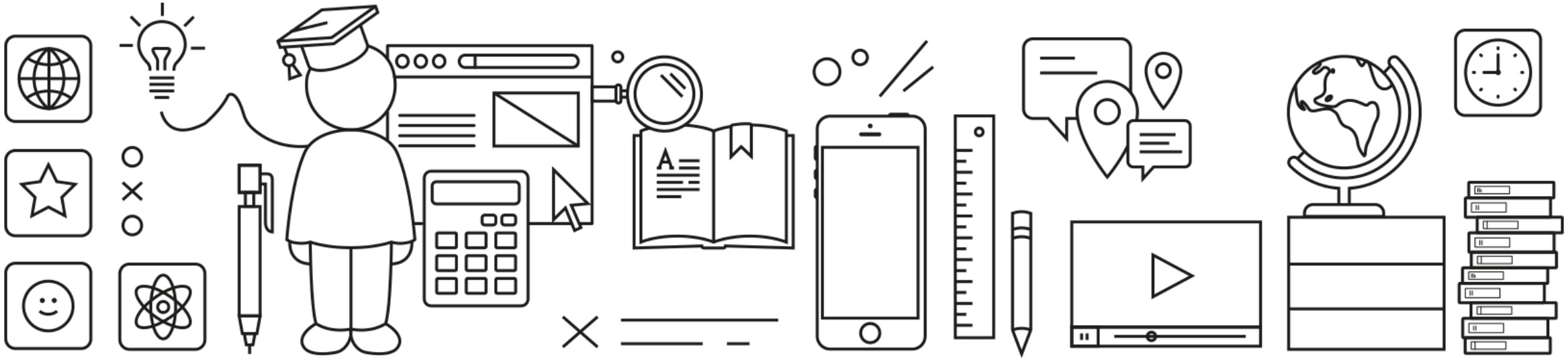
SAP Customer Experience

SAP Commerce Cloud Backoffice Framework Developer Training

Data Integration

THE BEST RUN SAP

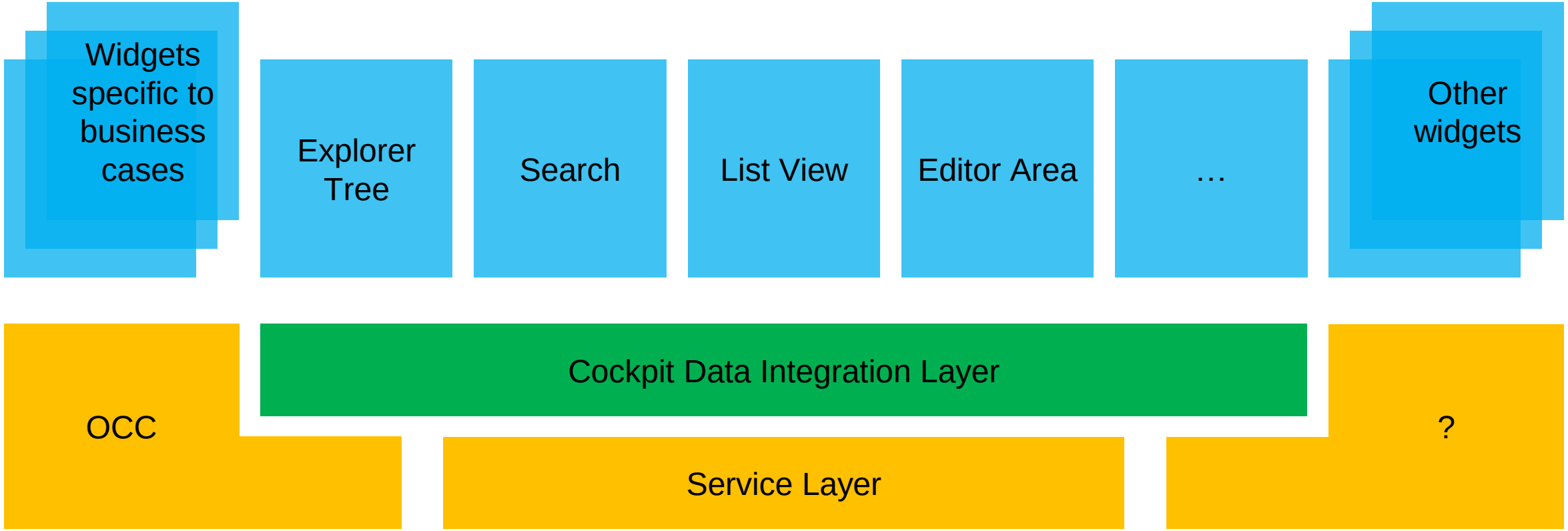


Overview

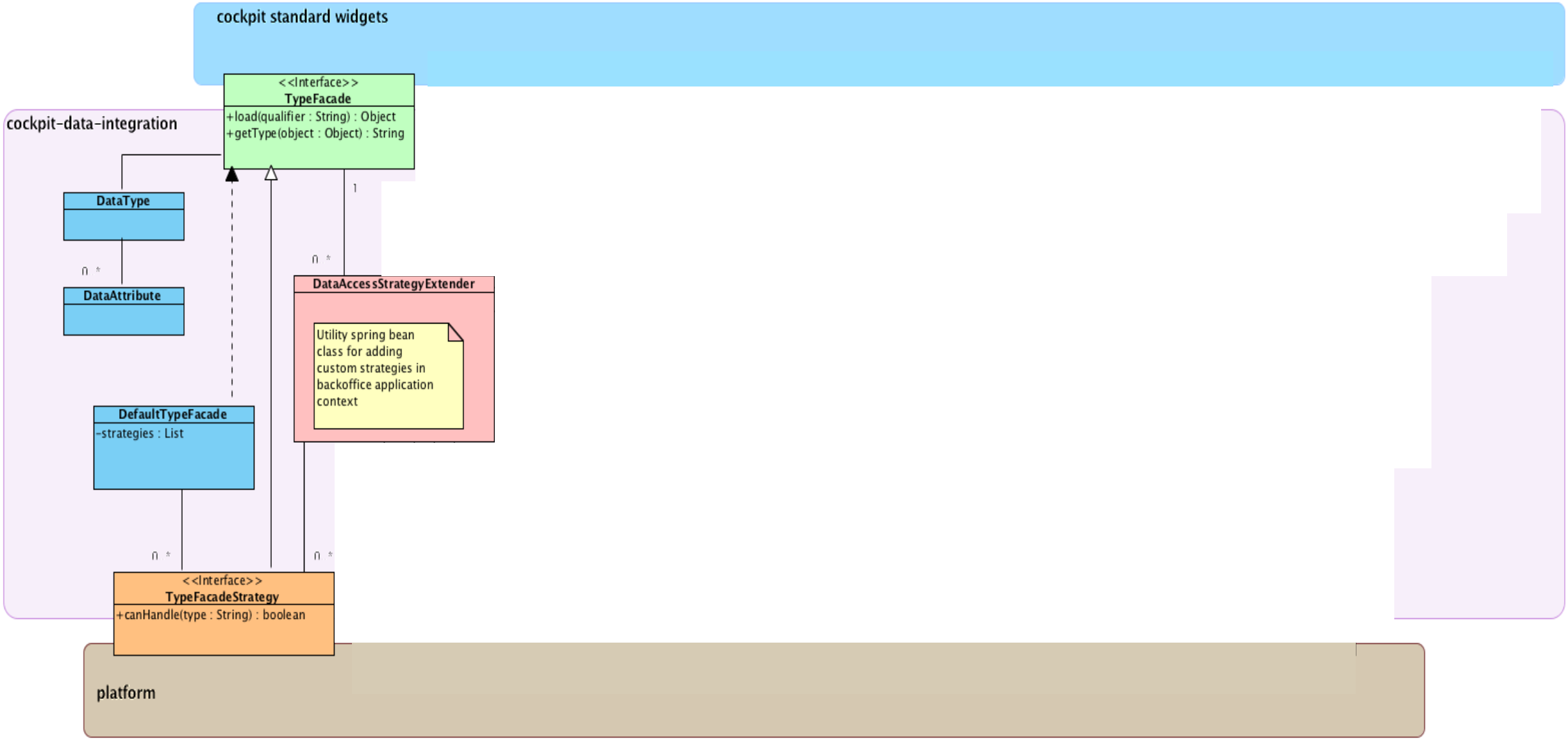
Backoffice Framework Data Integration

- Integration layer that supports **extendable** and **pluggable** domain models
  - Extendable because you can extend existing strategies
  - Pluggable because you can use any third-party system as the data layer
- Well-defined, cohesive SPIs
- Agnostic from any domain model
- Relevant design patterns involved: Façade, Strategy, Chain of Responsibility, Builder

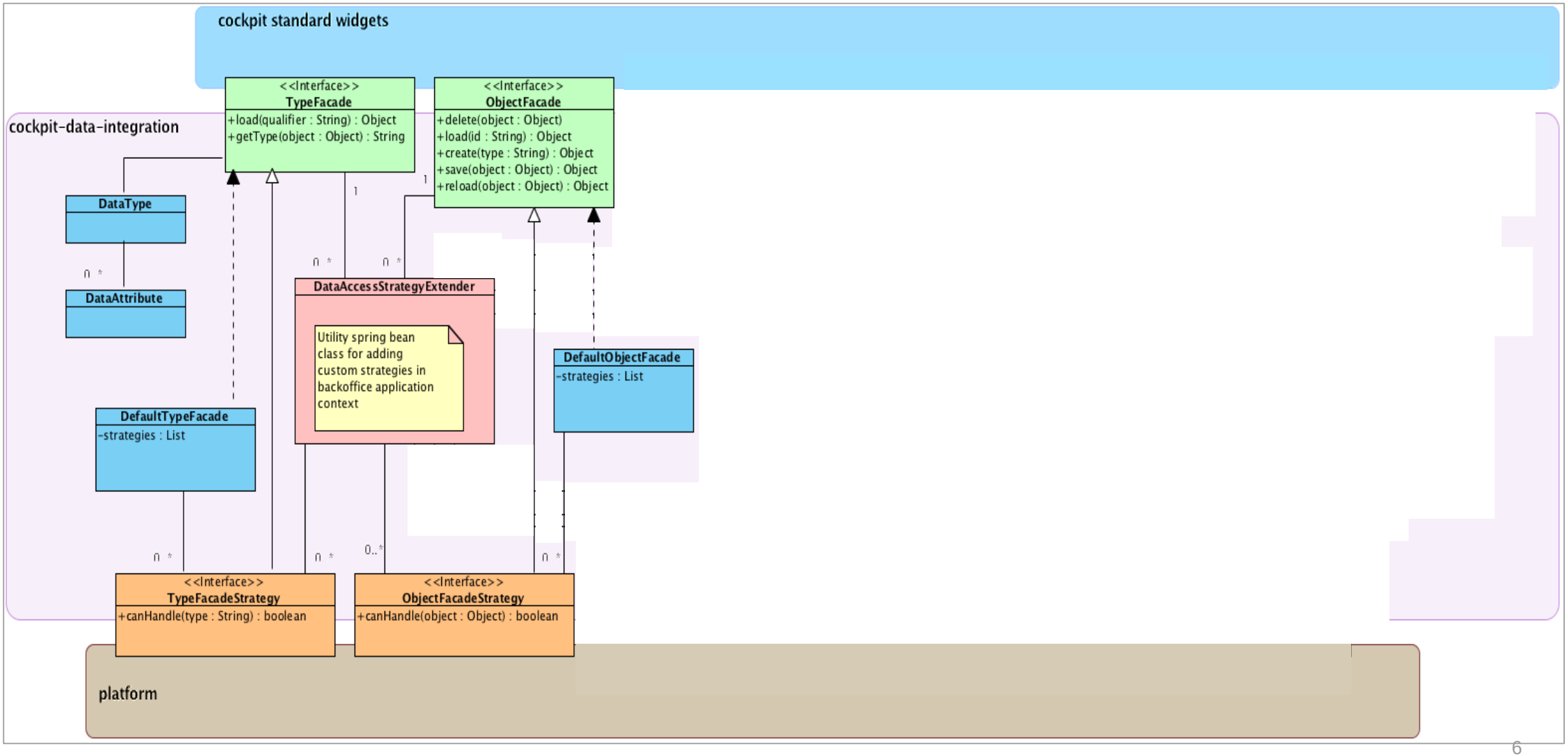
Overview - Layered View



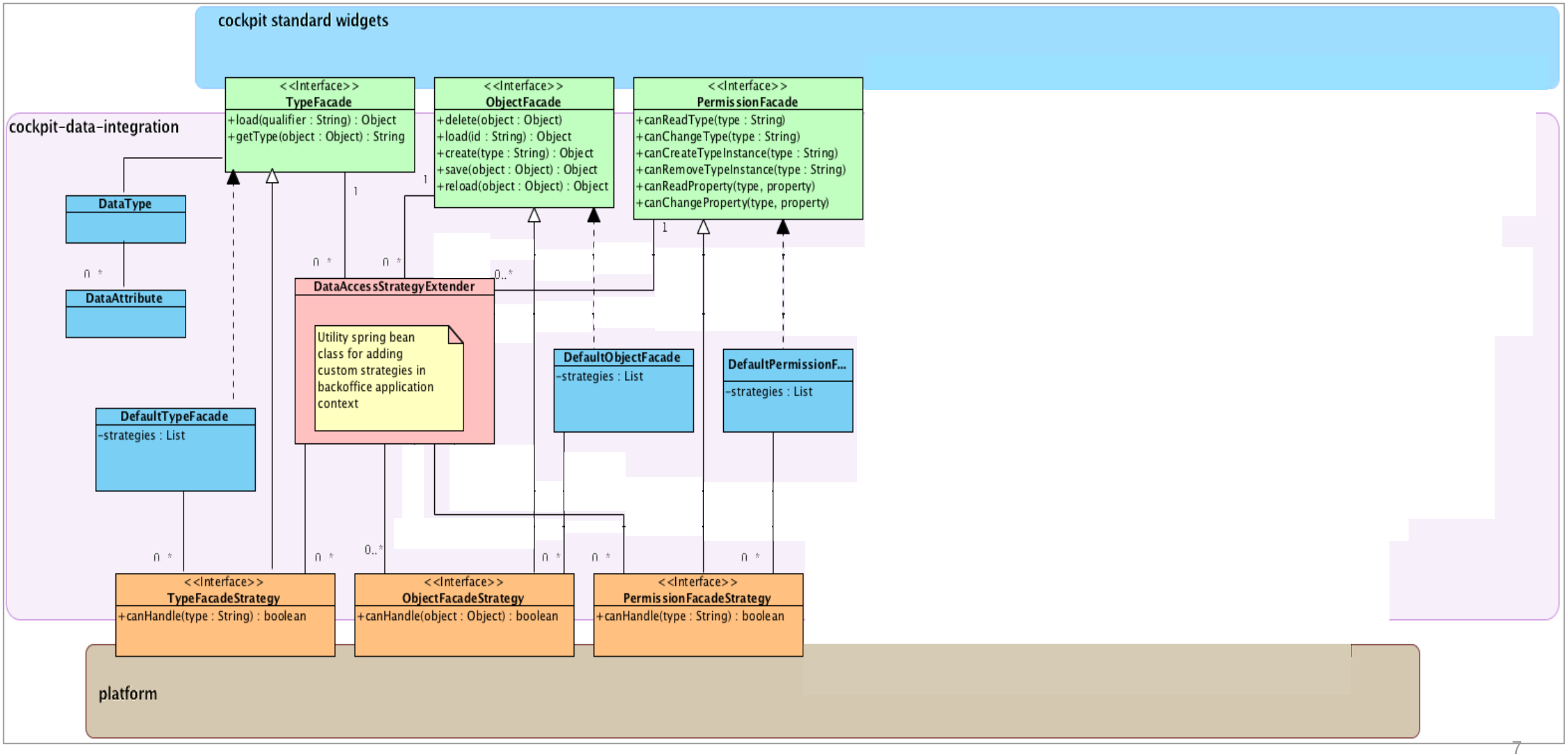
Overview - Architecture



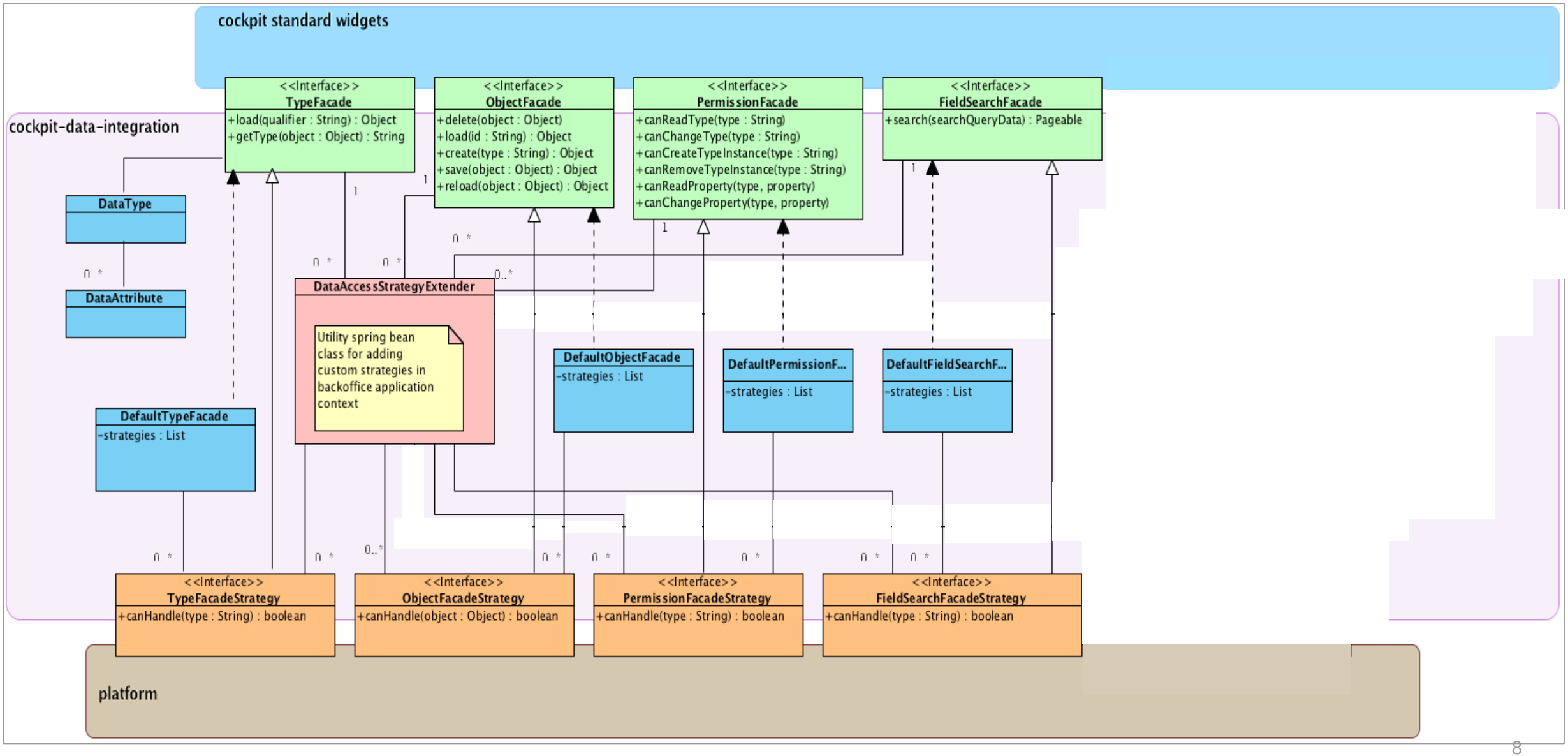
Overview - Architecture



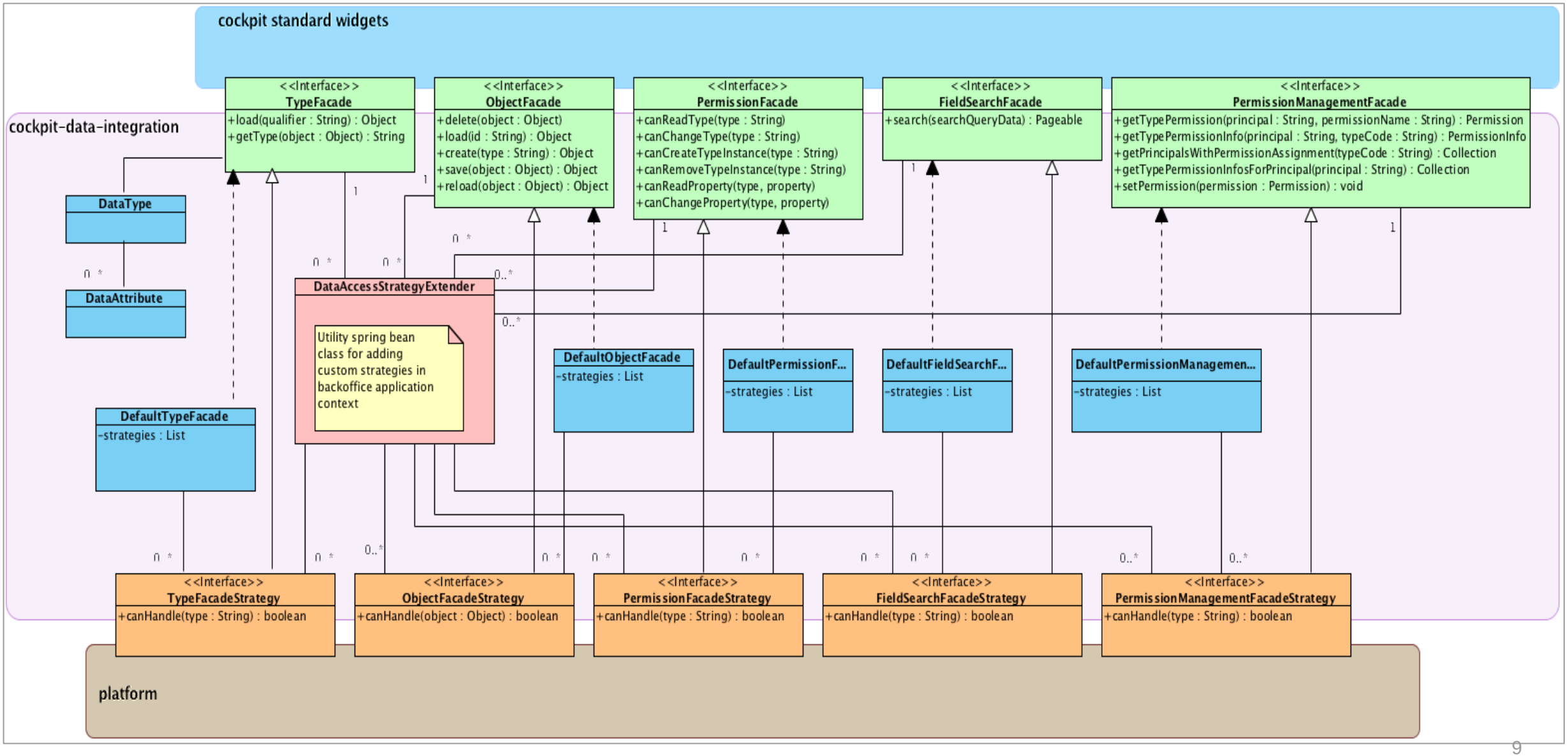
Overview - Architecture



Overview - Architecture



Overview - Architecture



SPI's Façades

Façades are used by all widgets to access any data – each façade will have an associated Strategy implementation

- **TypeFacade**: Type meta information
- **ObjectFacade**: For performing CRUD operations on any data instance
- **FieldSearchFacade**: Search engine based on field conditions
- **PermissionFacade**: Type- and instance-based permission checking
- **PermissionManagementFacade**: Permission definition and management for principals

SPI’s Strategies

- Specify a Strategy for each Façade for fine-grained handling of types
- Default implementations provided by SAP Commerce Cloud to integrate with the Commerce Platform OOTB:
  - `DefaultPlatformTypeFacadeStrategy`
  - `DefaultPlatformObjectFacadeStrategy`
  - `DefaultPlatformPermissionFacadeStrategy`
  - `DefaultPlatformFieldSearchFacadeStrategy`
  - `DefaultPlatformPermissionManagementFacadeStrategy`

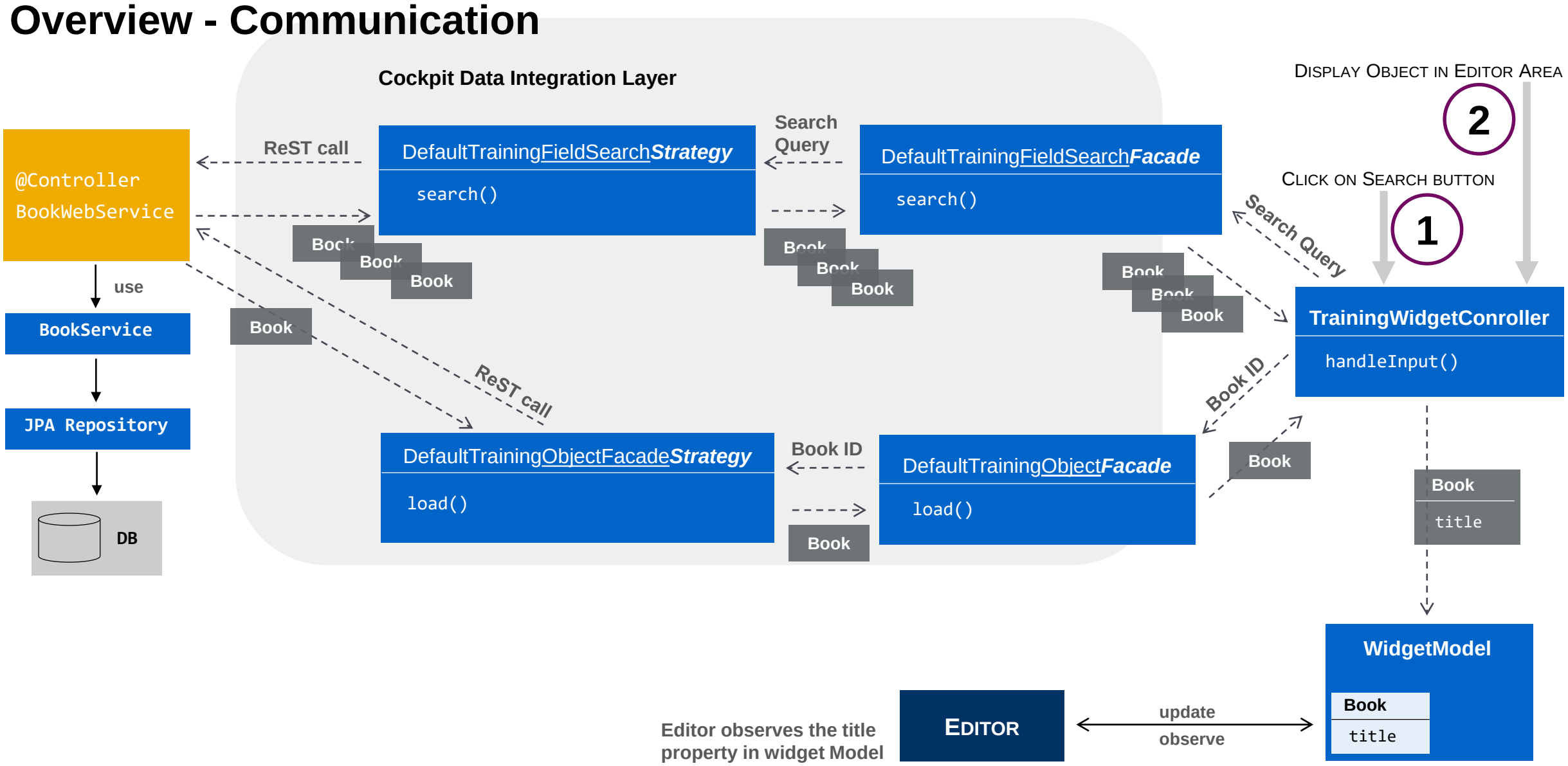
Extending the Data Access Strategy

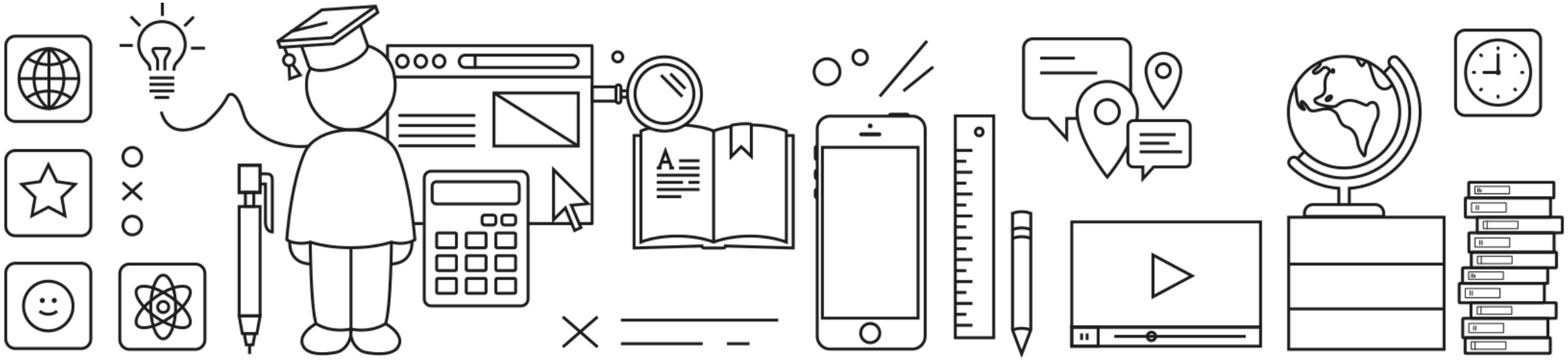
- Backoffice defines a `DataAccessStrategyExtender` bean
- To extend the data access strategy, inject your own strategies
  - type-specific
  - prioritized – referenced in order listed
  - each of the façade implementations has a list property
- All strategies must implement `canHandle()`
- Must also implement other methods depending on the specific strategy
  - Ex: `ObjectFacadeStrategy` must also implement
    - `load()`, `create()`, `save()`, `reload()` and others

Spring DataAccessStrategyExtender Bean

```
<bean class="com.hybris.cockpitng.dataaccess.util.DataAccessStrategyExtender">
	<property name="typeFacadeStrategies">
		<list>
			<!-- inject your Strategy here -->
			<ref bean="trainingTypeFacadeStrategy" />
			...
		</list>
	</property>
	<property name="objectFacadeStrategies">
		<list>
			<!-- inject your Strategy here -->
			<ref bean="trainingObjectFacadeStrategy" />
			...
		</list>
	</property>
	...
</bean>
```

Overview - Communication





SPIs

Metadata SPI

- Provides Metadata info for Types

- `TypeFacadeStrategy` – must implement
  - ❑ `load()`
  - ❑ `getType()`
  - ❑ `getAttributeDescription()`
- `DataType` and `DataAttribute`
  - ❑ Contain type and attribute metadata
  - ❑ Builders are provided to simplify loading type info
- Consider caching your metadata!

CRUD SPI

- Handles actual CRUD operations

- `ObjectFacadeStrategy` – must implement
  - `create()`
  - `save()`
  - `load()`
  - `reload()`
  - `delete()`

Search SPI

- Handles Searches

– `FieldSearchFacadeStrategy` - implements
  - `Pageable search(SearchQueryData)`

– `SearchQueryData`
  - Holds search criteria values
  - `SortData` has ordering information

– `Pageable` holds pageable list of results

Security SPIs

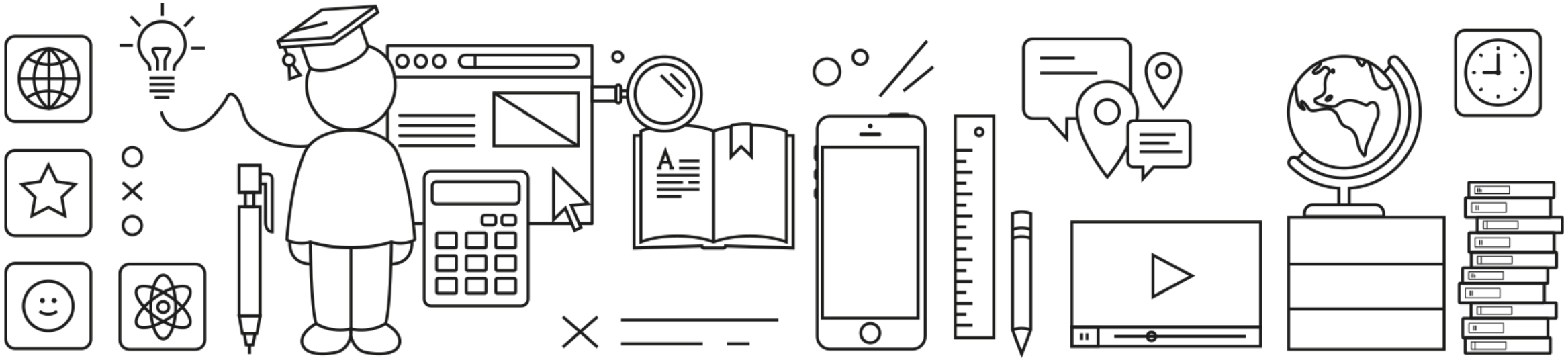
Permission Checking SPI

`PermissionFacadeStrategy` –
Interface that defines the following abstract methods:

- `canReadType()`
  `canChangeType()`
- `canReadInstanceProperty()`
  `canChangeInstanceProperty()`
- `canReadProperty()`
  `canChangeProperty()`
- `canCreateTypeInstance()`
  `canRemoveTypeInstance()`
- `canReadInstance ()`
  `canChangeInstance()`
  `canRemoveInstance()`
- `canChangeTypePermission()`
- `canChangePropertyPermission()`
- …and some locale-specific methods

Security SPIs, cont.

- Permission Management SPI
  - `PermissionFacadeStrategy` – implements
    - `getTypePermission()`
    - `getTypePermissionInfo()`
    - `getPrincipalPermissionInfo()`
    - `getFieldPermission()`
    - `getFieldPermissionInfo()`
    - `getPrincipalsWithPermissionAssignment()`
    - `getTypePermissionInfosForPrincipal()`
    - `setPermission()`
    - `updatePermissionInfo()`
    - `deletePermission()`



SPIs in Practice

Did we use any of it in the Bookstore Backoffice?

We used only one of the SPIs directly: the metadata SPI

BookDetailsController.java

```
@WireVariable
private TypeFacade typeFacade;
```

Also, we used services which, in turn, used the SPIs

BookDetailsController.java

```
@WireVariable
private PropertyValueService propertyValueService;
@WireVariable
private CockpitLocaleService cockpitLocaleService;
@WireVariable
private ObjectPreviewService objectPreviewService;
```

Implementations Used by Backoffice

- All the SPIs and services are interfaces that should be implemented
- There are default implementations which are in the Data Integration layer
- The Spring Context configuration decides which implementations will be used
- The Spring configuration of the Data Integration layer lays in the Data Integration module jar file (`cockpit-data-integration-18.11.0-RC2.jar`)
- You can *change* this configuration inside the ***backoffice*** extension
- This is the extension where the instance of the Backoffice *application* resides
- Remember that what we were doing in this training was to add stuff to an existing Backoffice application

Implementations Used by Backoffice, cont.

The Spring context configuration file can be found here:

`${HYBRIS_BIN_DIR}/ext-backoffice/backoffice/web/webroot/WEB-INF/backoffice-web-spring.xml`

For instance, in the following snippet, `cockpitLocaleService` is being set to a bean called `backofficeLocaleService` defined in the Backoffice application's Spring context.

backoffice-web-spring.xml

```
...
  <alias name="backofficeLocaleService" alias="cockpitLocaleService"/>
  <bean id="backofficeLocaleService" class="com.hybris.backoffice.i18n.BackofficeLocaleService">
    <property name="i18nService" ref="i18nService"/>
    <property name="cockpitLocalesFactory" ref="cockpitLocalesFactory"/>
    <property name="cockpitConfigurationService" ref="cockpitConfigurationService"/>
    <property name="cockpitProperties" ref="cockpitProperties"/>
    <property name="widgetConfigurationContextDecoratorList"
                          ref="widgetConfigurationContextDecoratorList"/>
    <property name="authorityGroupService" ref="authorityGroupService"></property>
  </bean>
...
```

How to find the beans?

For all the SPIs and services, there are beans defined in the Spring context

There’s a simple naming convention to follow:

- The bean’s name is the same as the Class’, with the first letter switched to lower case

For instance:

- `TypeFacade` has a bean called `typeFacade` associated with it
- `PropertyValueService` has a bean called `propertyValueService` associated with it

Where to learn about the Backoffice services API?

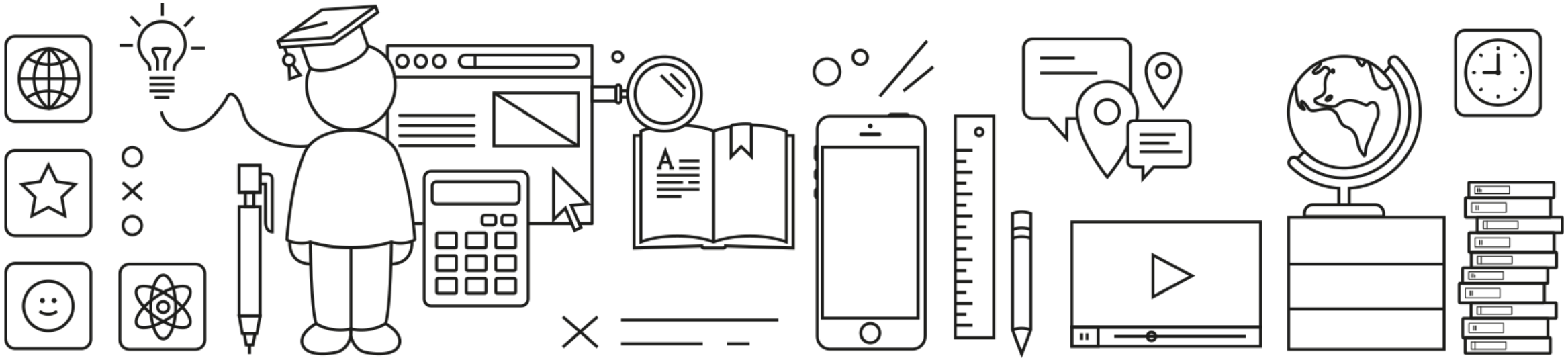
At the moment, there's no better place than the Backoffice framework's API

Backoffice Framework's API

For example, have a look at the services used in the `BookDetailsController`:

- PropertyValueService and its default implementation DefaultPropertyValueService
- CockpitLocaleService and its default implementation DefaultCockpitLocaleService
- ObjectPreviewService and its default implementation DefaultObjectPreviewService

- * We're also using a utility class called BackofficeTypeUtils



Property Accessors

Property Accessors

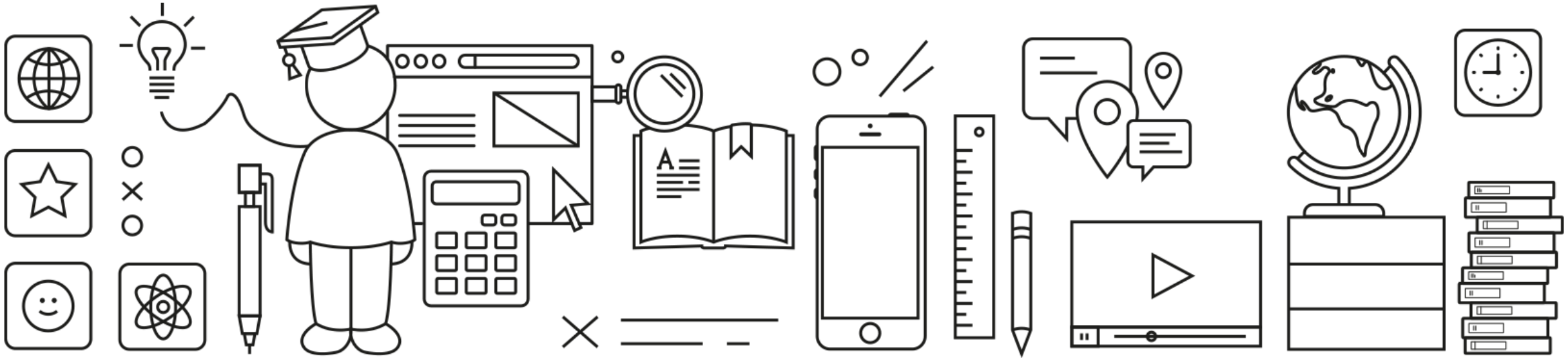
- Uses standard Spring Property Accessors to read/write data instance properties
- Provides a custom way to resolve properties when they are accessed for reading or writing
- Custom implementations pluggable via Spring DI

Property Accessors, cont.

```
interface PropertyAccessor {

	Class<?>[] getSpecificTargetClasses();

	boolean canRead ...

	TypedValue read ...

	boolean canWrite ...

	void write ...

}
```

Property Accessors, cont.

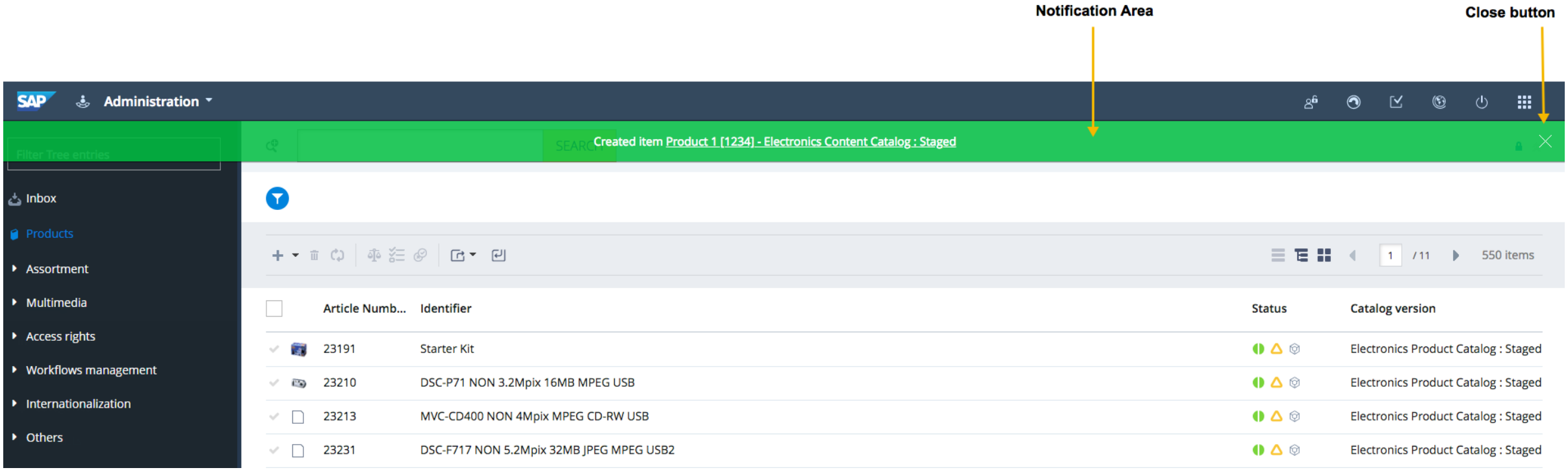
```
<alias name="myPropertyAccessors" alias="propertyAccessors"/>
<bean id="myPropertyAccessors" parent="defaultPropertyAccessors">
  <property name="sourceList">
    <list merge="true">
      <ref bean="myModelPropertyAccessor"/>
      ...
      <ref bean="myGenericModelPropertyAccessor"/>
    </list>
  </property>
</bean>
```



Notification API

Notification Area Widget

- To display notifications, this widget can be inserted anywhere in your application
- Listens for global and custom notifications
- Other widgets can publish notification events using `NotificationsUtils` class
- Contains Close button and Notification Area

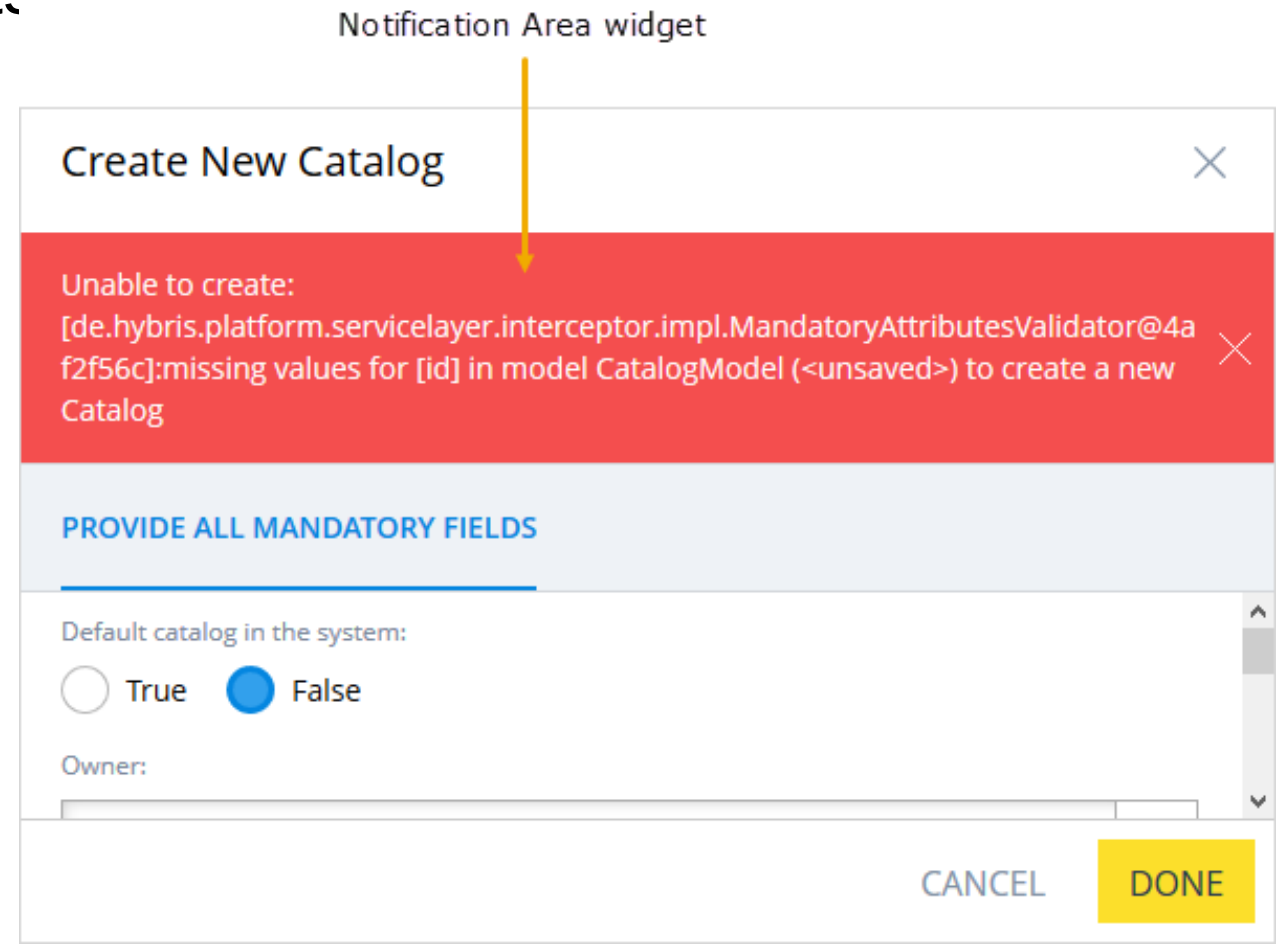


Notification Area Widget, continued

- Can be added into other widgets
  - Add a widget slot, e.g.:

configurableflow.zul

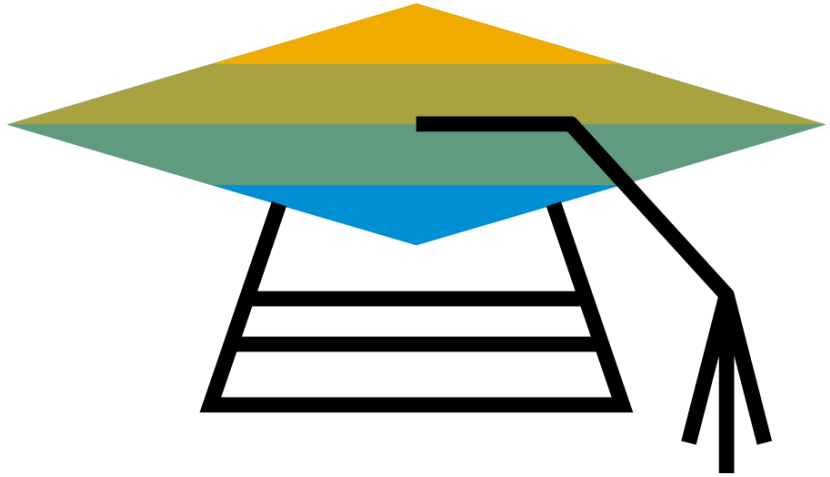
```
<widgetslot slotID="wizardNotificationarea"
    sclass="yw-notification"/>
```



Notification API

- `NotificationUtils.notifyUser()`
- `NotificationUtils.notifyUserVia()`
- `Message` – String to display
- `NotificationEvent.`*`Type`*
  - *`SUCCESS`*
  - *`INFO`*
  - *`WARNING`*
  - *`FAILURE`*
- `Notification ID` – a widget may choose to listen only for a specific `ID`, `notifyUserVia` can send an `ID`
- `Behavior` – sticky or timed

Thank you.



THE BEST RUN SAP®