Экзамен

№2 Основные технологии, расширение типовых возможностей системы

по платформе «Bitrix Framework»

«Академия 1С-Битрикс»

Версия 4.0.3, 02.07.2024 г.

Документ обновляется, актуальная версия доступна по ссылке https://academy.1c-bitrix.ru/~ex2desc

Входные требования

- Наличие базовых знаний HTML, CSS, JavaScript
- Навыки программирования на PHP, построения SQL запросов
- Сертификат по экзамену «№1 Интеграция дизайна и настройка»
- Сертификат по онлайн-тесту: «Администратор. Базовый»
  https://dev.1c-bitrix.ru/learning/course/index.php?COURSE_ID=35
- Сертификат по онлайн-тесту: «Технология Композитный сайт»
  https://dev.1c-bitrix.ru/learning/course/index.php?COURSE_ID=39

Предмет экзамена

Оцениваются навыки по правильному выбору и применению **основных технологий** платформы Bitrix Framework, необходимых для решения задач выходящих за рамки типового функционала системы.

Материалы для подготовки

- Курс «№2 - Основные технологии и расширение типовых возможностей системы»
  https://academy.1c-bitrix.ru/education/?COURSE_ID=66&INDEX=Y
- Урок «Инфоблоки – Архитектура»
  https://academy.1c-bitrix.ru/education/index.php?COURSE_ID=67&LESSON_ID=6377&LESSON_PATH=5864.6377
- Урок «Инфоблоки - Оптимизация выборки»
  https://academy.1c-bitrix.ru/education/index.php?COURSE_ID=67&LESSON_ID=6472&LESSON_PATH=5864.6472
- Документация по Bitrix Framework
  https://dev.1c-bitrix.ru/learning/course/index.php?COURSE_ID=43
- Документация по API
  https://dev.1c-bitrix.ru/api_help/

Формат экзамена

- Длительность: 4 часа
- Решение практических заданий путем кастомизации системы с помощью технологий и API Bitrix Framework.
- Удалённое подключение через терминал к экзаменационному серверу https://academy.1c-bitrix.ru/certification/exam-server.php.

  Внимание! При подготовке учитывайте, работа с сервером по терминалу - не такая быстрая, как локальная разработка. Тестовые билеты нужно решать с запасом времени. Познакомитесь с окружением заранее, чтобы во время экзамена не отвлекаться на незнакомую среду.

- Во время экзамена можно пользоваться только справочными материалами, доступными на экзаменационном сервере: локальные CHM файлы с документацией по API Framework и PHP. «Гуглить» и использовать «свои заготовки» - запрещено.

Оценка решения

Для успешного прохождения экзамена необходимо выполнение нескольких критериев:

- **Все задания из билета должны быть решены**

  Объем заданий рассчитан на время экзамена с учетом удаленного доступа, временем на проверку и запасом на возможные переподключения к серверу.

  Внимание! Методика проведения экзамена подразумевает, что кандидат знает необходимые технологии и имеет практику решения подобных задач. Разбираться как решается задача - не будет времени. Конкретные API конечно не нужно помнить, они доступны в локальном файле документации.

- **Соблюдение правил работы с Bitrix Framework**

  Экзамен – это оценка навыков правильного использования технологий Bitrix Framework. К темам экзамена указаны правила, что считается правильным при решении. При несоблюдении правил – экзамен не будет успешно пройден.

  Внимание! При нарушении хотя бы одного правила выделенные знаком **(!)** – экзамен не будет засчитан, даже если все остальное верно.

- **Решение должно работать верно**

  Для зачета решения задания необходимо чтобы по факту все работало верно. Если задание решено с учетом технологий, но, например, из-за опечатки не работает верно, то решение не будет засчитано.

  Внимание! У кандидата не будет возможности после экзамена объяснить, что он опечатался или он знал, но забыл сделать что-то во время экзамена. Проверяйте, что ваше решение по факту работает верно.

Темы экзамена

1. **Общие понятия работы с Bitrix Framework**

   - Реализация MVC в платформе. Модули, компоненты, шаблоны.
   - Структура файлов ядра системы.
   - Специальные константы и переменные.
   - Страница и порядок её выполнения.
   - Использование API платформы. Получение, добавление, обновление, удаление объектов. Возможности главного модуля: классы, функции, события.
   - Обработка адресов (UrlRewrite).
   - Пользовательские поля объектов системы.
   - JS библиотека Bitrix Framework.
   - Полезные инструменты: журнал событий, монитор производительности.

Правила:

- **(!) Отсутствие «бездумного копипаста» в решениях заданий экзамена**
  - **(!) При наличии лишнего программного кода, получении и обработки данных, не требующихся для решения задачи – решение не будет засчитано.**

    Документация содержит примеры работы с API, а типовые компоненты это отличные примеры реализации той или иной задачи. В рамках условий экзамена может быть удобно использовать их как основу для решения заданий. Но нельзя просто скопировать типовой компонент или пример из документации и «чуть его поправить под себя». В ваших решениях должен быть только тот код, который участвует в решении условий задач и ничего лишнего.
  - **(!) Ваше решение заданий должно соответствовать поставленной задаче и правилам.**

    За каждое решение - отвечает его автор. Типовые компоненты, примеры в документации, примеры из видео-уроков и т.д. являются *примером решения определённой задачи,* которая хоть и может быть похожа на вашу, но не обязательно решается именно так же. В продукте и типовых решениях очень много компонентов, не все они регулярно обновляются и могут содержать устаревшие подходы.

Необходимо осмысленно применять код из документации и типовых компонентов. Ваше итоговое решение должно соответствовать требованиям экзамена и условиям вашей задачи.

- **(!) Файлы ядра системы не изменяются, не используются прямые запросы к базе данных**
- **(!) Бизнес-логика не реализуется на страницах в публичной части или шаблоне сайта.** Программный код «оборачивается» в компоненты. Это позволит использовать штатные механизмы (параметры, кеширование, разделение логики и отображения, отладку производительности, и т.д.) и упростит сопровождение проекта другими разработчиками.
- Если для решения задачи есть готовая технология, то решение задачи «в ручную» считается неверным и трактуется как отсутствие необходимых знаний.
- Переменные типа: ID инфоблока, лимит на выборку и т.д., не прописываются в коде «жестко», а определяются заранее и затем используются по коду. Если это компонент, то удобно задать в виде настройки компонента. Если это обработчик события, то определить константу. Константы определять в файле размещенном в php_interface, файл подключать в init.php.

  Это позволит просто менять значения, упрощает перенос кода.
- Для обработчиков событий, агентов, кастомных функций - создавать отдельные файлы в php_interface, и подключать их в init.php, либо выносить их в свой модуль. Это позволит проще разбираться в коде проекта, особенно когда кастомного кода становится много, упростит отладку ошибок.
- Для вывода текстовых фраз использовать языковые файлы.

2. Основы производительности

- Эффективное использование API. За вызовом API зачастую стоит работа с базой данных. База данных – это внешний ресурс. По умолчанию считаем: дополнительное обращение к внешним источникам данных дороже, чем работа внутри PHP. Необходимо минимизировать количество обращений к базе данных, и оптимизировать сами обращение.
- Инструменты платформы: монитор производительности. Оценка работы всего проекта, определение наиболее «тяжелых» страниц. Детальная оценка работы страницы сайта: время работы скриптов, количество запросов и время их исполнения.

Правила:

- **(!) Объекты одного типа получаются не в цикле, а с помощью одного вызова API. Этот подход должен применяться «по умолчанию», если API объекта поддерживает такую возможность. Существуют задачи, для которых более**

эффективным будет другой подход, но они не рассматриваются в рамках данного экзамена.

- Количество обращений к базе данных – минимизируется. Прежде чем получать дополнительные данные отдельным вызовом API, который повлечет обращение к БД, стоит убедиться, что нет возможности вычислить нужные данные из уже существующих в PHP, значительно более «легким» способом.
- Запросы к базе данных – оптимизируются. Время выполнения запроса к базе данных созданного вызовом одного и того же API может значительно отличаться и зависит от параметров вызова: количества и типа выбираемых полей и свойств, фильтра, сортировки, количества отбираемых элементов.
    - При получении объектов необходимо выбирать только те поля и свойства, которые потребуются для решения задачи, если это позволяет API. Чем меньше объем получаемых данных, тем меньше нагрузка на сервер.
    - Количество получаемых элементов необходимо задавать осмысленно, исходя из условий задач. Используются возможности API, с помощью фильтра, сразу отбирать только элементы нужные для решения задачи.
    - Если получаемых элементов потенциально много, стоит продумывать минимизацию нагрузки за счет использования постраничного вывода элементов или задания лимита отбора по какому-либо параметру.

3. Информационные блоки

- Организация хранения данных в информационных блоках.
- Поля, свойства элементов и разделов. Пользовательские поля разделов.
- Выборка из информационных блоков: получаемые поля и свойства, фильтрация, сортировка, лимиты выборки, группировка.
- Использование сложной логики в фильтре.
- Хранение данных в отдельной таблице или в общей, свойства с множественными значениями.
- Работа с различными типами свойств: строка, число, список, дата, файл, привязка к элементам, привязка к разделам, HTML/текст.
- Организация хранения данных в нескольких информационных блоках, фильтрация и выборка связанных данных.
- Реализация постраничной навигации в компонентах.
- Получение и построение адресов ссылок на детальный просмотр элементов

Правила:

- **(!) При получении данных нескольких элементов или разделов информационного блока по однотипному фильтру - данные получаются не внутри цикла с помощью GetList или GetByID, а с помощью вызова GetList один раз.**
- Минимизируется объем данных, получаемых из БД - отбираются только те элементы, которые используются для решения задачи.
- При получении элементов и разделов выбираются только те поля и свойства, которые используются для решения задачи.

4. Расширение возможностей типовых компонентов

- Использование файлов .parameters.php, result_modifier.php, component_epilog.php в шаблоне компонента для расширения логики работы типовых компонентов.
- Кэшируемая и некешируемая часть, сохранение дополнительных данных в кеш типовых компонентов.
- Использование отложенных функций в шаблоне компонента с учетом работы кеширования.

Правила:

- **(!) При кастомизации компонентов, новый функционал обязательно должен корректно работать с включённым кешированием.**
- При расширении возможностей компонентов, входящих в состав комплексных, необходимо размещать код дополнительной «тяжелой» логики (получением данных из базы данных, вычислениями) в result_modifier.php обычных компонентов, а не странице комплексного компонента. Это позволит использовать штатный механизм кеширования, снижая нагрузку на сервер.

5. События

- Использование обработчиков событий для кастомизации возможностей системы.

Правила

- Обработчик события должен выполняться только для объектов из поставленной задачи. Например, если делается обработчик для элементов инфоблока «Новостей», то под действия обработчика не должны попадать элементы других инфоблоков.
- Учитывать, что решение задачи может подразумевать обработчик не одного события, а несколько. Например «при создании» и при «изменении».
- В обработчике события доступны не все поля объекта, и не всегда событие получает одинаковый набор входных данных, они могут зависеть от точки

срабатывания события. Необходимо контролировать набор пришедших данных и при необходимости получать недостающие.

- Обработчики событий для объектов, управляемых из административной панели должны корректно работать и при редактировании элемента на «странице редактирования», и при редактировании нескольких элементов «из списка».
- Выполнить возможные проверки на основании входных данных, прежде чем выполнять код, обработчик может быть ресурсоёмким. Например, задача – обрабатывать изменений новостей, которые активны. Значит первым делом в обработчике проверяется флаг активности (убедившись, что поле ACTIVE передается в обработчик). Тем более это актуально, если в обработчике выполняется запрос данных для решения задачи, обработка «не отсеянных» заранее элементов даст лишнюю нагрузку.

6. Создание компонента и кеширование в нем

- Структура компонента: описание, параметры, исполняемый код, шаблон.
- Типовая схема работы компонента: проверка входных параметров, получение данных и реализация бизнес-логики, кеширование, подключение шаблона.
- Использование постраничной навигации для отображения списков элементов.
- Использование технологии «Эрмитаж» - редактирование данных, выводимых компонентом, в публичной части сайта.
- Реализация кеширования в компоненте:
    - типовые параметры компонента для кеширования, API для кеширования, принцип реализации.
    - параметры создания кеша, отмена кеширования.
    - определение переменных сохраняемых в кеш компонента, допустимый размер файлов кеша.
- Тегированный кеш:
    - принцип работы тегированного кеша;
    - реализация в компонентах, API.
    - добавление своего тега к кешам собственным и штатным компонентам.

Правила:

- **(!) Получаемые динамические данные, выполнение «тяжелой» бизнес-логики и шаблон компонента – кэшируются.**
- **(!) Весь функционал компонента должна работать верно и при включенном кешировании.**

- **(!) В кеш компонента сохраняются значения только тех переменных, которые будут использоваться далее в некешируемой части компонента. Это необходимо чтобы не допустить разрастание размера файлов кеша и роста ресурсов на их парсинг. По умолчанию компонент сохраняет в кеш весь arResult (поведение для обратной совместимости), разработчик обязательно должен поместит в кеш только те данные, что будет использовать в некэшируемой части.**
- Логика реализовывается в component.php, в нем нет оформления вывода информации. В шаблоне – только вывод информации, не реализовывается логика.
- Компонент содержит необходимые настройки. Например:
    - ID инфоблока, если компонент выбирает данные из инфоблока.
    - Количество элементов на странице, если используется постраничная навигация.
- Компонент отображается в визуальном редакторе в осмысленном разделе компонентов, настройки компонента доступны администратору сайта в визуальном режиме.
- В компоненте проверяются входные данные на корректность значений, при неверных данных – работа компонента завершается с показом соответствующего текста ошибки.

7. Создание комплексного компонента

- Структура комплексного компонента: описание, параметры, исполняемый код комплексного компонента, тема комплексного компонента, простые компоненты в составе сложного, шаблоны простого компонента.
- Схема работы комплексного компонента: обработка входных параметров, определение переменных из адресной строки, определение значений переменных, определение подключаемой страницы.
- Принцип обработки адресов (UrlRewrite) в платформе. Реализация «ЧПУ» и «не ЧПУ» режима работы компонента. Шаблоны путей, переменные, псевдонимы для переменных.
- Варианты размещения шаблонов комплексного компонента и шаблонов простых компонентов, входящих в состав комплексного.

Правила:

- **(!) Страница компонента, значения переменных, шаблоны ссылок для страниц - определяется с помощью типовых механизмов комплексного компонента, а не «в ручную» по $_REQUEST и другим данным.**

- Адреса станиц для ЧПУ режима и имена переменные для не ЧПУ - задаются через визуальные настройки компонента. В компоненте определены адреса страниц и переменные «по умолчанию».

8. Агенты

- Принцип работы, стандартные агенты системы, создание собственного агента.
- Реализация периодически выполняемого агента.
- Особенности создания агентов: объекты системы, которые могут быть еще не определены при запуске агента.
- Запуск агентов на cron.

Правила

- При реализации Агента учитывается, что не все объекты системы существуют на момент его запуска.

9. Почтовая система

- Технология работы с исходящими email: почтовые события и шаблоны.
- Создание собственных почтовых событий.
- API для работы с почтовой системой.

10. Отложенные функции

- Принцип работы отложенных функций, какие API является «отложенными».
- Использование готового механизма отложенных функций для отображения контента «выше» или «ниже» чем его фактическое формирование в цикле создания страницы.

Правила:

- **(!) Функционал, реализованный с помощью отложенных функций одинаково корректно работает в компонентах и без кеширования и с использованием кеша.**

11. Создание собственных AJAX обработчиков в компонентах

- Использование JS библиотеки Bitrix Framework для отправки AJAX запросов и обработки результатов.
- Реализация обработки AJAX в компоненте.
- Реализация обработки AJAX в типовом компоненте без его копирования: использование не кешируемой части при расширении возможностей типового компонента, component_epilog.php

12. Многосайтовость и многоязычность.

- Многосайтовость на одном и на разных доменах.
- Работа с данными в многосайтовой конфигурации.
- Использование многосайтовой конфигурации для реализации разных языковых версий проекта.