Table of Contents

Sommaire

Business Object Container Tutorial

- clone https://github.com/ccaavvaa/boc-formation.git
- switch local git: tag 'lesson01' et créer une branche
- install une base MDR dans http://localhost/MDR/Basic
- installer les paquets:

```
yarn install
```

- lancer le script de création des tables dans MDR:

```
cd schema\Models\Basic
create_schema.bat
```

- compiler et lancer les tests

```
gulp test
```

- lancer couverture des tests

```
npm run coverage
```

Premier modèle

- Ajouter birthDate dans le schéma Person.json

```
{
    "$schema": "http://phenix.salviadeveloppement.com/json-schema/mdr-schema.json",
    "version": "1.0.0",
    "type": "object",
    "title": "Person",
    "name": "Person",
    "primaryKey": "personId",
    "properties": {
        "personId": {
            "type": "string",
            "format": "code"
        },
        "name": {
            "type": "string"
        },
        "firstName": {
            "type": "string"
        },
        "birthDate": {
            "type": "string",
            "format": "date"
        }
    }
}
```

- Générer les classes modèle. La classe Person sera créé dans src/lib/models/Person.ts

```
npm run entity
```

- Ajouter la classe Person dans la liste des classes métier dans src/lib/business-classes.ts

```
export const businessClasses = [
    ...
    // Person
    Person,
];
```

- Ajouter des tests Dans le fichier src/test/person/person.test.ts

```
describe('Person', function () {
    it('personId should not be empty', async function () {
        const c = createContainer();
        const person = await c.createNew<Person>(Person);
        await person.validate();
        assert(person.hasErrors);
        let personIdErrors: boc.IErrorInfo[];
        personIdErrors = person.errors.errors.get('personId');
        assert(personIdErrors && personIdErrors.length);
        const err = personIdErrors.find(
            (e) => e.error.srcId === 'PersonRules.notEmptyPersonId'
        );
```

```
        assert(err);
        assert(err.error.message === c.t("L'id de la personne est vide"));

        await person.set_personId('not empty');
        await person.validate();
        personIdErrors = person.errors.errors.get('personId');
        assert(!personIdErrors || !personIdErrors.find(
            (e) => e.error.srcId === 'PersonRules.notEmptyPersonId')
        );
    });
    it('personId should not be empty on personId prop change', async function () {
        const c = createContainer();
        const person = await c.createNew<Person>(Person);
        await person.set_personId('not empty');
        await person.set_personId(null);
        assert(person.hasErrors);
        let personIdErrors: boc.IErrorInfo[];
        personIdErrors = person.errors.errors.get('personId');
        assert(personIdErrors && personIdErrors.length);
        const err = personIdErrors.find(
            (e) => e.error.srcId === 'PersonRules.notEmptyPersonIdOnChange'
        );
        assert(err);
        assert(err.error.message === c.t("L'id de la personne est vide"));

        await person.set_personId('not empty');
        personIdErrors = person.errors.errors.get('personId');
        assert(!personIdErrors || !personIdErrors.find(
            (e) => e.error.srcId === 'PersonRules.notEmptyPersonId')
        );
    });
});
```

- Ajouter les règles dans src/lib/rules/person/PersonRules.ts

```
export class PersonRules {
    @boc.PropChange({
        constr: Person,
        propName: 'personId',
        description: 'Empty personId should immediate show error',
    })
    public static async notEmptyPersonIdOnChange(target: Person, msg: boc.Message) {
        this.checkPersonId(target);
    }

    @boc.Validate({
        constr: Person,
        description: 'personId should not be empty',
    })
    public static async notEmptyPersonId(target: Person, msg: boc.Message) {
        this.checkPersonId(target);
    }

    private static checkPersonId(person: Person) {
        const c = person.container;
        if (!person.personId) {
            person.errors.addError(c.t("L'id de la personne est vide"), 'personId');
        }
    }
}
```

Exercices

Ajouter les règles: nom, prénom, date de naissance sont obligatoire

Premier view model

- Ajouter la classe PersonView dans le fichier src/views/person/PersonView.ts
  - La classe est un view model pour la classe Person: boc.ViewModel, modelConstr: Person
  - Les propriétés de la classe Person sont mapés: mappingDef from #model
  - Une propriété âge est calculée

```
@boc.ClsInfo({
description: 'Person View',
isTransient: true,
modelConstr: Person,
mappingDef: [
  {
      from: '#model',
      mappings: [
          'personId',
          'name',
          'firstName',
          'birthDate',
          {
              from: 'id',
              to: 'personKey',
          },
      ],
  },
],
})
export class PersonView extends boc.ViewModel<Person> {
@boc.PropertyInfo({
  type: 'integer',
  jsFormats: ['integer'],
})
public get age(): number {
  return this.getProp('age');
}

public set_age(value: number) {
  return this.setProp('age', value);
}
}
```

- Ajouter les tests dans src/test/person/person-view.test.ts
  - le view est initialisé avec divers références vers l'objet person: id, $reference ou personId
  - l'âge est calculé quand le model change ou la date de naissance change

```
describe('PersonView', function () {
    it('create with model ', async function () {
        const c = createContainer();
        const person = await c.createNew<Person>(Person);
        const view = await person.createViewModel<PersonView>(PersonView);
        assert(view && view.model === person);
    });
    it('create without model', async function () {
        const c = createContainer();
        const view = await c.createNew<PersonView>(PersonView);
        assert(view && !view.model);
```

```
});
it('create without model 2', async function () {
    const c = createContainer();
    const view = await c.createNew<PersonView>(PersonView, null, { notUsed: true });
    assert(view && !view.model);
});
it('create with model index', async function () {
    const c = createContainer();
    const person = await c.createNew<Person>(Person);
    const view =
        await c.createNew<PersonView>(PersonView, null, { id: person.id });
    assert(view && (view.model === person));
});
it('create with model ref', async function () {
    const c = createContainer();
    const person = await c.createNew<Person>(Person);
    const view =
        await c.createNew<PersonView>(PersonView, null, { personRef: person.$reference });
    assert(view && (view.model === person));
});
it('create with personId', async function () {
    const c = createContainer();
    const person = await c.createNew<Person>(Person);
    await person.set_personId('p1');
    const view =
        await c.createNew<PersonView>(PersonView, null, { personId: person.personId });
    assert(view && (view.model === person));
});

it('calculate age', async function () {
    const c = createContainer();
    const person = await c.createNew<Person>(Person);
    const personAge = 5;
    const todayDate = new Date();
    const birthDate = new boc.DateTime('date', todayDate).addYears(-personAge);
    await person.set_birthDate(birthDate);
    const view = await person.createViewModel<PersonView>(PersonView);
    assert(view.age === personAge);
    await person.set_birthDate(person.birthDate.addYears(1));
    assert(view.age === personAge - 1);
});
it('Should serialize person properties', async function () {
    const c = createContainer();
    const person = await c.createNew<Person>(Person);
    const personAge = 5;
    const todayDate = new Date();
    const birthDate = new boc.DateTime('date', todayDate).addYears(-personAge);
    await person.set_birthDate(birthDate);
    await person.set_name('N');
    await person.set_firstName('F');
    await person.set_personId('P1');
    const view = await person.createViewModel<PersonView>(PersonView);
    const data = await view.getAllData(null, true);
    assert(data.id);
    assert(data.personId === person.personId);
    assert(data.name === person.name);
    assert(data.firstName === person.firstName);
    assert(data.birthDate === person.birthDate.toString());
    assert(data.age === personAge);
    assert(data.personKey === person.id);
});
```

```
});
```

- Ajouter la class PersonViewRules dans src/rules/person/PersonViewRules.ts

```
export class PersonViewRules {
    @boc.ObjectInit({
        constr: PersonView,
        isNew: true,
    })
    public static async init(target: PersonView, msg: boc.Message) {
        const c = target.container;
        if (!target.model && msg.data) {
            let model: Person;
            if (msg.data.personId) {
                model = await c.getOne<Person>(Person, { personId: msg.data.personId });
            } else if (msg.data.id) {
                model = c.getInMemByIndex<Person>(Person, msg.data.id);
            } else if (msg.data.personRef) {
                model = c.getInMemByRef(msg.data.personRef);
            }
            if (model) {
                await target.set_model(model);
            }
        }
    }
    @boc.ModelChanged({
        constr: PersonView,
    })
    public static async modelChanged(target: PersonView, msg: boc.Message) {
        const value = PersonRules.calculateAge(target.model);
        await target.set_age(value);
    }
    @boc.PropChange({
        constr: PersonView,
        propName: 'birthDate',
        path: '#model',
    })
    public static async calculateAge(personView: PersonView) {
        const value = PersonRules.calculateAge(personView.model);
        await personView.set_age(value);
    }
}
```

- Ajouter la fonction getAge dans src/rules/hepers.ts

```
public static getAge(birthDate: boc.DateTime, atDate?: boc.DateTime): number;
```

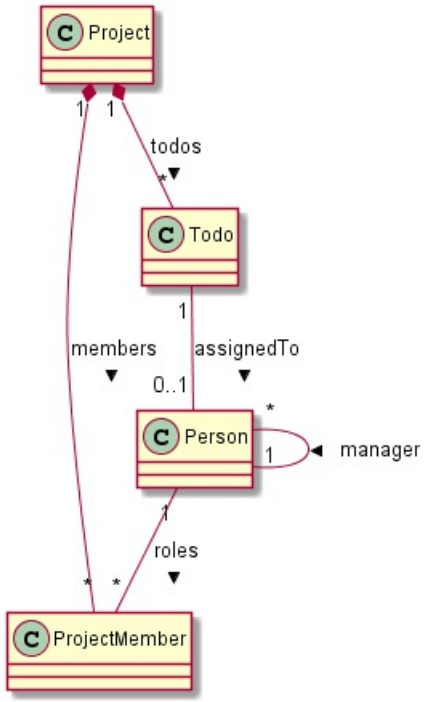
- Ajouter la fonction calculateAge dans la classe PersonRules

```
public static calculateAge(person: Person): number;
```

- Ajouter les tests pour getAge et calculateAge
- Ajouter les classes dans businessClasses et businessRules
- Exercices

- Ajouter une proprieté fullName
  - dans le model
  - dans le view model

Rélations



Types de rélations:

```
* compositions
    * HasOne
    * HasMany
* simples
    * Reference
    * Many
```

- Ajouter le définitions classes et les rélations dans schéma

  Project.json:

```
{
    "$schema": "http://phenix.salviadeveloppement.com/json-schema/mdr-schema.json",
    "version": "1.0.0",
    "type": "object",
    "title": "Project",
    "name": "Project",
    "primaryKey": "code",
    "properties": {
        "code": {
            "type": "string",
            "format": "code"
        },
        "name": {
            "type": "string"
        },
        "todos": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/TodoRef"
```

```
            }
        },
        "members": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/ProjectMemberRef"
            }
        }
    },
    "definitions": {
        "ProjectMemberRef": {
            "type": "object",
            "name": "ProjectMember",
            "title": "Project Member",
            "properties": {
                "role": {
                    "type": "string",
                    "format": "code"
                },
                "personId": {
                    "type": "string",
                    "format": "code"
                }
            },
            "relations": {
                "person": {
                    "foreignEntity": "Person",
                    "foreignKey": [
                        "personId"
                    ],
                    "key": [
                        "personId"
                    ],
                    "multiplicity": "one",
                    "inverseMultiplicity": "many"
                }
            }
        },
        "TodoRef": {
            "type": "object",
            "name": "Todo",
            "title": "Todo item",
            "properties": {
                "title": {
                    "type": "string"
                },
                "progress": {
                    "type": "integer"
                },
                "dueDate": {
                    "type": "string",
                    "format": "date"
                },
                "personId": {
                    "type": "string",
                    "format": "code"
                }
            },
            "relations": {
                "person": {
                    "foreignEntity": "Person",
```

```
                    "foreignKey": [
                        "personId"
                    ],
                    "key": [
                        "personId"
                    ],
                    "multiplicity": "one",
                    "inverseMultiplicity": "many"
                }
            }
        }
    }
}
```

Person.json

```
{
    "$schema": "http://phenix.salviadeveloppement.com/json-schema/mdr-schema.json",
    "version": "1.0.0",
    "type": "object",
    "title": "Person",
    "name": "Person",
    "primaryKey": "personId",
    "properties": {
        "personId": {
            "type": "string",
            "format": "code"
        },
        "name": {
            "type": "string"
        },
        "firstName": {
            "type": "string"
        },
        "birthDate": {
            "type":"string",
            "format":"date"
        },
        "refManager": {
            "type": "integer"
        }
    },
    "relations": {
        "manager": {
            "foreignEntity": "Person",
            "foreignKey": [
                "id"
            ],
            "key": [
                "refManager"
            ],
            "multiplicity": "one",
            "inverseMultiplicity": "many"
        },
        "teamMembers": {
            "foreignEntity": "Person",
            "foreignKey": [
                "refManager"
            ],
```

```
            "key": "id",
            "multiplicity": "one",
            "inverseMultiplicity": "many"
        }
    }
}
```

- Générer les classes model

```
npm run entity
```

- Ajouter les tests src/test/project/project.test.ts

```
describe('Project', function () {
    it('code, name should not be empty', async function () {
        const c = createContainer();
        const project = await c.createNew<Project>(Project);
        await project.validate();
        assert(project.hasErrors);
        let propertyErrors: boc.IErrorInfo[];
        for (const p of projectNotEmptyProperties) {
            propertyErrors = project.errors.errors.get(p);
            assert(propertyErrors && propertyErrors.length);
            const err = propertyErrors.find(
                (e) => e.error.srcId === 'ProjectRules.notEmptyProperties'
            );
            assert(err);
            assert(err.error.message === c.t('Valeur obligatoire'));
            const value = 'not empty';
            await project.setProp(p, value);
            await project.validate();
            propertyErrors = project.errors.errors.get(p);
            assert(!propertyErrors || !propertyErrors.find(
                (e) => e.error.srcId === 'ProjectRules.notEmptyProperties')
            );
        }
    });

    it('members only once', async function () {
        const c = createContainer();
        const project = await c.createNew<Project>(Project);
        const persons: Person[] = [];
        for (let i = 0; i < 2; i++) {
            const person = await c.createNew<Person>(Person);
            const n = i.toString();
            await person.set_personId(n);
            await person.set_name(n);
            await person.set_firstName(n);

            const projectMember = await c.createNew<ProjectMember>(ProjectMember);
            await projectMember.set_person(person);
            await projectMember.set_role('dev');
            await project.members.link(projectMember);
            persons.push(person);
        }

        project.members.link(await c.createNew<ProjectMember>(ProjectMember));
        const m = await c.createNew<ProjectMember>(ProjectMember);
        await m.set_person(persons[0]);
```

```
        await m.set_role('dev');
        await project.members.link(m);
        await project.validate();
        assert(project.hasErrors);
        let membersErrors = project.errors.errors.get('members');
        assert(membersErrors && membersErrors.length);
        const err = membersErrors.find(
            (e) => e.error.srcId === 'ProjectRules.membersOnlyOnce'
        );
        assert(
            err &&
            err.error.message ===
            c.t('Les membres suivants sont déclares plusieurs fois dans le projet.\n')
            .concat('0 0')
        );

        await project.members.unlink(m);
        await project.validate();
        membersErrors = project.errors.errors.get('members');
        assert(
            !membersErrors
            || !membersErrors.length
            || !membersErrors.find(
                (e) => e.error.srcId === 'ProjectRules.membersOnlyOnce'
            )
        );
    });
    it('calculate project name', async function () {
        const c = createContainer();
        const project = await c.createNew<Project>(Project);
        let v: string = ProjectRules.getProjectName(project);
        assert(v);
        await project.set_code('c');
        v = ProjectRules.getProjectName(project);
        assert(v === project.code);
        await project.set_name('n');
        v = ProjectRules.getProjectName(project);
        assert(v === project.name);
    });
});
```

dans src/test/project/project-member.test.ts

```
describe('Project Member', function () {
    it('role should not be empty', async function () {
        const c = createContainer();
        const projectMember = await c.createNew<ProjectMember>(ProjectMember);
        await projectMember.validate();
        assert(projectMember.hasErrors);
        let propertyErrors: boc.IErrorInfo[];
        for (const p of projectMemberNotEmptyProperties) {
            propertyErrors = projectMember.errors.errors.get(p);
            assert(propertyErrors && propertyErrors.length);
            const err = propertyErrors.find(
                (e) => e.error.srcId === 'ProjectMemberRules.notEmptyProperties'
            );
            assert(err);
            assert(err.error.message === c.t('Valeur obligatoire'));
            const value = 'not empty';
```

```
            await projectMember.setProp(p, value);
            await projectMember.validate();
            propertyErrors = projectMember.errors.errors.get(p);
            assert(!propertyErrors || !propertyErrors.find(
                (e) => e.error.srcId === 'ProjectMemberRules.notEmptyProperties')
            );
        }
    });
    it('person should not be empty', async function () {
        const c = createContainer();
        const projectMember = await c.createNew<ProjectMember>(ProjectMember);
        await projectMember.validate();
        assert(projectMember.hasErrors);
        let propertyErrors: boc.IErrorInfo[];
        propertyErrors = projectMember.errors.errors.get('person');
        assert(propertyErrors && propertyErrors.length);
        const err = propertyErrors.find(
            (e) => e.error.srcId === 'ProjectMemberRules.checkPersonReference'
        );
        assert(err);
        assert(err.error.message === c.t('Valeur obligatoire'));
        const value = await c.createNew<Person>(Person);
        await projectMember.set_person(value);
        await projectMember.validate();
        propertyErrors = projectMember.errors.errors.get('person');
        assert(!propertyErrors || !propertyErrors.find(
            (e) => e.error.srcId === 'ProjectMemberRules.checkPersonReference')
        );
    });
    it('should block deleting referenced person', async function () {
        const c = createContainer();
        const project = await c.createNew<Project>(Project);
        await project.set_name('P1');
        const projectMember = await c.createNew<ProjectMember>(ProjectMember);
        await project.members.link(projectMember);

        const person = await c.createNew<Person>(Person);
        await person.set_name('J');
        await projectMember.set_person(person);
        await person.toDelete();
        assert(!person.isDeleted);
        const personErrors = person.errors.errors.get('$');
        assert(personErrors);
        const err = personErrors.find(
            (e) => e.error.srcId === 'ProjectMemberRules.doNotDeleteReferencedPerson'
        );
        assert(err);
        assert(err.error.isTransient);
        const expectedMessage =
            c.t('0 participe dans les projets suivants:\n',
                PersonRules.calculateFullName(person))
                .concat('P1');
        assert(err.error.message === expectedMessage);
    });
});
```

- Ajouter les règles métiers dans src/lib/rules/project/ProjectRules.ts

```
export const projectNotEmptyProperties: Array<keyof Project & string> = [
    'code', 'name',
```

```
];

export class ProjectRules {

    @boc.Validate({
        constr: Project,
    })
    public static async notEmptyProperties(target: Project, msg: boc.Message) {
        for (const propName of projectNotEmptyProperties) {
            Helpers.checkEmptyProp(target, propName);
        }
    }

    @boc.Validate({
        constr: Project,
    })
    public static async membersOnlyOnce(target: Project, msg: boc.Message) {
        const c = target.container;
        const members = await target.members.toArray();
        const nullPersonId = 'nullPersonId';
        const counts = _.countBy(members, (m: ProjectMember) =>
            m.personId ? m.personId : nullPersonId);
        const personIds = Object.getOwnPropertyNames(counts)
            .filter((k) => k !== nullPersonId && counts[k] > 1);
        if (personIds.length) {
            const personNames: string[] = [];
            for (const personId of personIds) {
                const person = await c.getOne<Person>(Person, { personId });
                personNames.push(PersonRules.calculateFullName(person));
            }
            const error: string =
                c.t('Les membres suivants sont déclares plusieurs fois dans le projet.\n')
                .concat(
                    personNames.join('\n')
                );
            target.errors.addError(error, 'members');
        }
    }

    public static getProjectName(p: Project): string {
        return p.name || p.code || p.id.toString();
    }
}
```

dans src/lib/rules/project/ProjectMemberRules

```
export const projectMemberNotEmptyProperties: Array<keyof ProjectMember & string> = [
    'role',
];
export class ProjectMemberRules {

    @boc.Validate({
        constr: ProjectMember
    })
    public static notEmptyProperties(target: ProjectMember, msg: boc.Message) {
        for (const propName of projectMemberNotEmptyProperties) {
            Helpers.checkEmptyProp(target, propName);
        }
    }
```

```
    @boc.Deleting({
        constr: Person
    })
    public static async doNotDeleteReferencedPerson(target: Person, msg: boc.Message) {
        const c = target.container;
        const projectMembers =
            await c.getMany<ProjectMember>(ProjectMember, { personId: target.personId });

        /* istanbul ignore else */
        if (projectMembers.length) {
            const projects: string[] = [];
            for (const m of projectMembers) {
                const project = await m.parentObject();
                projects.push(ProjectRules.getProjectName(project));
            }
            const err =
                c.t('0 participe dans les projets suivants:\n',
                    PersonRules.calculateFullName(target))
                    .concat(
                        projects.join('\n')
                    );
            throw new boc.BOErr(500, err);
        }
    }

    @boc.Validate({
        constr: ProjectMember
    })
    public static async checkPersonReference(target: ProjectMember, msg: boc.Message) {
        const c = target.container;
        const p = await target.person();
        if (!p) {
            target.errors.addError(c.t('Valeur obligatoire'), 'person');
        }
    }
}
```

- Ajouter les classes dans le serveur

Exercices

1. Ajouter les régles pour la rélation person.manager
2. Ajouter un scénario complet avec sauvegarde dans la base avec la méthode container.save()
3. Ajouter des views pour le Project, Member, Persons. Utiliser des rélations TransientHasMany, TransientMany