

## PART 1: Project Setup

### Step 1: Clone the project

Clone the github repo containing the boiler plate code:

```
git clone https://github.com/zuliatowoade/divvy-up-starter.git
cd <git-project-directory>
```

### Step 2: Open the Project in Your IDE

Open the project in your favorite IDE (IntelliJ, VSCode, etc.).

Ensure your Gradle dependencies are resolved by running the following in the terminal:

```
./gradlew build
```

## PART 2: Setup Data Access (ExpenseDataManager)

### Step 1: Review Data Model Classes

Review content of the data model class, user, expense, CreateExpenseAllocationRequest

## PART 3: Setup Services (ExpenseDataManager)

### Step 1: Review Friends Service

Review `getFriendsByInitiatorName()` method

### Step 2: Update Friends Service

Add The following code to the `getFriendsByInitiatorName()` method

```
public List<User> getFriendsByInitiatorName(String username) {
    var result = friendsRepository.findByInitiatorName(username);
    List<User> response = new ArrayList<User>();
    for (Friends friend : result) {
        response.add(friend.getFriend());
    }
    return response;
}
```

### Step 3: Update Expense Service

Review expense service (saveExpense())

### Step 4: Update ExpenseAllocation Service

Review expense service (saveExpense())

## PART 4: Setup Controllers (ExpenseDataManager)

### Step 1: Update Expense Controller

Add the following code to the **createExpense()** method

```
@PostMapping
public ResponseEntity<Expense> createExpense(@RequestBody
Expense expense) {
    return
    ResponseEntity.ok(expenseService.saveExpense(expense));
}
```

### Step 2: Update Friends Controller

Add the following code to the **createExpense()** method

```
@GetMapping("/{username}")
public List<User> getAllFriends(@PathVariable String username) {
    return friendsService.getFriendsByInitiatorName(username);
}
```

### Step 3: Review User Controller

Review the UserController class

## PART 5: Setup Data Access (ExpenseManager)

### Step 1: Review Data Model Classes

Review content of the data model class, expense, CreateExpenseAllocationRequest

## PART 6: Setup Services (ExpenseManager)

### Step 1: Review Expense Service

Add the following code to the **createExpense()** method

```
// Generate UUID for the expense
var expenseId = UUID.randomUUID();

// Prepare the request body to send to the database service
var databaseServiceUrl = "http://localhost:8090/";// URL of the
Database Service Application
var expense = new Expense();
expense.setId(expenseId);
expense.setAmount(request.getAmount());
expense.setDescription(request.getDescription());
expense.setSplitType(request.getSplitType());
```

```

        expense.setDate(LocalDate.now());

        var updatedExpenses =
restTemplate.postForObject(databaseServiceUrl + "expenses", expense,
Expense.class);

        for (var friend : request.getFriends()) {
            CreateExpenseAllocationRequest allocationRequest = new
CreateExpenseAllocationRequest();
            allocationRequest.setExpenseId(expenseId);
            allocationRequest.setFriendId(friend.getUsername());

allocationRequest.setInitatorUser(request.getInitatorUser());
            allocationRequest.setAmount(friend.getAmount());

if(request.getSplitType().equals("EXACT_AMOUNT")) {
    allocationRequest.setAmount(friend.getAmount());
}
else {
    allocationRequest.setAmount(request.getAmount()
        .divide(BigDecimal.valueOf(request.getFriends().size()),
RoundingMode.HALF_UP));
}

            restTemplate.postForObject(databaseServiceUrl +
"expenseAllocation", allocationRequest,
CreateExpenseAllocationRequest.class);
        }

```

### Step 3: Review Friends Service

Review the FriendsService class in ExpenseManager (could be used later as you build on to the project)

### Step 4: Review User Service

Review the UserService class in ExpenseManager (could be used later as you build on to the project)

## PART 7: Review Controller Classes (ExpenseManager)

### Step 1: Review Controller Classes

Review content of all controller classes

## PART 8: Kafka Set Up

### Step 1: Download kafka

Download kafka: [Apache Kafka Downloads page](#)

Download the latest binary release (choose **Scala 2.13** if unsure)

Extract zip file

Open a terminal and cd into the folder (e.g cd kafka\_2.13-3.3.2)

## Step 2: Start Zookeeper

Run the following command

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

## Step 3: Start Kafka Broker

Open new terminal tab and run

```
bin/kafka-server-start.sh config/server.properties
```

## Step 4: Create Topic

Open new terminal tab and run

```
bin/kafka-topics.sh --create --topic expenses-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
```

## Step 5: List Kafka Topic (Optional)

Run command to list kafka topic

```
bin/kafka-topics.sh --list --bootstrap-server localhost:9092
```

## Step 6: Open Producer (Optional)

Run command to list kafka topic

```
bin/kafka-console-producer.sh --topic expenses-topic --bootstrap-server localhost:9092
```

## Step 7: Open Consumer(Optional)

Run command to list kafka topic

```
bin/kafka-console-consumer.sh --topic expenses-topic --from-beginning --bootstrap-server localhost:9092
```

# PART 9: Update ExpenseManager to Publish to Kafka

## Step 1: Update createExpense method

Add the following code to the **createExpense()** method at the end

```

if(updatedExpenses != null) {
    var message = "Expense added: " + updatedExpenses.getId() +
    " - " + updatedExpenses.getDescription();
    // Publish the event to Kafka
    kafkaProducerService.sendMessage("expenses-topic", message);
}

```

## PART 10: Update NotificationService to Consume from Kafka

Step 1: Add code below to consumeBillSplitEvent()

```

@KafkaListener(topics = "expenses-topic", groupId =
"my-group-id")
public void consumeBillSplitEvent(String message) {
    // Process the incoming message
    String notificationMessage =
createNotificationMessage(message);

    // Send notification to the frontend
    notifyFrontend(notificationMessage);
}

```

Step 2: Add code below to createNotificationMessage()

```

// Helper method to create a notification message based on the Kafka
message
private String createNotificationMessage(String message) {
    return "Your friend has been notified to pay the bill. Event
details: " + message;
}

```

Step 3: Add code below to notifyFriends()

```

// Notify the frontend via WebSocket
private void notifyFrontend(String notificationMessage) {
    messagingTemplate.convertAndSend("/topic/notifications",
notificationMessage);
}

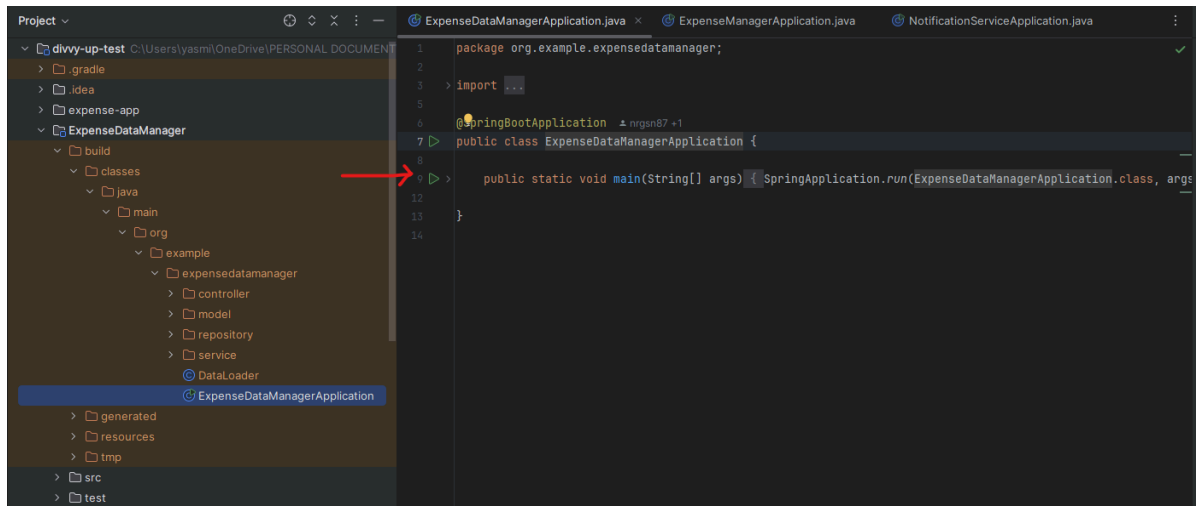
```

## PART 11: Run Backend

Step 1: Run all three Backend Services

Search for the “expenseDataManagerApplication.java” file (Shift, Shift)

Run `expenseDataManagerApplication.java` by clicking on the second play button.



Search for the “`expenseManagerApplication.java`” file (Shift, Shift)

Run `expenseManagerApplication.java` by clicking on the second play button.

Search for the “`NotificationServiceApplication.java`” file (Shift, Shift)

Run `NotificationServiceApplication.java` by clicking on the second play button.

## Step 2: Review Data Loader Class

Navigate to `DataLoader.java` in `ExpenseDataManager` and review

## Step 3: Navigate to H2 (database) Url

Navigate to h2 url to view the data loaded in the db and all other consequent data we will include

<http://localhost:8090/h2-console/login.jsp?jsessionid=49e016d4ce88b05e6744f46e70d65b34>

Change the JDBC url to: **`jdbc:h2:mem:testdb;NON_KEYWORDS=user`**

**Password: sa**

## PART 12: Review UI Code (expense-app)

### Step 1: Starting the UI

Navigate to `expense-app` folder and review the code

From a terminal, run the following command to start the UI.

```
cd expense-app
npm install
npm run build
npm start
```

## PART 13: Authentication (Auth0)

### Step 1: Import Auth0 Library

Navigate to App.js and add the code below to import the Auth0 library and the Login page

```
import LoginButton from './LoginButton';  
import { useAuth0 } from '@auth0/auth0-react';
```

### Step 2: Authenticate user

comment the whole function “function App() and add the code below to verify if the user is authenticated if not, it will route the user to the login page

```
function App() {  
  const { user, isAuthenticated, isLoading } = useAuth0();  
  
  return (  
    isAuthenticated ? (  
      <div>  
        <ExpensePage/>  
        <NotificationComponent />  
      </div>  
    ): <LoginButton/>  
  );  
}  
export default App;
```

### Step 4: Import Logout Button

Navigate to ExpensePage.js and add the following code to import the logout page

```
import LogoutButton from './LogoutButton';
```

### Step 5: Add Logout Button

On the same page ExpensePage.js add the following after line 168 (<button type="submit" className="submit-btn">Add Expense</button>)

```
<LogoutButton/>
```

### Step 6: Import Auth0 library in index.js

Navigate to index.js import the Auth0 library by adding the following code

```
import {Auth0Provider} from '@auth0/auth0-react';
```

### Step 7: Add Auth0 Domain and Client Domain for API connection

on the same file, index.js, comment line 7-13 and add the following code: the following code

```
const domain = process.env.REACT_APP_AUTH0_DOMAIN;  
const clientId = process.env.REACT_APP_AUTH0_CLIENT_ID;  
const root = ReactDOM.createRoot(document.getElementById('root'));  
  
root.render(  
  <React.StrictMode>  
    <Auth0Provider  
      domain={domain}  
      clientId={clientId}  
      redirectUri={window.location.origin}  
    >  
      <Router>  
        <FriendsProvider>  
          <App />  
        </FriendsProvider>  
      </Router>  
    </Auth0Provider>  
  </React.StrictMode>  
>);
```

### Step 8: Rerun the application

```
npm install  
npm run build  
npm start
```

### Step 9: Review LoginButton and LogoutButton files

Open LoginButton.js and LogoutButton.js and review

## Strength Goals



1. Add a service to process payment and publish payment processing event to kaAdd details on how to move to the next steps and connect the authenticated user to the database