(
        parent: _controller,
        curve: Curves.easeIn,
```

(Picture 10: Set type of animation and duration)

Animations are utilized within the application to ensure a better user-experience while utilizing apps. Custom animations allow to control duration. An example of an animation is fastOutSlowIn, this is a function within flutter. But by default, it has its own duration, this was changed from to a duration of 1. A controller is also required to control the animations and to know when they should be triggered. Animations are sort of like a custom function, it can be called when a button is pressed or a state changes.

Authentication & Verification:

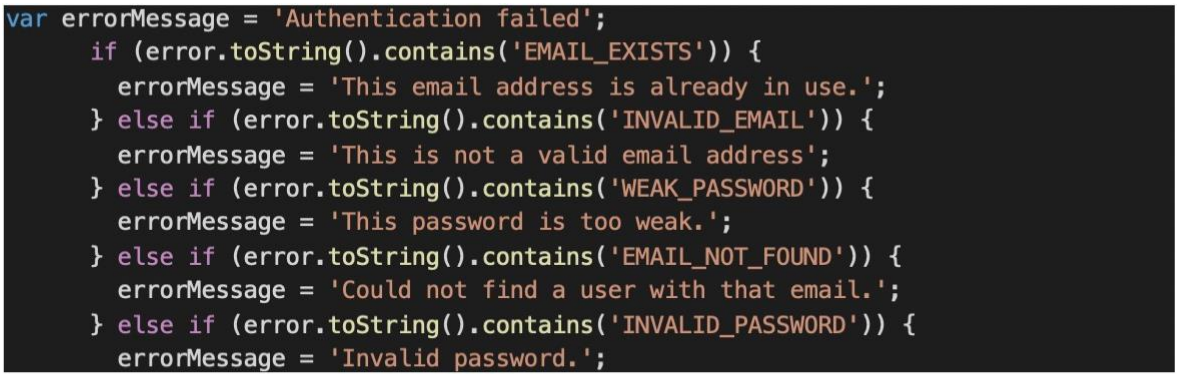


```
try {
      if (_authMode == AuthMode.Login) {
        await Provider.of<Auth>(context, listen: false).login(
          _authData['email'],
          _authData['password'],
        );
      } else {
        await Provider.of<Auth>(context, listen: false).signup(
          _authData['email'],
          _authData['password'],
        );
      }
void _switchAuthMode() {
    if (_authMode == AuthMode.Login) {
      setState(() {
        _authMode = AuthMode.Signup;
      });
      _controller.forward();
    } else {
      setState(() {
        _authMode = AuthMode.Login;
      });
      _controller.reverse();
    }
  }
```

(Picture 11: Authenticating variables entered by user)

Authentication and verification are conducted by a function called AuthMode which is provided by the Flutter IDE. This function AuthMode checks the inputted email and password with what is stored in the database. If the information is correct, then it changes the state of the screen on the controller. If Data is not correct, then the user can create a new account that will be sent to the database. The function **listen** is what communicates with the database and waits for a response back. Based on this response back it is what activates the rest of the script.

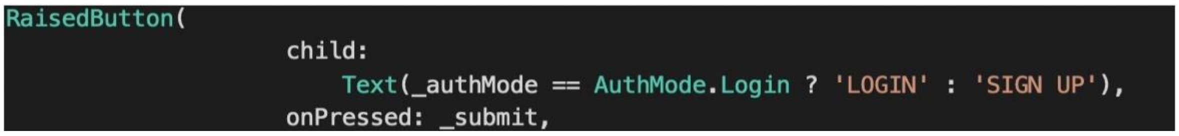
Algorithmic Thinking



```
var errorMessage = 'Authentication failed';
      if (error.toString().contains('EMAIL_EXISTS')) {
        errorMessage = 'This email address is already in use.';
      } else if (error.toString().contains('INVALID_EMAIL')) {
        errorMessage = 'This is not a valid email address';
      } else if (error.toString().contains('WEAK_PASSWORD')) {
        errorMessage = 'This password is too weak.';
      } else if (error.toString().contains('EMAIL_NOT_FOUND')) {
        errorMessage = 'Could not find a user with that email.';
      } else if (error.toString().contains('INVALID_PASSWORD')) {
        errorMessage = 'Invalid password.';
```

(Picture 12: Show error message based on the incorrect input)

Algorithmic Thinking is conducted in the example with the utilization of nested if statements, which means that an action triggers another if statement. This is done as a form to show the user the error messages dependent on the issue they are discovering.



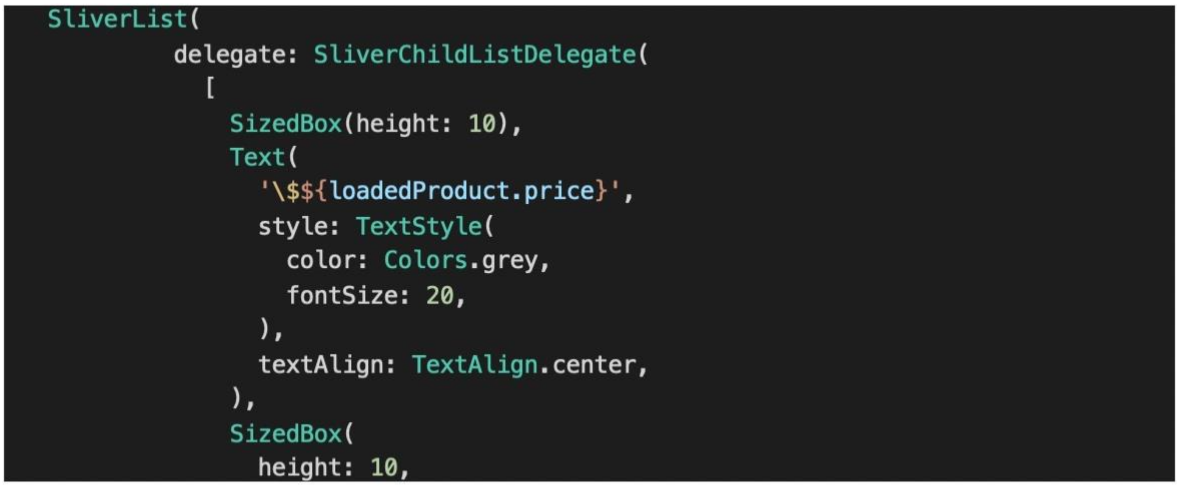
```
RaisedButton(
                    child:
                        Text(_authMode == AuthMode.Login ? 'LOGIN' : 'SIGN UP'),
                    onPressed: _submit,
```

(Picture 13: Showcasing OnPressed to add functionality to button)

The only form of conditions that were used is if statements, the reason for not using for or while is because no loops were utilized within this application. If statements were utilized more towards validation and to trigger something based on an action. For button’s if statements weren’t utilized as there was a onPressed function that did that same thing as an if statement.

Programming GUI:

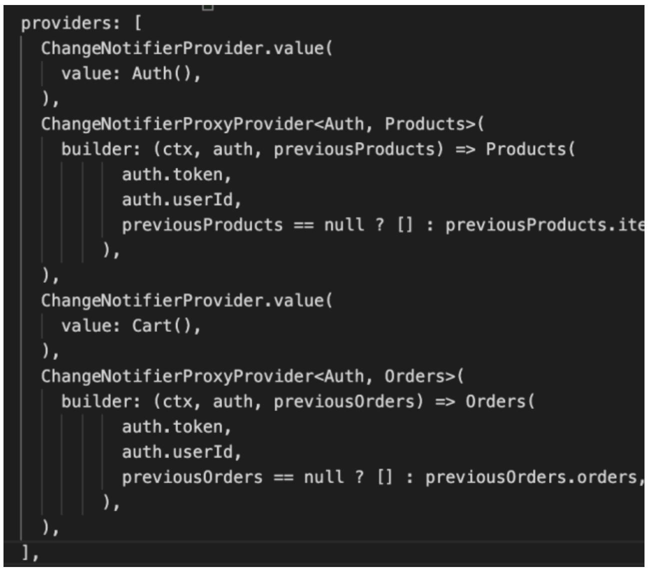
Many IDE’s contain a GUI for the developer to be able to input the assets they would like to show on the screen. Flutter and Dart is different as there is no GUI everything has to be hard programmed; Flutter’s screen is organized by widgets on the screen, based on their order in the codebase. The only way to change the look of a widget or a screen is by changing the code properties associated to the widget.



```
   SliverList(
            delegate: SliverChildListDelegate(
              [
                SizedBox(height: 10),
                Text(
                  '\$${loadedProduct.price}',
                  style: TextStyle(
                    color: Colors.grey,
                    fontSize: 20,
                  ),
                  textAlign: TextAlign.center,
                ),
                SizedBox(
                  height: 10,
```

(Picture 14: Styling for widget)

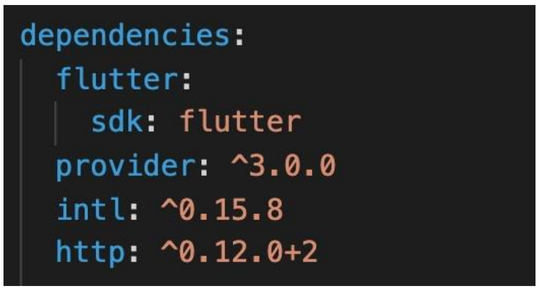
Data Structures:



```
providers: [
  ChangeNotifierProvider.value(
    value: Auth(),
  ),
  ChangeNotifierProxyProvider<Auth, Products>(
    builder: (ctx, auth, previousProducts) => Products(
          auth.token,
          auth.userId,
          previousProducts == null ? [] : previousProducts.ite
        ),
  ),
  ChangeNotifierProvider.value(
    value: Cart(),
  ),
  ChangeNotifierProxyProvider<Auth, Orders>(
    builder: (ctx, auth, previousOrders) => Orders(
          auth.token,
          auth.userId,
          previousOrders == null ? [] : previousOrders.orders,
        ),
  ),
],
```

(Picture 15: One Dimensional Array being utilized to list the providers needed to load when the application opens up.)

Libraries:



```
dependencies:
  flutter:
    sdk: flutter
  provider: ^3.0.0
  intl: ^0.15.8
  http: ^0.12.0+2
```

(Picture 16: Showcasing the libraries used)

There was a total of 4 libraries that were utilized. The Flutter SDK[1] that provided to utilize the stateless and stateful widgets that show up on the screen. The provider[2] allows for state management within the widget, it is what allows to update information on the screen. Intl[3] is used for data and time formatting. Http[4] allows for external connection with the database.

Word Count: 1048

1 https://flutter.dev/docs/development/tools/sdk/releases
2 https://pub.dev/packages/provider
3 https://pub.dev/packages/intl
4 https://pub.dev/packages/http