```java

package acme.features.client.contract;

import java.util.Collection;
import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import acme.client.data.datatypes.Money;
import acme.client.data.models.Dataset;
import acme.client.helpers.MomentHelper;
import acme.client.services.AbstractService;
import acme.client.views.SelectChoices;
import acme.entities.contract.Contract;
import acme.entities.project.Project;
import acme.roles.Client;

@Service
public class ClientContractCreateService extends AbstractService<Client, Contract> {

    @Autowired
    private ClientContractRepository repository;


    @Override
    public void authorise() {
        super.getResponse().setAuthorised(super.getRequest().getPrincipal().hasRole(Client.class));
    }

    @Override
    public void load() {
        Contract object;
        Client client;

        client = this.repository.findClientByClientId(super.getRequest().getPrincipal().getActiveRoleId());
        object = new Contract();
        object.setDraft(true);
        object.setClient(client);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final Contract contract) {
        assert contract != null;
        int projectId;
```

```
48          Project project;
49          projectId = super.getRequest().getData("project", int.class);
50          project = this.repository.findOneProjectById(projectId);
51          Date currentMoment = MomentHelper.getCurrentMoment();
52          Date moment = new Date(currentMoment.getTime());
53          super.bind(contract, "code", "moment", "providerName",
    "customerName", "goals", "budget", "isDraft", "project");
54          contract.setProject(project);
55          contract.setMoment(moment);
56      }
57
58      @Override
59      public void validate(final Contract contract) {
60          assert contract != null;
61          Collection<Contract> contracts = null;
62          if (!super.getBuffer().getErrors().hasErrors("code")) {
63              Contract existing;
64              existing = this.repository.findOneContractByCode
    (contract.getCode());
65              super.state(existing == null || existing.equals(contract),
    "code", "client.contract.form.error.duplicated");
66          }
67          if (!super.getBuffer().getErrors().hasErrors("budget"))
68              super.state(contract.getBudget().getAmount() >= 0, "budget",
    "client.contract.form.error.negative-budget");
69          if (!super.getBuffer().getErrors().hasErrors("budget")) {
70              contracts = this.repository.findAllContractsOfAProjectById
    (contract.getProject().getId());
71              double totalBudget = contracts.stream().filter(p -> p.getId() !
    = contract.getId()).mapToDouble(c -> this.eurConverter(c.getBudget())).sum
    ();
72              double projectCost = contract.getProject().getCost().getAmount
    ();
73              totalBudget += this.eurConverter(contract.getBudget());
74              super.state(totalBudget <= projectCost, "budget",
    "client.contract.form.error.exceeds-project-cost");
75          }
76      }
77
78      private double eurConverter(final Money money) {
79          String currency = money.getCurrency();
80          double amount = money.getAmount();
81
82          if (currency.equals("EUR"))
83              amount = amount;
84          else if (currency.equals("USD"))
85              amount = amount * 0.90; // Tasa aproximada de conversión USD a
    EUR
86          else if (currency.equals("GBP"))
87              amount = amount * 1.17; // Tasa aproximada de conversión GBP a
```

```
    EUR
 88         else
 89             super.state(false, "budget",
    "client.contract.unsopportedCurrency");
 90         return amount;
 91 
 92     }
 93 
 94     @Override
 95     public void perform(final Contract contract) {
 96         assert contract != null;
 97 
 98         this.repository.save(contract);
 99     }
100 
101     @Override
102     public void unbind(final Contract contract) {
103         assert contract != null;
104         boolean isDraft;
105         Collection<Project> projects;
106         projects = this.repository.findAllPublishedProjects();
107         SelectChoices choices;
108         choices = SelectChoices.from(projects, "title", contract.getProject
    ());
109         isDraft = contract.isDraft() == true;
110 
111         Dataset dataset;
112 
113         dataset = super.unbind(contract, "code", "moment", "providerName",
    "customerName", "goals", "budget", "isDraft", "project");
114         dataset.put("contractId", contract.getId());
115         dataset.put("isDraft", isDraft);
116         dataset.put("project", choices.getSelected().getKey());
117         dataset.put("projects", choices);
118 
119         super.getResponse().addData(dataset);
120     }
121 
122 }
123 
```