NAME : THIRSHA SREE H

REG NO **: 20BCE2518**

Step 1: Set up your Meteor project

Create a new Meteor project:

bash

meteor create loan-management-app

cd loan-management-app

Step 2: Install required packages

Install necessary packages, including alanning:roles for role-based access control and useraccounts:core for user authentication:

bash

meteor add alanning:roles

meteor add useraccounts:core

Step 3: Define collections and roles

Create collections for Users and Loans in a file named collections.js:

javascript

// collections.js

Users = new Mongo.Collection('users');

Loans = new Mongo.Collection('loans');

Define roles for users in another file named roles.js:

javascript

// roles.js

Roles.createRole('admin');

Roles.createRole('borrower');

Roles.createRole('lender');

Step 4: Implement server-side logic

In the main.js file:

// main.js

Meteor.publish('users', function() {

```
  return Users.find();
});

Meteor.publish('loans', function() {
  return Loans.find();
});

Meteor.methods({
  'createUser': function(email, role) {
    const userId = Users.insert({ email, role });
    if (role === 'admin') {
      Roles.addUsersToRoles(userId, 'admin');
    } else if (role === 'borrower') {
      Roles.addUsersToRoles(userId, 'borrower');
    } else if (role === 'lender') {
      Roles.addUsersToRoles(userId, 'lender');
    }
  },
  'requestLoan': function(amount) {
    Loans.insert({ amount, status: 'pending', userId: Meteor.userId() });
  },
  'confirmPayment': function(loanId) {
    Loans.update(loanId, { $set: { status: 'paid' } });
  }
});
```

Step 5: Implement client-side logic

```
// main.js
Meteor.subscribe('users');
```

```
Meteor.subscribe('loans');
Template.registerForm.events({
  'submit form': function(event) {
    event.preventDefault();
    const email = event.target.email.value;
    const role = event.target.role.value;
    Meteor.call('createUser', email, role);
  }
});

Template.loanRequestForm.events({
  'submit form': function(event) {
    event.preventDefault();
    const amount = event.target.amount.value;
    Meteor.call('requestLoan', amount);
  }
});

Template.confirmPaymentForm.events({
  'submit form': function(event) {
    event.preventDefault();
    const loanId = event.target.loanId.value;
    Meteor.call('confirmPayment', loanId);
  }
});

Template.pastLoans.helpers({
  loans: function() {
    return Loans.find({ userId: Meteor.userId() });
```

```
  }
});

Template.pastPayments.helpers({
  payments: function() {
    return Loans.find({ status: 'paid', userId: Meteor.userId() });
  }
});
```

Step 6: Create HTML Templates

In the loan-management-app.html file:

```
html
<!-- loan-management-app.html -->
<head>
  <title>Loan Management App</title>
</head>

<body>
  {{> registerForm}}
  {{> loanRequestForm}}
  {{> confirmPaymentForm}}
  {{> pastLoans}}
  {{> pastPayments}}
</body>

<template name="registerForm">
  <h2>Register</h2>
  <form>
    <label>Email:</label>
```

```
    <input type="email" name="email" required>

    <label>Role:</label>
    <select name="role">
      <option value="admin">Admin</option>
      <option value="borrower">Borrower</option>
      <option value="lender">Lender</option>
    </select>

    <button type="submit">Register</button>
  </form>
</template>

<template name="loanRequestForm">
  <h2>Request Loan</h2>
  <form>
    <label>Amount:</label>
    <input type="number" name="amount" required>

    <button type="submit">Request Loan</button>
  </form>
</template>

<template name="confirmPaymentForm">
  <h2>Confirm Payment</h2>
  <form>
    <label>Loan ID:</label>
    <input type="text" name="loanId" required>
```

```
    <button type="submit">Confirm Payment</button>
  </form>
</template>

<template name="pastLoans">
  <h2>Past Loans</h2>
  {{#each loans}}
    <p>Loan ID: {{_id}}, Amount: {{amount}}, Status: {{status}}</p>
  {{/each}}
</template>

<template name="pastPayments">
  <h2>Past Payments</h2>
  {{#each payments}}
    <p>Loan ID: {{_id}}, Amount: {{amount}}, Status: {{status}}</p>
  {{/each}}
</template>
```

Step 7: Add User Authentication

Integrate the useraccounts:core package for user authentication. Follow the package documentation to set it up according to your needs.

Step 8: Secure Your Methods

Update your server-side methods in server/main.js to include checks for user roles and proper validation:

javascript

```
Meteor.methods({
  'createUser': function(email, role) {
    if (!Roles.userIsInRole(this.userId, 'admin')) {
      throw new Meteor.Error('not-authorized', 'Only admins can create users.');
    }
```

```
    // Rest of the code remains the same
  },
  'requestLoan': function(amount) {
    if (!Roles.userIsInRole(this.userId, 'borrower')) {
      throw new Meteor.Error('not-authorized', 'Only borrowers can request loans.');
    }
    // Rest of the code remains the same
  },
  'confirmPayment': function(loanId) {
    if (!Roles.userIsInRole(this.userId, 'lender')) {
      throw new Meteor.Error('not-authorized', 'Only lenders can confirm payments.');
    }
    // Rest of the code remains the same
  }
})
```

Step 9: Deploy the Application