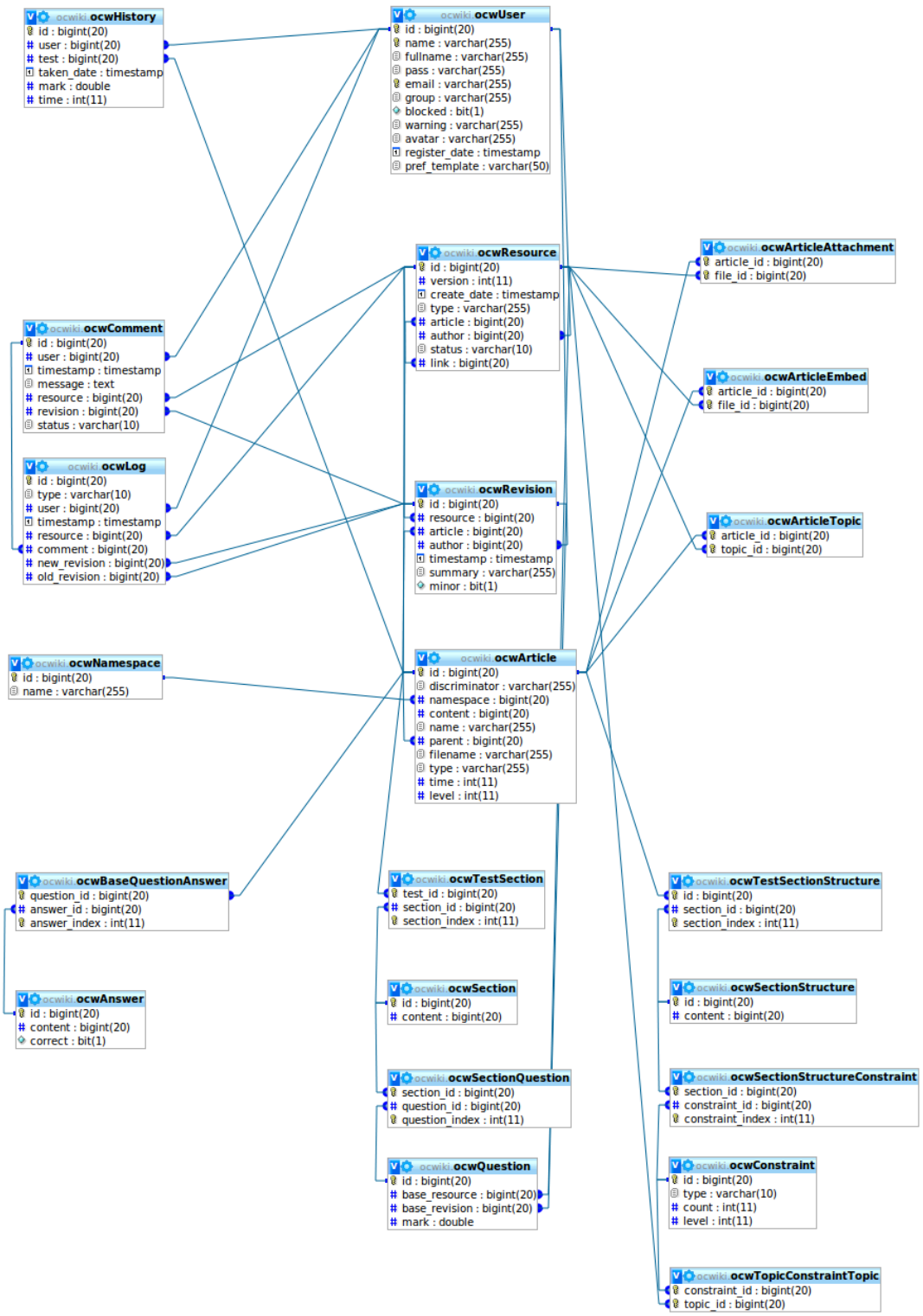
Tài liệu thiết kế

Mục lục

Thiết kế cơ sở dữ liệu



Action

Module

Tại sao cần module?

Để phân chia hệ thống thành các phần nhỏ, có thể test được (unit test), có thể bật/tắt, cấu hình, thay thế... mà không cần phải dịch lại mã nguồn.

Cấu hình

Thẻ <module> điều khiển mọi hoạt động của module, để chương trình nhận diện một module, cần tạo một thẻ <module> với các thẻ con sau:

- name: tên của module
- title: tiêu đề
- class (tuỳ chọn): lớp trợ giúp
- position: mã số vị trí đặt module (tuỳ thuộc vào template)
- order: thứ tự xuất hiện của module (từ nhỏ đến lớn)
- page (tuỳ chọn): tên trang jsp, tương đối với thư mục /templates/xxx/modules, nếu không có thẻ này thì trang mặc định là <tên module>.jsp
- inAction* (tuỳ chọn): chỉ bật module với action được chỉ định
- articleType* (tuỳ chọn): chỉ bật module khi action liên quan đến kiểu bài viết được chọn
- loginRequired (tuỳ chọn): chỉ bật module nếu người dùng đã đăng nhập
- requiredGroup* (tuỳ chọn): chỉ bật module nếu người dùng thuộc một trong những nhóm được chỉ định (có thể có nhiều thẻ này trong một thẻ module)

Các thẻ đánh dấu * có thể xuất hiện nhiều hơn một lần trong định nghĩa module.

Ví dụ:

```
<module>
    <name>question-test</name>
    <title>Được sử dụng trong đề thi</title>
    <class>oop.module.question.UsedByTestModule</class>
    <position>action-bottom</position>
    <order>4</order>
    <page>question/test.jsp</page>
    <inAction>article.view</inAction>
    <articleType>oop.data.BaseQuestion</articleType>
</module>
```

Lớp trợ giúp

Để lấy dữ liệu từ CSDL cung cấp cho module có thể sử dụng lớp trợ giúp extends từ DefaultModule. Trong lớp này cần cài đặt hàm init() để truy cập CSDL và các getter để trang JSP có thể lấy được dữ liệu này.

Ví dụ:

```
public class ChildrenModule extends DefaultModule {

	private List<Resource<Topic>> children;

	@Override
	public void init() throws Exception {
		children = TopicDAO.fetchChildren(getResource().getId());
	}

	public List<Resource<Topic>> getChildren() {
		return children;
	}

}
```

Lưu ý: Không đặt lệnh truy cập CSDL trong getter vì hàm này có thể được gọi rất nhiều lần dẫn đến việc truy cập CSDL liên tục một cách không cần thiết.

Trang JSP

Trong trang JSP đối tượng của lớp trợ giúp có thể được truy cập thông qua biến `module`.

Ví dụ:

```
<ul>
<c:forEach items="${module.children}" var="child">
    <li><ocw:articleLink resource="${child}"></ocw:articleLink></li>
</c:forEach>
</ul>
```

URL

Bài viết
${homeDir}/article/${id}
Hành động (action)
${homeDir}/action/${code}
Cơ sở dữ liệu

Tags

actionButton & actionLink

Tạo liên kết đến hành động, dùng thẻ này để code rõ ràng và dễ thay đổi hơn.
Cú pháp:
<ocw:action(Button|Link) name=... [id=...] [class=...] [confirm=...]>
    [<ocw:param name=... value=...>]
    content
</ocw:action(Button|Link)>
Trong đó confirm là một lời gọi hàm JavaScript trả về true/false.
Ví dụ:

<ocw:actionLink name="article.changelog">

<ocw:param name="article" value="${action.resource}" />

Nhật kí

</ocw:actionLink>

articleButton & articleLink

<ocw:article(Button|Link) resource=resource_object>
</ocw:article(Button|Link)>
Trong đó resource_object là một đối tượng kiểu oop.data.Resource.
Ví dụ:

<ocw:articleButton resource="${action.resource}">

Quay về câu hỏi

</ocw:articleButton>

userLink

<ocw:userLink user=user_object>

content

</ocw:userLink>

user_object là đối tượng kiểu oop.data.User.

form

Sinh ra một thẻ form chứa sẵn editToken bên trong.
<ocw:form action=... [id=...] [class=...] [method=...] >
    content
</ocw:form>

parse

Chuyển đổi văn bản từ định dạng nội bộ sang HTML, khi trình bày bất kì bài viết nào cần bọc phần văn bản trong thẻ này. Cú pháp:
<ocw:parse resource=... [content=...]>

[…]
</ocw:parse>
Trong đó resource là đối tượng lớp oop.data.Resource, trong văn bản có thể có các nội dung động lấy thông tin từ đối tượng này. Thuộc tính content là nội dung cần chuyển đổi, thuộc kiểu String hoặc oop.data.Text. Thuộc tính content là tuỳ chọn, thay vào đó có thể đặt văn bản vào thân thẻ.

Functions