EAPLI

Sample Project eCafeteria

eCafeteria

Pretende-se desenvolver um sistema que compreende um conjunto de serviços relacionados com a utilização e exploração de uma cantina que funciona num estabelecimento de ensino, podendo existir várias caixas de pagamento/entrega de refeições na mesma cantina.

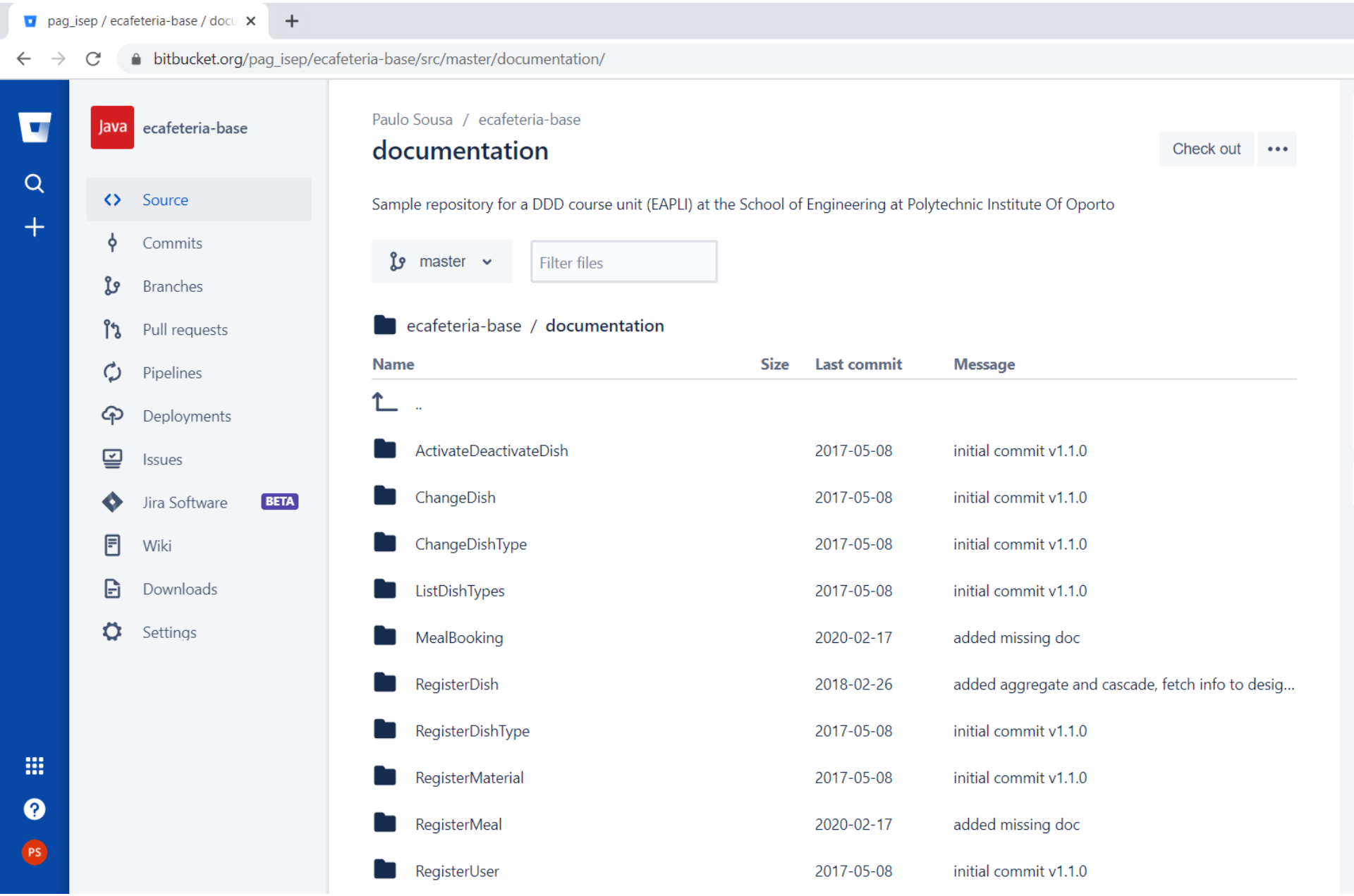
O sistema prevê diferentes tipos de utilizadores e serviços a eles dirigidos, em particular:
• **Utente**: consulta de ementas, gestão de reservas de refeições, consulta da conta corrente e do saldo;
• **Operador de Caixa**: entrega de refeições e carregamento de cartões;
• **Gestor de Ementas**: definição dos tipos de pratos, definição de pratos (receitas), e alergénios, consulta e elaboração de ementas, análise das preferências dos utentes. Cargo tipicamente desempenhado por nutricionistas;

O sistema não está integrado com sistemas de pagamentos. As refeições podem ser reservadas até à véspera desde que a conta do utente tenha saldo disponível. Os pagamentos são efectuados presencialmente nas caixas num horário específico fora dos horários de refeições. A conta pode ser creditada/carregada numa das caixas da cantina.

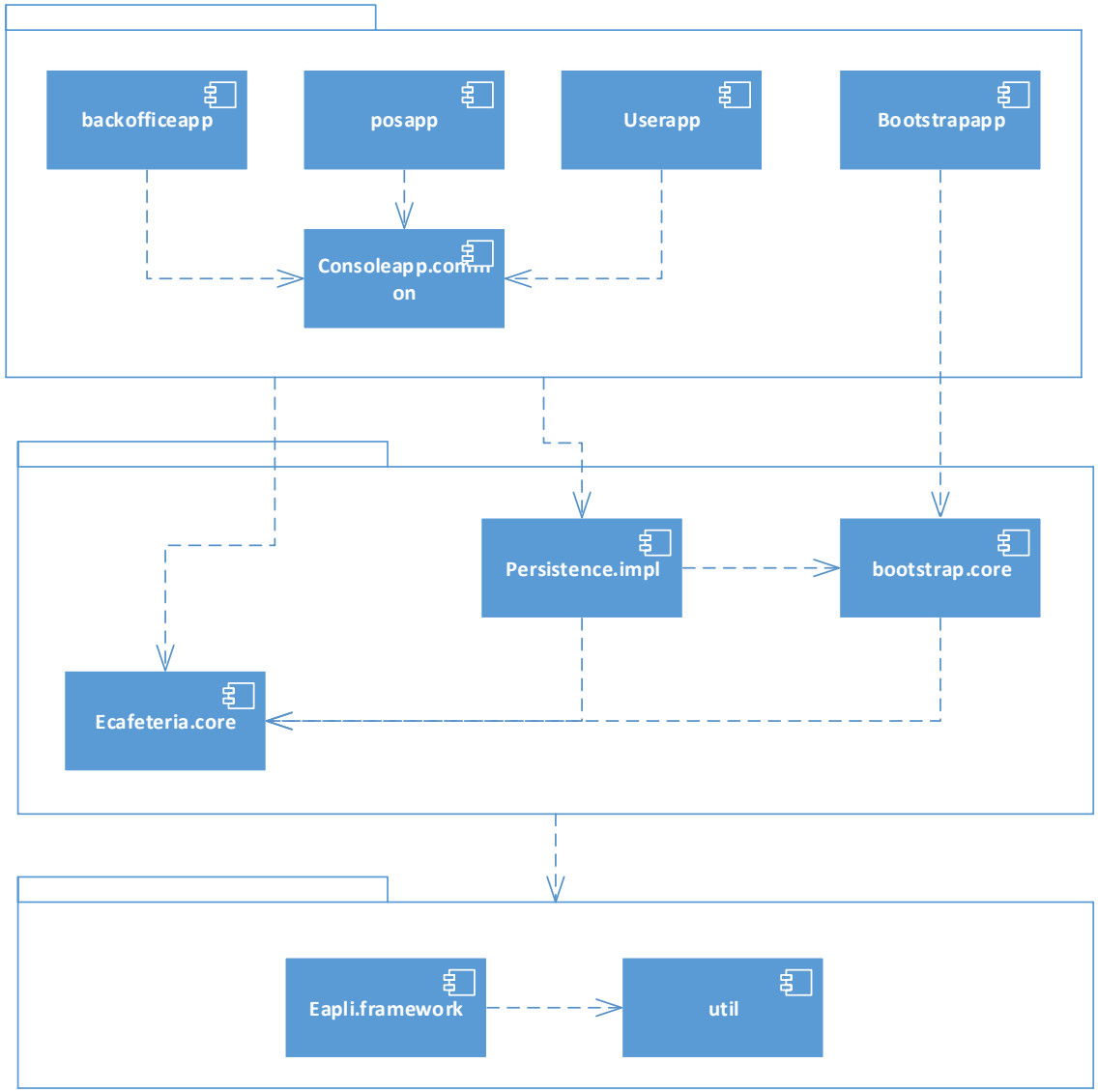
As principais áreas funcionais da aplicação são:
1. **Reservas** – os utentes da cantina têm que fazer um registo prévio na plataforma eletrónica que permite a marcação e o cancelamento de refeições. Podem fazer consultas de consumos e têm também disponível um serviço de alertas em função do saldo da conta;
2. **Caixa** – entrega de refeições;
3. **Ementas** – permite elaborar e disponibilizar as ementas semanais. A análise das vendas e reservas anteriores é fundamental para determinar quais as ementas que vão de encontro às preferências dos utentes;

eCafeteria

- Cafeteria management
- Users, Kitchen and Menu managers, Cashiers
- User app
- Backoffice app
  - Kitchen management
  - Menu management
- POS
  - Delivery station


pag_isep / ecafeteria-base / docu
bitbucket.org/pag_isep/ecafeteria-base/src/master/documentation/
Java
ecafeteria-base
Source
Commits
Branches
Pull requests
Pipelines
Deployments
Issues
Jira Software
BETA
Wiki
Downloads
Settings
PS
Paulo Sousa / ecafeteria-base
documentation
Check out
Sample repository for a DDD course unit (EAPLI) at the School of Engineering at Polytechnic Institute Of Oporto
master
Filter files
ecafeteria-base / documentation
Name
Size
Last commit
Message
..
ActivateDeactivateDish 2017-05-08 initial commit v1.1.0
ChangeDish 2017-05-08 initial commit v1.1.0
ChangeDishType 2017-05-08 initial commit v1.1.0
ListDishTypes 2017-05-08 initial commit v1.1.0
MealBooking 2020-02-17 added missing doc
RegisterDish 2018-02-26 added aggregate and cascade, fetch info to desig...
RegisterDishType 2017-05-08 initial commit v1.1.0
RegisterMaterial 2017-05-08 initial commit v1.1.0
RegisterMeal 2020-02-17 added missing doc
RegisterUser 2017-05-08 initial commit v1.1.0

Components (a.k.a. projects)



- Backofficeapp, userapp, pos
- Core, console.common
- Persistence.impl
- bootstrap

- Framework
  - Utility classes for DDD applications with JPA in EAPLI context
- Util
  - Generic utility classes

pom.xml (ecafeteria.base)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="…">
    <modelVersion>4.0.0</modelVersion>
    <groupId>eapli</groupId>
    <artifactId>ecafeteria</artifactId>
    <version>1.3.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <properties> … </properties>
    <modules> … </modules>
    <dependencies>
        <dependency>
            <groupId>eapli</groupId>
            <artifactId>eapli.framework.core</artifactId>
            <version>9.2.0</version>
        </dependency>
        <dependency>
            <groupId>eapli</groupId>
            <artifactId>eapli.framework.infrastructure.authz</artifactId>
            <version>9.2.0</version>
        </dependency>
        <dependency>
            <groupId>eapli</groupId>
            <artifactId>eapli.framework.infrastructure.pubsub</artifactId>
            <version>9.2.0</version>
        </dependency>
        <dependency> … </dependency>
        <dependency> … </dependency>
    </dependencies>
    <repositories>
        <repository>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
            <id>bintray-pagsousa-eapli</id>
            <url>http://dl.bintray.com/pagsousa/eapli</url>
        </repository>
    </repositories>
</project>
```

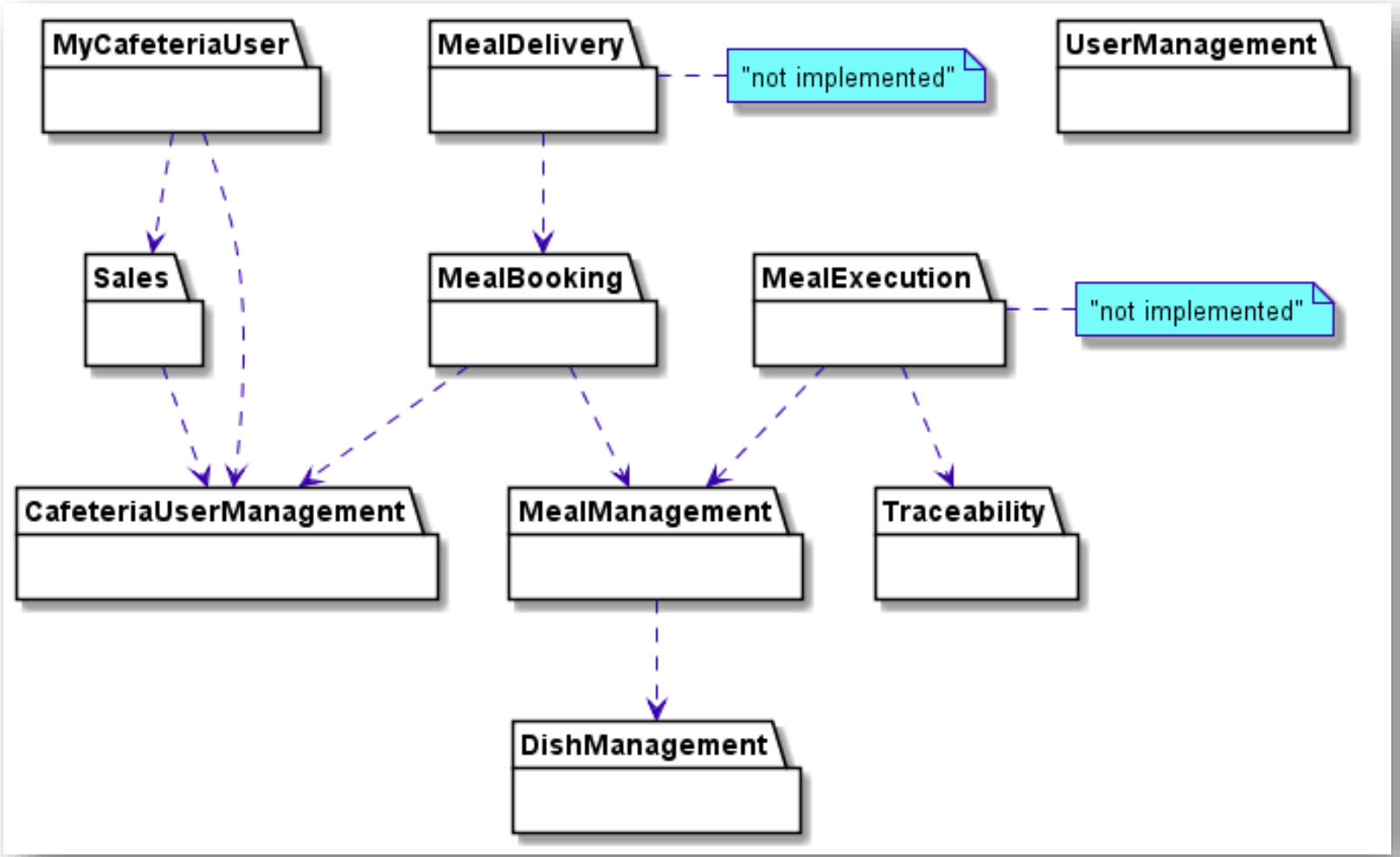
eCafeteria design decisions

- Layers
  - Business oriented
    - Presentation
    - Application
    - Domain
    - Persistence

DTO alternatives are also present

- Domain objects travel to UI for output

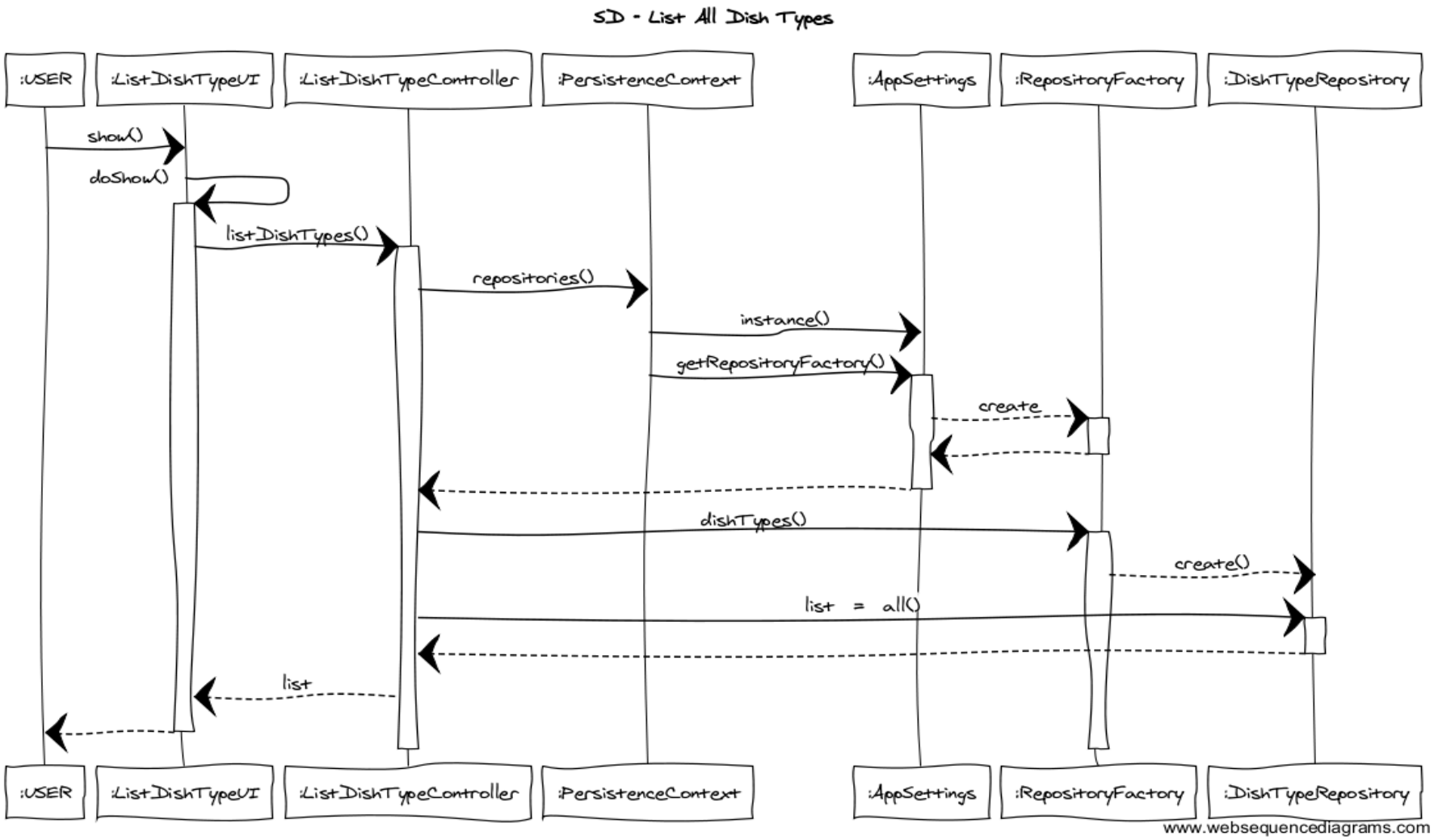
Core packages



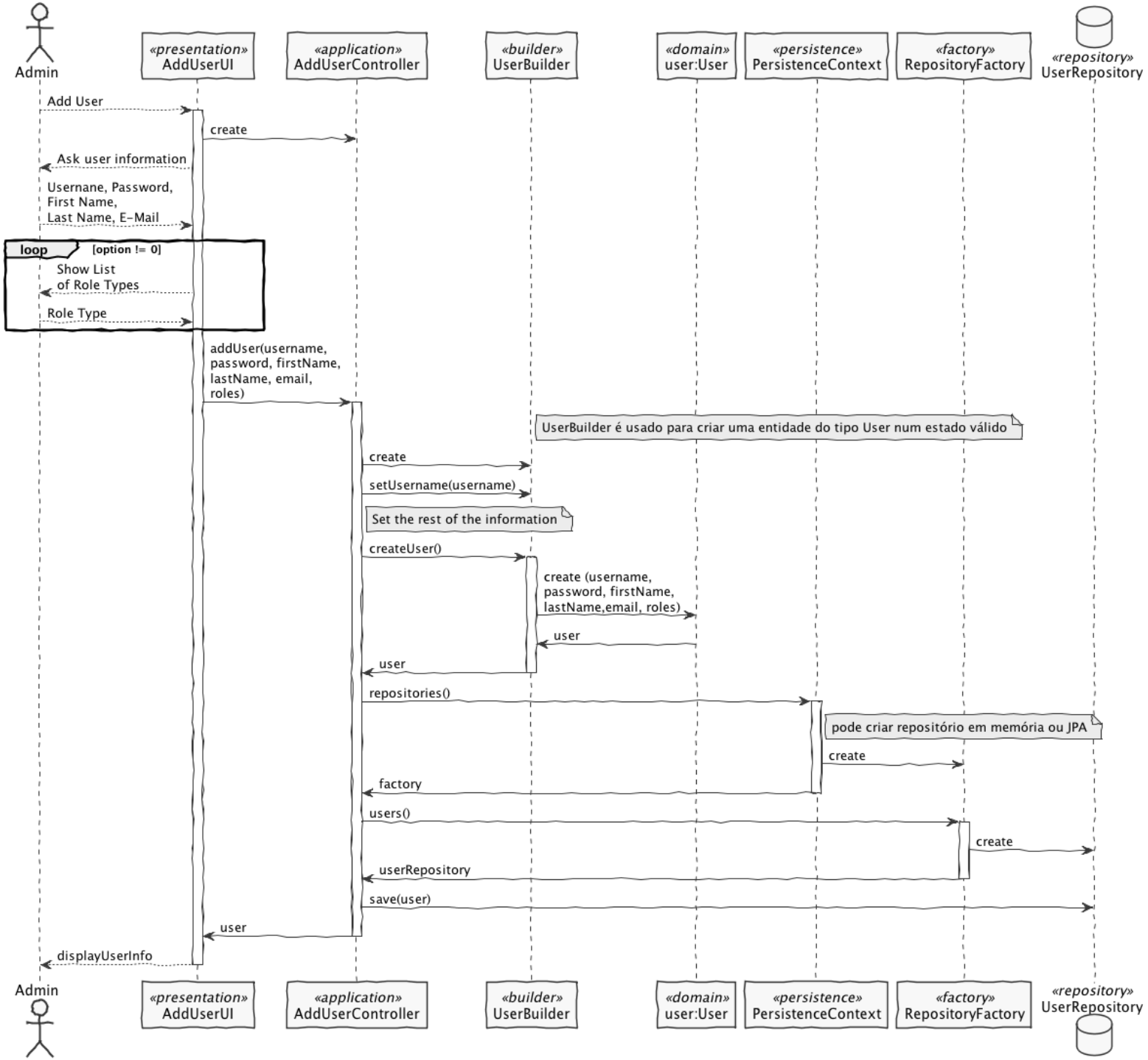
Some additional design decisions

- Support two repositories
  - In memory
  - Relational database
- Decide which repository implementation to use based on property file
- Bootstrap data
- Simple main menu

List Dish Types



Register New User



Domain invariants as unit tests

```
@Test
public void ensurePasswordHasAtLeastOneDigitAnd6CharactersLong()
{
	new Password("abcdefgh1");
}

@Test(expected = IllegalArgumentException.class)
public void ensurePasswordsSmallerThan6CharactersAreNotAllowed()
{
	new Password("ab1c");
}

@Test(expected = IllegalArgumentException.class)
public void ensurePasswordsWithoutDigitsAreNotAllowed() {
	new Password("abcdefgh");
}
```

Implemented Uses cases

- Backoffice > Kitchen
  - Add Material
  - List Material
- Backoffice > Chef
  - Add dish type
  - Edit dish type
  - Deactivate dish type
  - List dish types
  - Add dish
  - List dish
  - Add dish (DTO)
  - List dish (DTO)
  - Edit dish > nutricional info
  - Edit dish > price
  - Deactivate dish
  - Register meal
  - List meal
- Backofffice > Reporting
  - Dishes per type
  - High calories dishes
  - Dishes per caloric category
- Backoffice > Admin
  - Add user
  - List users
  - Deactivate user
  - Approve new user
- User
  - Signup
  - List movements
  - Book a meal
  - List my bookings
- POS
  - Recharge user account

Interesting Uses cases

- Domain objects or DTOs
  - Add dish vs. Add dish (DTO)
  - List dish vs List dish (DTO)
- Changing an attribute (value object) of an entity
  - Edit dish > nutricional info
  - Edit dish > price
- Reporting
  - Dishes per type
  - High calories dishes
  - Dishes per caloric category
- Transactional control
  - Approve new user
- Pub/Sub
  - Approve new user

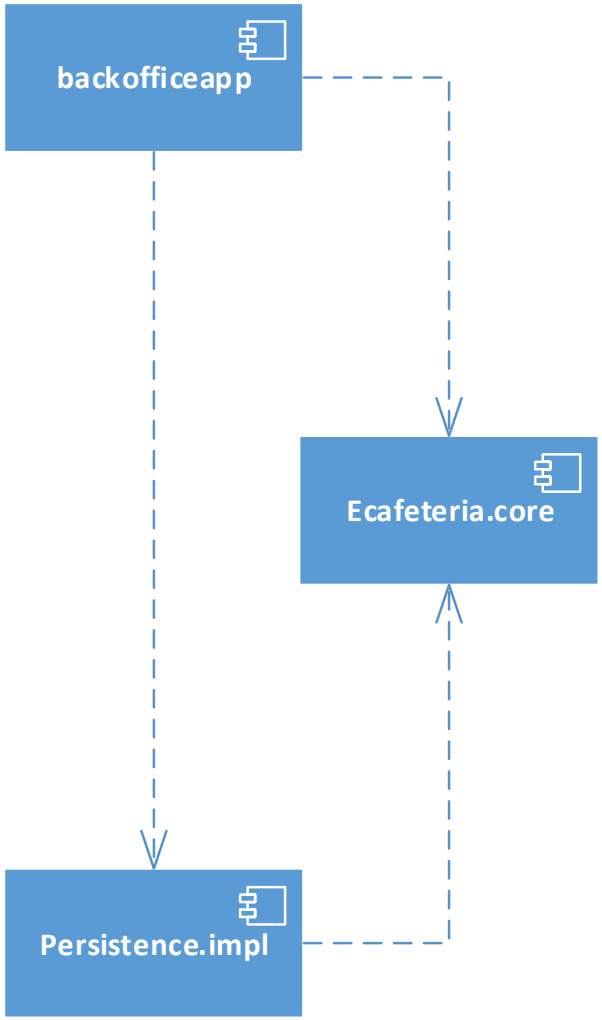
Next steps

1. Read project description
2. Discuss and clear assumptions in PL
3. Clone class' repository
4. Study framework code
5. Create base structure as suggested in this class
6. Analyse – design – code – test – document

Persistence

PersistenceContext

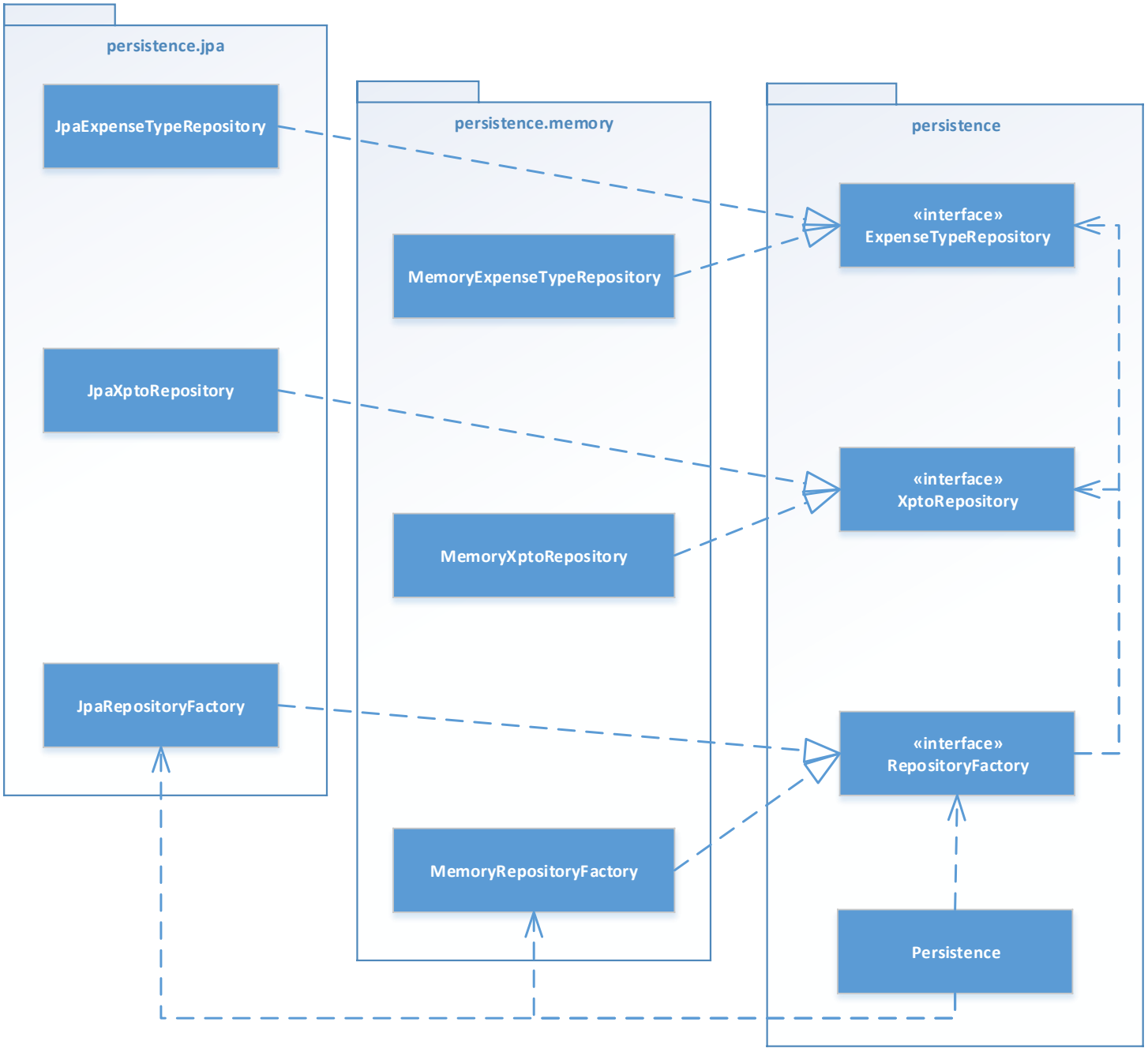
Persistence



- Separate the definition of repositories (core) from the actual implementation (persistence.impl)
- Apply "Abstract Factory" GoF pattern

Persistence

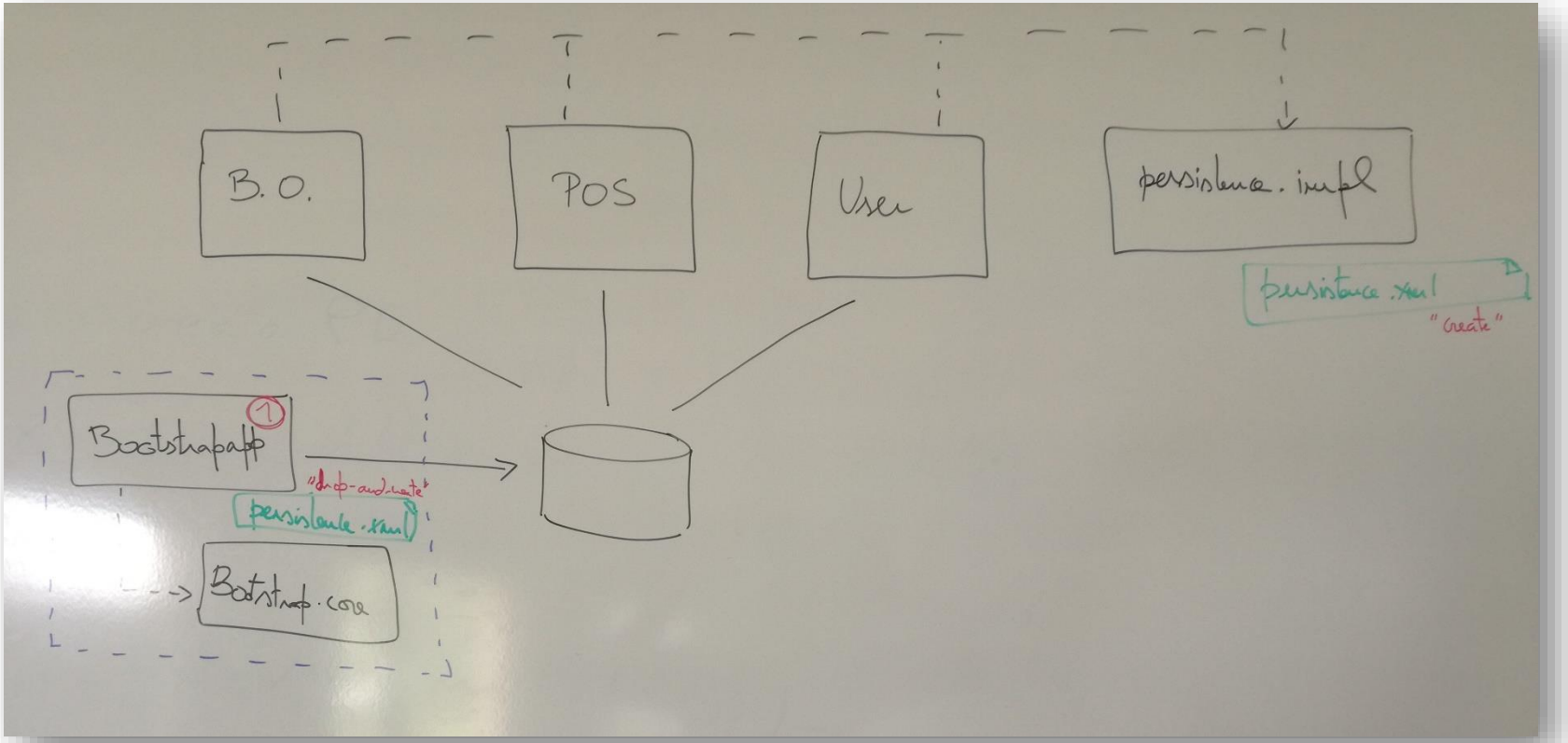
- Controller needs to access the repositories
- But we have three repository implementations
  - In memory
  - Relational database thru JPA
  - Relational database thru SpringData
- Hide persistence details from rest of the code
  - Interfaces
  - Dependency injection or Factories


persistence.jpa
JpaExpenseTypeRepository
JpaXptoRepository
JpaRepositoryFactory
persistence.memory
MemoryExpenseTypeRepository
MemoryXptoRepository
MemoryRepositoryFactory
persistence
«interface»
ExpenseTypeRepository
«interface»
XptoRepository
«interface»
RepositoryFactory
Persistence

Persistence Context Usage

```
public class RegisterMaterialController implements Controller {

    private final MaterialRepository repository = PersistenceContext.repositories().materials();

    public Material registerMaterial(String acronym, String description)
            throws DataIntegrityViolationException, DataConcurrencyException {
        Application.ensurePermissionOfLoggedInUser(ActionRight.MANAGE_KITCHEN);

        final Material mat = new Material(acronym, description);
        return this.repository.save(mat);
    }
}
```

Bootstrap

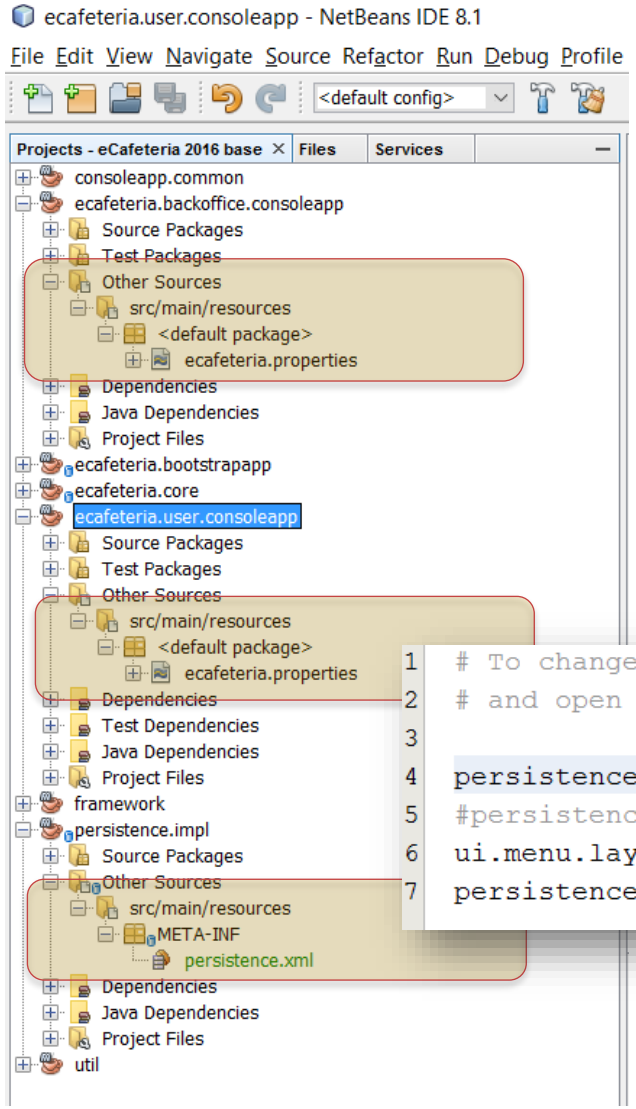


Separate BootstrapApp for database initialization

bootstrap

```
public class ECafeteriaBootstraper implements Action {

        @Override
        public boolean execute() {
                // declare bootstrap actions
                final Action[] actions = { new UsersBootstrap(), };
                // execute all bootstrapping
                boolean ret = false;
                for (final Action boot : actions) {
                        ret |= boot.execute();
                }
                return ret;
        }
}

public class UsersBootstrap implements Action {

        @Override
        public boolean execute() {
                registerAdmin();
                return false;
        }

        private void registerAdmin() {
                final String username = "admin";
                final String password = "admin";
                final String firstName = "John";
                final String lastName = "Doe";
                final String email = "john.doe@emai.l.com";
                final List<RoleType> roles = new ArrayList<RoleType>();
                roles.add(RoleType.Admin);

                final UserRegisterController userController = new UserRegisterController();
                userController.registerUser(username, password, firstName, lastName, email, roles);
        }
}
```

Resources




```
1 # To change this template, choose Tools | Templates
2 # and open the template in the editor.
3
4 persistence.repositoryFactory=eapli.ecafeteria.persistence.jpa.JpaRepositoryFactory
5 #persistence.repositoryFactory=eapli.ecafeteria.persistence.inmemory.InMemoryReposit
6 ui.menu.layout=horizontal
7 persistence.persistenceUnit=eapli.eCafeteriaPU
```

- Persistence.impl has persistence.xml
- Application projects define the properties file

Repository Implementations

```
public interface MaterialRepository extends DataRepository<Material, Long> {

    Material findByAcronym(String acronym);
}
```

```
public class InMemoryMaterialRepository extends InMemoryRepositoryWithLongPK<Material>
        implements MaterialRepository {

    @Override
    public Material findByAcronym(String acronym) {
        return matchOne(e -> e.id().equals(acronym));
    }
}
```

```
class JpaMaterialRepository extends CafeteriaJpaRepositoryBase<Material, Long>
        implements MaterialRepository {

    @Override
    public Material findByAcronym(String acronym) {
        return matchOne("e.acronym=:acronym", "acronym", acronym);
    }
}
```

Persistence Context

```
16  public class PersistenceContext {
17
18      private PersistenceContext() {
19      }
20
21      public static RepositoryFactory repositories() {
22          final String factoryClassName = Application.settings().getRepositoryFactory();
23          try {
24              return (RepositoryFactory) Class.forName(factoryClassName).newInstance();
25          } catch (ClassNotFoundException | IllegalAccessException | InstantiationException ex) {
26              // FIXME handle exception properly
27              Logger.getLogger(PersistenceContext.class.getName()).log(Level.SEVERE, null, ex);
28              return null;
29          }
30      }
31  }
32
```

```
1  # To change this template, choose Tools | Templates
2  # and open the template in the editor.
3
4  persistence.repositoryFactory=eapli.ecafeteria.persistence.jpa.JpaRepositoryFactory
5  #persistence.repositoryFactory=eapli.ecafeteria.persistence.inmemory.InMemoryRepositoryFactory
6  ui.menu.layout=horizontal
7  persistence.persistenceUnit=eapli.eCafeteriaPU
```

Persistence Context Usage

```
public class RegisterMaterialController implements Controller {

    private final MaterialRepository repository = PersistenceContext.repositories().materials();

    public Material registerMaterial(String acronym, String description)
            throws DataIntegrityViolationException, DataConcurrencyException {
        Application.ensurePermissionOfLoggedInUser(ActionRight.MANAGE_KITCHEN);

        final Material mat = new Material(acronym, description);
        return this.repository.save(mat);
    }
}
```

JPA Repositories (framework)

- JpaBaseRepository
  - Generic repository implementation that expects the entity manager factory to be injected by a container, e.g., web server
- JpaNotRunningInContainerBaseRepository
  - For scenarios where the code is not running in a container but transaction is managed by the outside, e.g., controller
- JpaTransactionalBaseRepository
  - For scenarios not running in a container but transactions are created and committed by each repository method; the connection is also closed automatically in each method.
- JpaAutoTxRepository
  - Dual behaviour to either have outside transactional control or explicit transaction in each method

Persistence

Controlo Transicional

Full transaction control by the repository

```
9   class JpaMaterialRepository extends CafeteriaJpaRepositoryBase<Material, Long>
            implements MaterialRepository {
11
12      @Override
        public Material findByAcronym(String acronym) {
14          return matchOne("e.acronym=:acronym", "acronym", acronym);
15      }
16  }
17
```

```
    class CafeteriaJpaRepositoryBase<T, K extends Serializable>
            extends JpaTransactionalRepository<T, K> {
17
18      CafeteriaJpaRepositoryBase(String persistenceUnitName) {
19          super(persistenceUnitName);
20      }
21
22      CafeteriaJpaRepositoryBase() {
23          super(Application.settings().getPersistenceUnitName());
24      }
25  }
26
```

Transaction control (1)

- Accepting a signup request needs to
  - Create a system user
  - Create a cafeteria user
  - Change the status of the signup request
- Three different aggregates!

Transaction control (2): use JpaAutoTxRepository

```
14  class JpaUserRepository extends JpaAutoTxRepository<SystemUser, Username>
15          implements UserRepository {
16
17      public JpaUserRepository(TransactionalContext autoTx) {
18          super(Application.settings().getPersistenceUnitName(), autoTx);
19      }
```

```
15  class JpaCafeteriaUserRepository
16          extends JpaAutoTxRepository<CafeteriaUser, MecanographicNumber>
17          implements CafeteriaUserRepository {
18
19      public JpaCafeteriaUserRepository(TransactionalContext autoTx) {
20          super(Application.settings().getPersistenceUnitName(), autoTx);
21      }
```

```
14  class JpaSignupRequestRepository
15          extends JpaAutoTxRepository<SignupRequest, Username>
16          implements SignupRequestRepository {
17
18      public JpaSignupRequestRepository(TransactionalContext autoTx) {
19          super(Application.settings().getPersistenceUnitName(), autoTx
20      }
21
```

Transaction control (3): explicit control by the controller

```
38  public class AcceptRefuseSignupRequestController implements Controller {
39
40      private final TransactionalContext TxCtx
                = PersistenceContext.repositories().buildTransactionalContext();
42      private final UserRepository userRepository
43              = PersistenceContext.repositories().users(TxCtx);
44      private final CafeteriaUserRepository cafeteriaUserRepository
45              = PersistenceContext.repositories().cafeteriaUsers(TxCtx);
46      private final SignupRequestRepository signupRequestsRepository
47              = PersistenceContext.repositories().signupRequests(TxCtx);
48
```

Transaction control (3): explicit control by the controller

```
49      public SignupRequest acceptSignupRequest(SignupRequest theSignupRequest)
50              throws DataIntegrityViolationException, DataConcurrencyException {
51          Application.ensurePermissionOfLoggedInUser(ActionRight.ADMINISTER);
52
53          if (theSignupRequest == null) {
54              throw new IllegalStateException();
55          }
56
57          // explicitly begin a transaction
58          TxCtx.beginTransaction();
59
60          SystemUser newUser = createSystemUserForCafeteriaUser(theSignupRequest);
61          createCafeteriaUser(theSignupRequest, newUser);
62          theSignupRequest = acceptTheSignupRequest(theSignupRequest);
63
64          // explicitly commit the transaction
65          TxCtx.commit();
66
67          return theSignupRequest;
68      }
```

Persistence

Acesso concorrente
Configuração servidor

Problema – Acesso Concorrente

- Exemplo Optimistic
  1. Utilizador A inicia a alteração do DishType "Fish" com a descrição "Fish dish" e altera para "Fly-fisch"
  2. Utilizador B inicia a alteração do DishType "Fish" com a descrição "Fish dish" e altera para "Cat-fisch"
  3. Utilizador A grava as alterações com sucesso
  4. Utilizador B tenta gravar as alterações mas não tem sucesso. Pois entretanto o registo já tinha sido alterado pela utilizador A.
- Exemplo Pessimistic
  1. Utilizador A inicia a alteração do DishType "Fish" com a descrição "Fish dish" e altera para "Fly-fisch"
  2. Utilizador B tenta iniciar a alteração do DishType "Fish" sem sucesso. O registo está bloqueado.
  3. Utilizador A grava as alterações com sucesso
  4. Utilizador B inicia a alteração do DishType "Fish" com a descrição "Fly-fisch"

Solução

- Como implementar a abordagem Optimistic?
  1. Setup: ambiente multi-utilizador
  2. Preparar as classes persistentes de domínio para acesso concorrente
  3. Apanhar a exceção de acesso concorrente
  4. Tratamento e propagação da exceção até ao ecrã do utilizador

Solução – 1. Setup servidor

- Preparar o H2 para modo servidor
  1. Criar uma nova unidade persistente ou simplesmente testar alterando a existente de:

```
<property name="javax.persistence.jdbc.url" value="jdbc:h2:..\db\ecafeteria;"/>
```

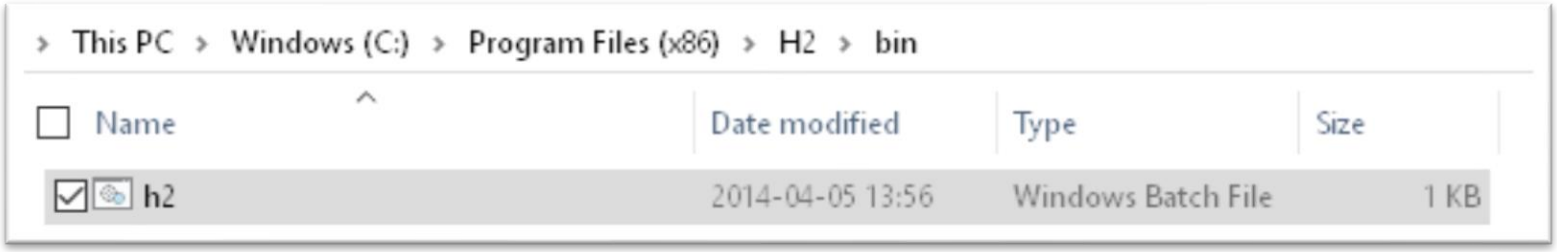
Para:

```
<property name="javax.persistence.jdbc.url"
          value="jdbc:h2:tcp://localhost/~/ecafeteria"/>
```

PS: //localhost ou outro endereço onde esteja o H2

Solução – 1. Setup servidor

- Preparar o H2 para modo servidor (cont.)
  2. Arrancar com o H2 localhost- basta executar a batch H2 localizada na pasta bin.



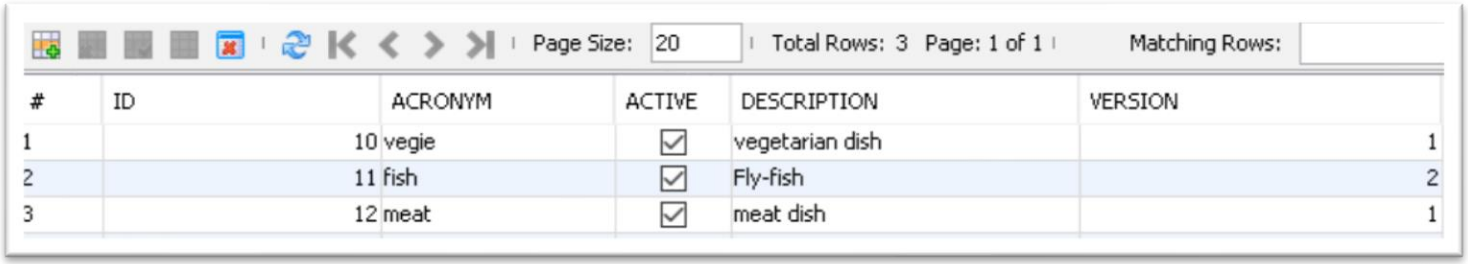
Solução – 2. Classes persistentes

- Acrescentar o controlo de versões.

```
@Entity
public class DishType implements AggregateRoot<String>, Serializable {

    private static final long serialVersionUID = 1L;

    // ORM primary key
    @Id
    @GeneratedValue
    private Long id;
    @Version
    private Long version;

    // business ID
    @Column(unique = true)
    private String acronym;
    private String description;
```

Solução – 2. Classes persistentes

- O atributo version será incrementado automaticamente sempre que exista uma alteração ao registo.
- Será este atributo que permitirá ao JPA perceber que a versão que está a ser gravada está desatualizada.



#	ID	ACRONYM	ACTIVE	DESCRIPTION	VERSION
1	10	vegie	☑	vegetarian dish	1
2	11	fish	☑	Fly-fish	2
3	12	meat	☑	meat dish	1

Solução – 3. Apanhar a exceção

- A exceção tem de ser apanhada no momento em que existe a tentativa de persistir o objeto. Onde o fazer no projeto PL?

Solução – 3. Apanhar a exceção

- É necessário apanhar a exceção nos métodos save e delete(classe JpaTxRepositoryBase)
- O JPA lança a exceção OptimisticLockException
  - "…cannot be merged because it has changed or been deleted since it was last read"
- Ex. Para o save:

```
try {
        tx = em.getTransaction();
        tx.begin();
        entity = em.merge(entity);
        tx.commit();
} catch (final OptimisticLockException exMerge) {
        throw new DataConcurrencyException(ex);
```

Solução -4. Tratamento e propagação

- Tratamento e propagação da exceção até ao ecrã do Utilizador
  1. Criar classe exception para a propagação para evitar dependências de todas as classes para o JPA

```
public class DataConcurrencyException extends Exception {

    public DataConcurrencyException(Throwable arg0) {
        super(arg0);
    }
}
```

Solução -4. Tratamento e propagação

- Tratamento e propagação da exceção até ao ecrã do Utilizador
  2. Declarar o throws no método onde é gerada a exception:

```
@Override
public T save(T entity) throws DataConcurrencyException {
```

Solução -4. Tratamento e propagação

- Tratamento e propagação da exceção até ao ecrã do Utilizador

3. Declarar o throws nos métodos por onde pode passar a exception:

```
public class ChangeDishTypeController implements Controller {

    public DishType changeDishType(DishType theDishType, String newDescription)
            throws DataConcurrencyException {
        Application.ensurePermissionOfLoggedInUser(ActionRight.MANAGE_MENUS);

        if (theDishType == null) {
            throw new IllegalStateException();
        }

        theDishType.changeDescriptionTo(newDescription);

        final DishTypeRepository repo = Application.repositories().dishTypes();
        return repo.save(theDishType);
    }
```

Solução -4. Tratamento e propagação

- Tratamento e propagação da exceção
  4. Apresentação ao utilizador

```
public class ChangeDishTypeUI extends AbstractUI {

    private final ChangeDishTypeController theController = new ChangeDishTypeController();

    protected Controller controller() {
        return this.theController;
    }

    @Override
    protected boolean doShow() {
        final Iterable<DishType> dishTypes = this.theController.listDishTypes();
        final SelectWidget<DishType> selector = new SelectWidget<>(dishTypes, new DishTypePrinter());
        selector.show();
        final DishType theDishType = selector.selectedElement();
        if (theDishType != null) {
            final String newDescription = Console
                    .readLine("Enter new description for " + theDishType.description() + ": ");

            try {
                this.theController.changeDishType(theDishType, newDescription);
            } catch (final DataConcurrencyException exMerge) {
                System.out.println("Data has changed or been deleted since it was last read. Please try again.");
            }
        }
```

Demonstração funcional

- Dois utilizadores a alterar o mesmo DishType

JPA –SQL executado pelo update

```
UPDATE DISHTYPE
SET DESCRIPTION = 'vegggg',
      VERSION = 2
 WHERE ID = 99
     AND VERSION = 1
```

- Além de alterar a descrição, incrementa a versão do registo para 2
- No Where além de pesquisar pela chave(ID), é confirmado que se está a alterar o registo com a versão 1. O que acontece se esta versão já não for 1?