Справочные материалы

Об ArcH₂O

Введение

ArcH2O - это архитектурная платформа, которая объединяет архитекторов, аналитаиков, разработчиков и специалистов по

эксплуатации в едином пространстве, устраняя хаос, дублирование и разрозненность инструментов.

Контекст и вызовы

Современные ИТ-ландшафты огромны и динамичны:

- сотни микросервисов и бизнес-приложений,
- тысячи межсервисных взаимодействий и интеграций,
- десятки команд и сотни владельцев систем.

Без централизованной платформы архитектурное описание быстро устаревает и теряет ценность.
Традиционные инструменты (Visio, Draw.io, Miro, PlantUML, Archimate и др.) помогают частично, но не масштабируются и не дают единого источника истины.

Ключевые проблемы традиционного подхода:

- Трудности поддержки и переиспользования «картинок» в визуальных редакторах,
- Отсутствие интеграции с инструментами автоматизации и развёртывания,
- Отсутствие единого подхода, стандартов (типизации) и автоматизации проверок,
- Отсутствие версионности и возможности планировать и применять изменения.

Для кого полезна платформа

- Архитекторы могут формализовать архитектуру и переиспользовать модели.
- Разработчики всегда работают с актуальной архитектурой сервисов.
- DevOps и эксплуатация видят связи сервисов и систем в реальном времени.

- Специалисты по ИБ получают прозрачность для анализа доступов и рисков.
- Менеджмент имеет полную картину ИТ-ландшафта и контролирует стратегическое развитие.

Особенности

- Сквозной процесс проектирования, документирования, ревью, согласования и публикации;
- Поддержка любых методологий и архитектурных фреймворков;
- Возможности двусторонней интеграции и встроенная поддержка Marlin (IDP);
- Доступность - легкая установка On-Premise и быстрый доступ в On-Cloud;
- Низкий порог входа и продвинутые возможности;
- Единое пространство для совместной работы.

Преимущества

- Ускоряет внедрение нового функционала за счёт совместной работы в единой платформе всех участников команд;
- Повышает качество разрабатываемого ПО благодаря встроенным в пайплайн процессам проектирования и возможности контроля план-факт;
- Снижает затраты на внедрение архитектурных практик за счёт системы рекомендаций и проверок стандартов;
- Сокращает время на проектирование за счёт автоматизации и упрощения процессов;
- Экономит бюджет благодаря снижению риска многократно переделывать функционал из-за некачественного или отсутствующего этапа проектирования;
- Решает вопрос масштабируемости архитектуры за счёт декомпозиции, переиспользования компонентов и контроля целостности вне зависимости от количества описываемых систем.

Основные возможности

- Пространства и репозитории
  Группируйте проекты в пространствах. Управляйте доступом, ветками и версиями.
- Редактор кода и DSL
  Встроенный редактор с подсветкой синтаксиса. Пишите архитектуру как код — с поддержкой комментариев и форматирования.
- Diff и история изменений
  Сравнивайте версии, просматривайте Diff, отслеживайте конфликты при слиянии.
- Роли и доступ
  Настраивайте право по ролям и зонам. Контролируйте, кто что видит и редактирует — от одного пользователя до всей компании.
- Диаграммы в реальном времени
  Меняете код — диаграмма обновляется автоматически. Поддержка автолейаута, zoom, drag & drop и экспорта в изображение.
- Интеграции и расширения
  API, SSO, IDP Morlin, Markdown, Wiki, экспорт.

Взаимодействие Платформы ArcH$_2$O с другими системами

Взаимодействие платформы с KeyCloak
KeyCloak - система идентификации и управления доступом с открытым исходным кодом, которая позволяет управлять идентификацией пользователей, контролировать доступ к приложениям и данным, обеспечивая единую точку входа (SSO). На платформе ArcH$_2$O система KeyCloak используется для идентификации и управление доступом пользователей.

Взаимодействие платформы с GitLab
GitLab - веб-платформа для управления репозиториями Git, непрерывной интеграции и непрерывной поставки (CI/CD), отслеживания ошибок, планирования проектов и управления кодом. Он предоставляет возможности для хранения кода, совместной работы

над проектами, отслеживания изменений, управления задачами, автоматизации процессов сборки и развертывания приложений. ArcH₂O использует GitLab для хранения исходного кода, его сборки и развертывания.

Взаимодействие платформы с PostgreSQL
PostgreSQL - свободная объектно-реляционная система управления базами данных (СУБД). Она широко используется благодаря своей надежности, масштабируемости и отличной поддержке стандартов SQL. Таким образом, PostgreSQL используется в системе ArcH₂O в качестве СУБД, то есть для хранения и управления данными.

Взаимодействие платформы с Kubernetes
Kubernetes - это платформа с открытым исходным кодом для автоматизации развертывания, масштабирования и управления приложениями. Он обеспечивает инструменты для управления ресурсами, оркестрации контейнеров, автоматического восстановления и других операций, необходимых для работы распределенных приложений в контейнерах.

Взаимодействие платформы с Marlin
ArcH₂O интегрирована с IDP Marlin для проверки план-факт архитектуры микросервисов.

Глоссарий

Термин	Определение
Пространство	Зона, которая разграничивает доступ пользователей к данным.
Репозиторий	Версионное хранилище схемы моделей, представлений и документации.
Ветка	Отдельная последовательность изменений в репозитории, позволяющая вести доработку схемы моделей, представлений и документации

	параллельно с основной версией без влияния на неё до момента слияния
Коммит	Операция, при которой изменения в файлах сохраняются в локальном репозитории.
Публикация/мердж/слияние	Процесс объединения изменений из одной ветки в другую
DSL	Декларативный язык, который позволяет описывать модели, представления и схемы
Модели	Элементы, связи между ними и их свойства
Представления	Диаграммы, их стили и свойства
Схемы	Классы типов элементов, связей, представлений и т.д.
Markdown	Язык разметки для форматирования текста

Интерфейс платформы

Интерфейс ArcH₂O представляет собой web-приложение и открывает доступ ко всем возможностям платформы.

Ключевые разделы

- /editor/... главный интерфейс Редактора для работы с кодом, структурой, представлениями и т.д.
- /space/... списки всех пространств, участников, репозиториев и веток и управления ими

- /page/... репозитории с документацией, ссылками на которые удобно делиться к коллегами
- /view/... представления, этими ссылками на диаграммы удобно делиться к коллегами

Элементы редактора

Интерфейс редактора является главным рабочим местом пользователи с ролью editor в платформе. Он состоит из панелей, которые можно скрывать, сворачивать и менять им размер для удобства использования.

Панель Меню

Позволяет управлять отображением остальных панелей. Содержит кнопку "гамбургера" в верхней части для доступа к меню всей платформы. В нижней части есть кнопки для быстрого перехода к документации платформы и аккаунта текущего пользователя, для выхода.

На кнопках могут появляться точки, они н акцентируют внимание, что в панели есть важные данные.

Панель Проводника

Панель открывается нажатием на кнопку "Файлы" в меню. Позволяет управлять любыми файлами и директориями в текущем репозитории.

В верхней части расположена строка с кнопками доступных действий. При наведении на кнопку появляется подсказка. Часть действий доступны также через контекстное меню (клик правой кнопкой мыши по имени файла или директории).

При создании файла или директории, допустимо указывать путь перед именем, например: a/b/c/myfile

Есть возможность перетаскивания.

Цветовая индикация аналогична панели Коммита и описана ниже.

Панель Структуры

Панель открывается нажатием на кнопку "Файлы" в меню.

Содержит список элементов в текущем открытом файле.

Панель Коммита

Панель открывается нажатием на кнопку "Коммит" в меню. Позволяет управлять изменениями: видеть что изменилось в каждом файле, сбросить или сохранять их.

В верхней части расположена строка с кнопками доступных действий. При наведении на кнопку появляется подсказка. Доступно контекстное меню.

При сохранении изменений создаётся новый коммит в текущей ветке репозитория. Опционально можно указать описание коммита, которое отображается в панели "Версии" ниже.

Цветовая индикация позволяет различать текущий статус несохраненных файлов:

- зелёный - файл создан
- оранжевый - файл изменён
- желтый - файл перемещен или переименован
- красный - файл удалён
- фиолетовый - файл имеет конфликты слияния

В этой же панели включается режим слияния.

Панель Версий

Панель открывается нажатием на кнопку "Коммит" в меню. Содержит граф коммитов всех веток репозитория.

Позволяет наглядно видеть все версии и ветки, а также связи между ними. При наведении отображается информация о коммите. При клике, в панели "Коммит" выше открывается список файлов, измененных в данном комите.

Панель Редактора кода

Панель открывается нажатием на кнопку "Код" в меню. Позволяет смотреть и редактировать содержимое файлов, а также отображает diff-изменений и конфликтов слияния.

Сверху находится панель вкладок открытых файлов. Доступно контекстное меню при клике правой кнопкой на вкладку.

Панель Рендера

Панель всегда отображается справа. Позволяет отображать представления любых диаграмм.

Поддерживается масштабирование колёсиком мыши. При клике на элементы открывает панель редактора элементов справа. Двойной клик на элемент приближает его, а на пустое поле - масштабирует так, чтобы было видно всю диаграмму. Доступно контекстное меню при клике правой кнопкой мыши.

При открытии нескольких файлов диаграмм, сверху появляется панель вкладок, аналогичная вкладкам в редакторе кода.

Панель Консоли

Панель открывается нажатием на кнопку "Консоль" в меню. Позволяет видеть ошибки, предупреждения и рекомендации при компиляции DSL и рендеринге представлений.

Панель Поиска

Панель открывается нажатием на кнопку "Поиск" в меню. Позволяет искать по содержимому, имени, пути и расширению файлов текущего репозитория с использованием регулярных выражений. А также делать массовую замену найденной строки во всех файлах.

При клике на результат поиска, открывается файл и подсвечивается найденный текст. В верхней части расположена строка с кнопками доступных действий.

Панель Статуса

Панель всегда расположена внизу. Позволяет видеть текущие пространство/репозиторий/ветку/коммит и при клике даёт возможность перейти в другое пространство/репозиторий/ветку/коммит.

ArcH$_2$O DSL

ArcH$_2$O DSL - это декларативный язык, который позволяет описывать:

- модели - элементы, связи между ними и их свойства
- представления - диаграммы, их стили и свойства
- схемы - классы типов элементов, связей, представлений и т.д.
- и иные сущности платформы

Структура

Файлы ArcH$_2$O DSL имеют расширение .a2o и могут содержать:

- импорты
- классы
- экземпляры (классов)
- свойства

Hello World
Любой файл начинается с секции импортов, а затем идёт описание модели, например:

```
from @arch2o/core import { Element }

helloWorld = new Element
```

Здесь мы импортировали класс Element из пространства arch2o и репозитория core. Это системный репозиторий платформы, который содержит все базовые классы. Далее создали экземпляр helloWorld класса Element.

Базовый синтаксис

Комментарии
Поддерживаются блочные и строчные комментарии

```
/*
  Многострочный
  комментарий
*/
```

```
// Однострочный комментарий
```

Важно: в свойствах комментарии недопустимы! В следующем примере коментарий станет частью описания.

```
new Element {
  desc: Описание // это не комментарий, а продолжение описания
}
```

Переносы строк
Игнорируются любые проблелы и переносы строк (кроме свойств). Следующий код валиден

```
a
=
new
Element
"Название"
{
desc: Описание
}
```

Однако следует придерживаться канонического стиля

```
a = new Element "Название" {
  desc: Описание
}
```

Идентификаторы
Идентификаторы классов всегда начинаются с прописной буквы

```
// создаём собственный класс Foo
Foo extents Element
```

Идентификаторы экземпляров всегда начинаются со строчной буквы

```
foo = new Foo
```

Экземпляры можно определять без идентификатора (он нужен только чтобы можно было обратиться к ним)

```
new Foo
```

Для доступа к идентификаторам вложенных элементов используется точка.

```
a = new Foo {
  b = new Foo
}
c = new Foo

c -> a.b
```

Название
У классов и экземпляров можно указать название

```
foo = new Element "Название для foo"
Foo extend Element "Название для Foo"
```

Важно: название не должно содержать в себе двойных кавычек!

Тело
Классы и экземпляры могут содержать внутри себя:

- свойства
- экземпляры
- идентификаторы
- стили

Для этого используются парные фигурные скобки

```
Bar extend Foo {
  // ...
}
foo = new Foo "Опциональное название" {
```

```
  // ...
}
new Bar {
  // ...
}
```

Импорты
Можно импортировать из других файлов любые сущности, у которых есть идентификатор

```
from @testSpace/testRepo/a/b/c import {
    x.y.Z as Z,
    m, n
}
```

Здесь мы импортируем x.y.Z с именем Z, m и n из пространства testSpace, репозитория testRepo и файла /a/b/c.a2o

Можно опускать имя пространства, если репозиторий находится в этом же пространстве

```
from demoRepo/x import {
  * as q
}
```

Здесь импортируем всё содержимое файла /q.a2o под именем q.

Для вложенных сущностей вида x.y.Z и для * использование as обязательно.

Если мы импортируем из текущего репозитория, то можно использовать абсолютные или относительные пути:

```
// абсолютный путь от корня репозитория начинается со слеша /
from /x/y import a

// относительные пути относительно текущего файла: та же директория, на уровень выше и на два уровня соответственно
```

```
from ./z import b
from ../w import c
from ../../u import d
```

Здесь также видно, что фигурные скобки можно опускать, если импортируется одна сущность.

При импорте из файлов index.a2o, допускается опускать ися файла:

```
from /x/y/index.a2o import a
// эту запись можно написать короче
from /x/y import a
// важно, что здесь 'y' - это директория, а не файл

// Другой пример импорта из другого репозитория:
from @testSpace/testRepo import a
// это сокращённая запись для:
from @testSpace/testRepo/index.a2o import a
```

Свойства
Это key-value данные, которые можно указать у класса, экземпляра, а также в корне файла.

```
str: текст

Bar extend Foo {
  myBool: true
  myNum: 3.14
}

new Element {
  mlStr: `
    Многострочный
    текст
  `
}
```

Экземпляры

Экземпляры - это любые объекты указанного типа. Именно экземпляры элементов и связей отображаются на диаграммах, именно они в совокупности образуют модель метаданных.

```
from /w import { Foo, Bar }
from /a/b/c import { bar, x, a, b, c }

foo = new Foo "Опциональное название" {
  baz: свойство

  // создан внутри foo
  b = new Bar

  // идентификаторы экземпляров
  bar
  x.y.z

  a -> b

  // внутри тела можно опустить первый аргумент у связей
  -> c
  // запись выше эквивалентна: foo -> c, т.к. мы сейчас внутри foo
}

// к экземпляру b можно обратиться только через foo.
q -> foo.b
```

Есть ещё ключевое слово this, которое позволяет обратиться к корню файла из тела элемента. Его следует использовать только при конфлите имен внутри тела, например:

```
foo = new Foo "Первый"
bar = new Bar {
  foo = new Foo "Второй"

  // т.к. мы внутри bar
  bar -> foo // этот foo - "Второй"
```

```
  // используем this
  bar -> this.foo // этот foo - "Первый"
}
```

Связи

Связи являются частным случаем экземпляров в виде упрощенного синтаксиса:

```
a -> b
```

Это тоже самое что и

```
new Relation {
  source: a
  target: b
}
```

Если мы используем собственные типы связей, то синтаксис будет выглядеть так:

```
from /x import y

MyRel extend y.SomeRelation

first = a -MyRel-> b
second = a -y.SomeRelation-> b
```

или тоже самое

```
first = new MyRel {
  source: a
  target: b
}
second = new y.SomeRelation {
  source: a
  target: b
```

}

Связи можно направлять в обратную сторону

c <-MyRel- d

Как и у всех экземпляров, можем опционально давать связям идентификатор, имя и тело myRel = a -> b "Название" { desc: Описание }

Как мы видели в примере выше, для вложенных в экземпляры связей, можно для удобства опускать первый аргумент, в этом случае, вместо него будет использован текущий экземпляр, внутри которого объявлена связь.

Вместо правого аргумента можно объявить новый экземпляр

```
a -> new Foo
a <- new Foo

// важно, что в случае объявления, название и тело относятся к Bar
a -> new Bar "Название Bar" {
  desc: Описание Bar
}

// a в этом случае - к самой связи
a -> b "Название связи" {
  desc: Описание связи
}
```

Наследование
Можно создавать собственные типы через наследование любых классов. Можно наследоваться от любых классов, многократно. Наследуется всё: имя, свойства и вложенные сущности.

```
from @arch2o/core import Element

Service extend Element "Сервис" {
```

```
  new StringProperty "Владелец" {
    key: owner
  }
  new StringProperty "Язык" {
    key: language
    optional: true
  }
}
JavaService extend Service {
  language: Java
}

new Service {
  owner: Вася
}
new Service {
  owner: Петя
  language: C#
}
new JavaService {
  owner: Коля
  // language = Java будет унаследовано от JavaService
}
```

Базовые классы

Все базовые классы платформы можно импортировать из @arch2o/core:

Элементы

Абстрактный элемент модели (узел графа) Element - это основа любых ваших объектов, от пользователей и систем, до целей компании и оборудования.

Если создавать экземпляр прямо из Element, то мы получим просто экземпляр, не имеющий никакой семантики, а на стандартной диаграмме это будет просто прямоугольник

new Element

Мы рекомендуем так делать только для первичных набросков архитектуры когда ещё не выработана (не выбрана) схема.

Гораздо эффективнее создать необходимые типы элементов, например C4Model-подобная схема:

```
System extend Element
Container extend Element
Component extend Element

mySys = new System {
  myService = new Container {
    dbComponent = new Component {
      -> myDb
    }
    logic = new Component {
      -> dbComponent
      -> someSys
    }
    api = new Component {
      -> logic
    }
  }
  myDb = new Container

  myService -> myDb
}

someSys = new System
```

Связи

Абстрактная связь Relation между элементами (ребро графа).

Если создавать экземпляр прямо из Relation, то мы получим просто связь, не имеющую никакой семантики, а на стандартной диаграмме это будет просто стрелка

```
a -> b
// или тоже самое
new Relation {
  source: a
  target: b
}
// важно: во втором случае необходимо явно импортировать Relation
```

В некоторых простых случаях этого достаточно, но гораздо эффективнее создать собственные типы связей, например Archimate-подобные связи:

```
Composition extend Relation "Композиция"
Trigger extend Relation "Вызов"

myService -Composition-> someComponent {
  desc: myService включает в себя someComponent как неотъемлемую часть
}
myService -Trigger-> someProcess {
  desc: myService инициирует вызов процесса someProcess
}
// и т.д.
```

Связи всегда направленные. Если у вас двусторонняя связь, то это две связи.

Свойства

Для типизации ваших свойств классов, есть три основных класса:

- StringProperty
- NumberProperty
- BooleanProperty Они позволяют создать свойство для класса, задать ему ключ, имя, описание и обязательность

```
Application extend Element {
  new StringProperty {
```

```
    key: owner
  }
  new NumberProperty {
    key: rank
    optional: true
  }
  new BooleanProperty {
    key: cloudNative
  }

  // значение по умолчанию
  cloudNative: false
}

myApp = new Application {
  owner: Иванов Иван
}
```

Если мы хотим дать возможность указывать произвольные key-value свойства, то следует использовать DynamicStringProperty

```
Foo extend Element {
  new DynamicStringProperty
}

new Foo {
  blabla: бла бла бла
}
```

Все свойства Foo будут типа String.

При этом DynamicStringProperty не конфликтует с другими свойствами

```
Foo extend Element {
  new DynamicStringProperty

  new StringProperty {
    key: owner
```

```
  }

  new NumberProperty {
    key: rank
    optional: true
  }
}

foo = new Foo {
  // это будет строка, а не число
  someProp: 12345

  // это будет строка, а не boolean
  anotherProp: true

  // это будет число, т.к. мы явно декларировали rank как Number
  rank: 2
}
```

В коде выше будет ошибка, т.к. foo должен обязательно содержать owner.

Представления

Для отображения моделей используется класс View

```
first = new Service
second = new Service

new View {
  first
  second
}
```

Здесь мы создали стандартное представление, на котором будут отображены два элемента.

Если мы не перечислим объекты для отображения, то по умолчанию отобразятся все экземпляры текущего файла

```
new View {

}
```

или

```
new View
```

Если мы вовсе не укажем представление

```
first = new Service
second = new Service
```

то платформа сама создаст его и наполнит экземплярами из текущего файла.

Представлению, как и другим экземплярам, можно присвоить идентификатор, имя м описание

```
landscape = new View "Системный ландшафт" {
  desc: Отображает все системы компании

  // ...
}
```

Также можно указать дополнительные параметры представления. Доступны настройки layout:

- direction направление рёбер графа:
    - LR слева направо
    - TB сверху вниз
    - BT снизу вверх
    - RL справа налево
- ranker режим ранжирования:

    - network-simplex минимизирует длину рёбер
    - tight-tree старается сделать дерево компактным
    - longest-path узлы с большей глубиной размещаются выше/левее
- router тип линий связей
    - normal простые прямые линии
    - orthogonal линии с прямыми углами
    - manhattan оптимизированный orthogonal, учитывающий препятствия
    - metro как manhattan, но не прямые углы
- connector стиль соединения:
    - normal обычные прямые углы
    - rounded скругленные углы
    - smooth плавные кривые Безье
    - jumpover "мостики" в местах пересечений, только для manhattan

По умолчанию используются следующие настройки

```
new View {
  layout: {
    direction: LR
    connector: rounded
    router: manhattan
    ranker: longest-path
  }
}
```

Но все эти примеры просто отображают на диаграмме переданные экземпляры, что не очень удобно для описания больших моделей. Для решения этой проблемы, можно создавать различные типы представлений. Например, точки зрения (viewpoints) как в Archimate и любые иные.

При создании собственного представления можно перечислить типы элементов и связей, которые нужно отображать

```
LandscapeView extend View "Системный ландшафт" {
  System
  Trigger
}
```

В этом представлении будут отображены только экземпляры класса System и только связи Trigger.

Вывод динамических связей пока находится в разработке
Если мы хотим дать пользователю возможность переключать на одной странице несколько представлений (например, если описываем системы с нескольких точек зрения), То используем класс Views и помещаем в него несколько View

```
new Views {
  default: application

  landscape = new LandscapeView {
    // ...
  }
  application = new ApplicationView {
    // ...
  }
  stakeholder = new StakeholderView {
    // ...
  }

  layout: {
    router: metro
  }
}
```

Здесь видим опциональное свойство default, которое позволяет открыть по умолчанию указанное представление, иначе будет открыто первое.

Также внутри Views можно задать layout, который по умолчанию, будет применен для всех представлений в нём.

Стили

Всем элементам и связям можно задавать собственный стиль отображения с помощью класса Styles. Стили могут быть заданы внутри классов и представлений и наследуются в следующем порядке

*

```
Application extend Element {
  new Styles
}
```

Свойства

Свойства - это key-value данные, которые могут быть у классов, элементов, связей и даже в корне файла

По умолчанию все значения свойств интерпретируются как строки, однако вы можете типизировать их.

Типы

Поддерживаются следующие типы:

- однострочная строка без переносов
- многострочная строка с переносами
- число
- boolean
- список строк
- вложенные свойства

Числа

Следующие примеры числовых свойств все валидны:

ni: 42
nib: 4_2
nin: -42
nins: - 42
nip: + 42
nd: 0.42
ndnf: 4.
ndof: .42
ndn: -0.42
ndnof: -.42
ne: 1.42e5
neu: 1.42E5
neb: 1_3.4_2e5_6
nen: -1.42e5
nei: 1e5
nenf: 1.e5
nesp: 1.42e+5
nesn: 1.42e-5
nenof: -.42e5
n16: 0xFF
n16b: 0xFF_CD
n16u: 0XFF
n8: 0o777
n2: 0b1111
n2u: 0B1111
nbi: 12345678901234567890123456789012345678901234567890n
nbib: 123_456_789_012345678901234567890n

Однострочные строки
Следующие примеры строковых свойств все валидны:

s: простая строка. Допустимы любые символы, никаких ограничений, даже такие `"\#?* и т.д.
sl: Пробелы в начале и в конце будут удалены. В этой строке будут удалены 3 пробела в начале и 5 проблеов в конце
sc: текст // комментарии использовать нельзя, они являются частью текста

sq: "   для сохранения пробелов в начале и конце текста используются двойные кавычки   "
sqt: "   причём сами двойные кавычки в начале и в конце будут удалены, а пробелы останутся   "
sqs: "важно: если оставить лишь одну двойную кавычку в начале или в конце, то она не будет удалена
sqca: "текст" // после двойных кавычек комментарии также являются частью текста
sqe: "Экранировать двойные кавычки " в тексте не требуется"

Многострочные строки
Следующие примеры многострочных свойств все валидны:

sm: `doc
   Обратные кавычки позволяют писать многострочный текст.
   Один перенос позволяет удобно форматировать текст и ни на что не влияет.
   Два переноса подряд интерпретируются как реальный перенос строки.
   В данном случае атрибут sm рапарсится как 3 строки, а не 10.

   В конце и в начале каждой строки удаляются все пробелы (включая отступы),
   однако при однократном переносе в конце предыдущей строки принудительно добавится один пробел.

   Если в тексте есть обратные кавычки, их необходимо экранировать через обратный слэш. TODO не реализовано
   // Комментарии здесь не работают, т.к. становятся частью текста.
   `

smqi: `doc
   закрывающую обратную кавычку можно располагать с любым отступом или без него вовсе
`

smr: `raw
Ключевое слово raw включает режим, в котором весь текст сохраняется как есть (включая пробелы и переносы):

отступ в 4 пробела в начале и 5 пробелов в конце этой строке сохранятся

В данном случае smr распарсится как 4 строки, т.е. в точности как написано.
`
// Свойства разных типов, комментарии внутри атрибутов запрещены:
str: строка, допустимы любые символы, пробелы вокруг будут удалены
quotedStr: " строка в кавычках, которые будут удалены, но пробелы внутри сохранятся"
quotedNotEscapeStr: "кавычки " внутри текста игнорируются и не требуют экранирования"
quotedSingleStr: "непарные кавычки игнорируются
multiLineStr: `
многострочный текст.
В конце и в начале каждой строки удаляются все проблелы (включая отступы).
Один перенос ни на что не влияет и трактуется как пробел,

для переноса строки используйте два переноса подряд (как в markdown).
Разрешены любые символы, кроме обратных кавычек.
Обратные кавычки внутри текста запрещены, т.к. ещё не реализовано их экранирование.
В строках с открывающей и закрывающей кавычками запрещено писать текст.
`
rawStr: `raw
Ключевое слово raw сразу после открывающей кавычки включает режим,
в котором текст остается без изменений со всеми пробелами и переносами
Обратные кавычки также запрещены, пока не будет реализовано экранирование
`

Boolean
Следующие примеры boolean-свойств все валидны:

```
  b1: true
   b2: false
   b3:       true
```

Вложенные свойства
// todo

```
a: {
   b: {
      c: 42
   }
}
```

Markdown

Платформа ArcH₂O имеет поддержку Markdown.

Wiki-документация

Есть возможность создания wiki-документации.

Лучше всего создавать документацию в отдельных репозиториях, это позволяет размещать к неё доступ на отдельной странице вида: /page/простро/репозиторий_документации. Также такое разделение не будет мешать процессам документирования и проектирования, так как они будут в разных репозиториях. Можно создавать неограниченное количество репозиториев.

Заметки

Помимо документации, можно использовать Markdown точечно в нужных местах репозитория. Например, сопровождать описание моделей или представлений отдельными md-файлами. Достаточно просто создать md-файл в проводнике.

Markdown Cheatsheet

Headers

H1

H2

H3
H4
H5
H6
Emphasis

Emphasis, aka italics, with *asterisks* or *underscores*.

Strong emphasis, aka bold, with asterisks or underscores.

Combined emphasis with asterisks and *underscores*.

Strikethrough uses two tildes. Scratch this.

Lists

1. First ordered list item
2. Another item ⋅⋅* Unordered sub-list.
3. Actual numbers don't matter, just that it's a number ⋅⋅1. Ordered sub-list
4. And another item.

⋅⋅⋅You can have properly indented paragraphs within list items. Notice the blank line above, and the leading spaces (at least one, but we'll use three here to also align the raw Markdown).

⋅⋅⋅To have a line break without a paragraph, you will need to use two trailing spaces.⋅⋅ ⋅⋅⋅Note that this line is separate, but within the same paragraph.⋅⋅ ⋅⋅⋅(This is contrary to the typical GFM line break behaviour, where trailing spaces are not required.)

- Unordered list can use asterisks
- Or minuses
- Or pluses

Links

I'm an inline-style link

I'm an inline-style link with title

I'm a reference-style link

I'm a relative reference to a repository file

You can use numbers for reference-style link definitions

Or leave it empty and use the link text itself.

URLs and URLs in angle brackets will automatically get turned into links. http://www.example.com or http://www.example.com and sometimes example.com (but not on Github, for example).

Some text to show that the reference links can follow later.

Images

Here's our logo (hover to see the title text):

Inline-style: 

Reference-style: 

Highlighting

```
var s = "JavaScript syntax highlighting";
alert(s);
```

```
s = "Python syntax highlighting"
print s
```

```
No language indicated, so no syntax highlighting.
But let's throw in a <b>tag</b>.
```

Footnotes

Here is a simple footnote[^1].

A footnote can also have multiple lines[^2].

You can also use words, to fit your writing style more closely[^note].

[^1]: My reference. [^2]: Every new line should be prefixed with 2 spaces. This allows you to have a footnote with multiple lines. [^note]: Named footnotes will still render with numbers instead of the text but allow easier identification and linking.
This footnote also has been made with a different syntax using 4 spaces for new lines.

Blockquotes

Blockquotes are very handy in email to emulate reply text. This line is part of the same quote.
Quote break.

This is a very long line that will still be quoted properly when it wraps. Oh boy let's keep writing to make sure this is long enough to actually wrap for everyone. Oh, you can *put* Markdown into a blockquote.
Horizontal Rule

Three or more...

Hyphens

Asterisks

Underscores

Line Breaks

Here's a line for us to start with.

This line is separated from the one above by two newlines, so it will be a *separate paragraph*.

This line is also a separate paragraph, but... This line is only separated by a single newline, so it's a separate line in the *same paragraph*.

Ролевая модель

Роли пользователей

	Пользователь	Администратор
Просмотр списка пространств	✅	✅
Создание пространств	✅	✅
Управление пространствами	В зависимости от роли в пространстве	✅

Роли участников пространств

	Владелец	Редактор	Читатель
Просмотр страниц пространства	✅	✅	✅
Добавление/редактирование/удаление участников	✅	❌	❌
Редактирование пространств	✅	✅	❌
Удаление пространств	✅	❌	❌
Создание/редактирование/удаление репозиториев	✅	✅	❌

Создание веток	✅	✅	❌
Редактирование/удаление веток	✅	✅ если он является владельцем ветки	❌
Создание коммитов	✅	✅ если он является владельцем ветки	❌
Мердж	✅	✅ если он является владельцем ветки	❌

FAQ

Как получить доступ к платформе?

1. Обратиться к администратору для выдачи одной из ролей: *user* или *admin* для доступа к платформе;
2. Авторизоваться в платформе под своей учётной записью;
3. Обратиться к администратору или владельцам пространств для выдачи вам доступа ко всем или конкретным пространствам.

Как выйти из аккаунта?

1. Нажать в левом меню кнопку "Аккаунт" со значком человечка;
2. Выбрать пункт "Выйти".

Как сохранить диаграмму как картинку или ссылку?

1. Нажать правой кнопкой мыши на пустом месте диаграммы в панели "Рендера";
2. Выбрать из контекстного меню нужный пункт.

Квалификация, численность, функции персонала, оказывающего поддержку Платформе ArcH$_2$O

Роль «Разработчик»

Разработчик Платформы ArcH$_2$O обладает следующими компетенциями:

- Опыт разработки на Java/Type Script не менее 2-ух лет;
- Опыт разработки API, обеспечивающих взаимодействие по следующим протоколам: gRPC, REST, Kafka messaging;
- Опыт работы с СУБД (PostgreSQL);
- Уверенное знание принципов работы систем контроля версий (Git);
- Опыт работы с системами контейнеризации (Docker);
- Базовые знания в области автоматизации процесса разработки (CI/CD);
- Опыт использования agile-методологии разработки с использованием: code-review, merge-request, feature-branch, регулярная поставка релизов и т.п.

Роль «Аналитик»

Аналитик на Платформе ArcH$_2$O обладает следующими компетенциями:

- Опыт работы системным аналитиком от 2-ух лет;
- Опыт сбора, анализа и управления требованиями на разработку;
- Опыт работы с REST API и/или SOAP;
- Опыт работы с БД;
- Умение работать с SQL-запросами;
- Знания и опыт использования нотаций UML, BPMN и др.;
- Опыт разработки функциональных требований, технических заданий, проектной документации;
- Опыт с xml, xsd-схемами, json.

Роль «Специалист по тестированию»

Специалист по тестированию на Платформе $ArcH_2O$ обладает следующими компетенциями:

- Опыт работы от 2-ух лет;
- Понимание архитектуры клиент-серверных и веб-приложений;
- Опыт проведения функционального тестирования;
- Понимание подходов тестирования frontend, backend и умение применять их;
- Опыт тестирования API;
- Умение составлять тестовую документацию;
- Умение анализировать бизнес и функциональные требования;
- Опыт использования баг-трекеров и написания подробных и понятных баг-репортов;
- Понимание методологий Scrum и Agile;
- Опыт работы с инструментами для проведения тестирования (Postman, SoapUI, Swagger и т.п.);
- Знание REST/SOAP/JSON/XML;
- Опыт использования Git, Docker, Kubernetes.

Численность персонала платформы Octo

В процессе разработки и проектирования платформы задействовано: 3 разработчика, 1 аналитик и 1 специалист по тестированию.

Функциональные обязанности персонала платформы Octo

В функциональные обязанности сотрудника с ролью "Разработчик" входят:

- Выполнение задач, поставленных ведущим разработчиком или руководителем проекта;
- Анализ, проектирование, разработка и тестирование программного обеспечения в соответствии со стандартами, принятыми в рамках проекта;
- Планирование своей работы и ведение проектной документации;
- Обеспечение высокого качества своих разработок;
- Согласование действий с другими разработчиками при проведении совместных разработок.

В функциональные обязанности сотрудника с ролью "Аналитик" входят:

- Сбор, анализ и документирование требований (интервьюирование заказчиков);
- Взаимодействие с заказчиками и экспертами предметных областей;
- Разработка технических заданий (ТЗ) и постановка задач;
- Построение моделей, прототипов;
- Проектирование программного обеспечения и комплексных систем;
- Координация разработки, тестирования, приемки и внедрения ПО;
- Составление документации;
- Тестирование и устранение ошибок при разработке ПО.

В функциональные обязанности сотрудника с ролью "Специалист по тестированию" входят:

- Разрабатывать планы, графики, методики и описания тестирования;
- Моделировать ситуации, которые могут возникнуть в условиях эксплуатации программного обеспечения;
- Работать в связке с разработчиком;
- Создавать тест-планы, тест-кейсы;
- Выполнять тестирование программных продуктов;
- Выполнять нагрузочные тестирования;
- Анализировать результаты, полученные во время прохождения тестов;
- Классифицировать выявленные ошибки и заносит их в базу данных для текущего программного продукта;
- Контролировать процесс ликвидации выявленных ошибок разработчиком ПО;
- Составлять документацию для проведения функционального тестирования;
- Участвовать в проведении опытных эксплуатаций программных продуктов;
- Заполнять таблицы баз данных тестовыми данными.

Руководства

Для администратора

Подготовка и установка

Хост демонстрационного стенда

- минимальная конфигурация сервера - 4 CPU, 4 RAM, 40GB SSD
- установленный helm >= v3.16.0
- установленный kubectl >= v1.29.3
- установленный ingress controller
- сетевые доступы до:
    - https://repo.hd-tech.ru
    - https://public.ecr.aws

Конфигурация OIDC-провайдера
ADFS

- создать в Active Directory группы для администраторов и пользователей Arch2O
- создать новую Application Group с типом "Web browser accessing a web application"
- в поле Redirect URI добавить адрес главной страницы ArcH2O
- в разделе выбора политик доступа выбрать Permit everyone
- в настройках Web application перейти в раздел Issuance Transform Rules и создать следующие правила маппинга:

    - Claim rule template - Send LDAP Attributes as Claims

LDAP Attribute	Outgoing Claim Type
E-Mail-Addresses	email
Given-Name	given_name

Surname	family_name

○
Claim rule template - Send Group Membership as a Claim

User`s group	Outgoing claim type	Outgoing claim value
выбрать группу администраторов	roles	admin
выбрать группу пользователей	roles	user

- в разделе Client Permissions разрешить области openid, email, profile и завершить настройку
- открыть консоль PowerShell от имени администратора и выполнить конфигурацию CORS

Set-AdfsResponseHeaders -EnableCORS $true
Set-AdfsResponseHeaders -CORSTrustedOrigins <адрес главной страницы Arch2O>

Keycloak

- создать публичного клиента и заполнить поля:

Valid redirect URIs	Valid post logout redirect URIs	Web origins
адрес главной страницы Arch2O	+	+

- добавить клиенту роли admin и user
- создать группы для администраторов и пользователей Arch2O
- в настройках групп создать мапперы на соответствующие роли клиента
- создать новый Client Scope arch2o с типом Default
- перейти в настройки scope arch2o и создать маппер - Mapper type = Group Membership, Token Claim Name = roles
- добавить клиенту scope arch2o

Для работы arch2o необходимо создать Роль, БД, и добавить дополнительные права этой роли. *Команды выполняются из под root пользователя*

```
CREATE USER arch2o WITH PASSWORD '123456';
CREATE DATABASE arch2o;
GRANT ALL PRIVILEGES ON DATABASE arch2o TO arch2o;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO arch2o;
GRANT CREATE ON SCHEMA public TO arch2o;
```

Создать namespace и secret для аутентификации в частном репозитории

```
kubectl create namespace arch2o
kubectl create secret docker-registry arch2o-registry
--docker-server=<registry_name>
--docker-username=DUMMY_USERNAME
--docker-password=DUMMY_DOCKER_PASSWORD -n arch2o
```

Добавить репозиторий, скопировать содержимое values.yaml в локальный файл arch2o.yaml

helm repo add arch2o-helm https://repo.hd-tech.ru/repository/arch2o-helm --username "username" --password "password"
helm repo update
helm search repo arch2o-helm/arch2o # посмотреть список актуальных версий
helm show values arch2o-helm/arch2o > arch2o.yaml

Отредактировать arch2o.yaml:

- AuthProvider.url - адрес провайдера аутентификации
- arch2o.version - версия приложения
- arch2o.frontend.apiUrl - url адрес backend сервиса
- arch2o.frontend.ClientId - id клиента frontend
- arch2o.backend.data.user - имя пользователя базы данных приложения
- arch2o.backend.data.password - пароль пользователя базы данных приложения
- arch2o.backend.data.url - JDBC URL подключения для к базе данных приложения
- arch2o.backend.license - лицензия продукта
- arch2o.backend.rootCA - сертификат корневого УЦ
- ingress.host - домен среды развёртывания
- ingress.tls.certificate - ssl сертификат в PEM-формате
- ingress.tls.key - приватный ключ сертификата в PEM-формате

Запустить установку с конкретной версией:

helm upgrade -i arch2o -f ./arch2o.yaml --version 1.3.3 arch2o-helm/arch2o -n arch2o

Для входа в Arch2O перейти по адресу *https://example.ru* (домен указан в переменной ingress.host)

Лицензия

Общие положения
Лицензия устанавливает:

- получателя, на имя которого она выпущена;
- наименование компании получателя лицензии;
- дату и время начала действия;
- дату и время окончания действия.

Применение
Файл лицензии конфигурируется при установке/обновлении платформы ArcH$_2$O параметром license в values.yaml.

При ручном изменении лицензии в kubernetes в обход процедуры обновления следует перезапустить микросервис arch2o-backend для её применения.

Для владельца и редактора

Управление пространствами и участниками

Необходимые права:

- Создание пространства: любой пользователь;
- Добавление/редактирование/удаление участников: владелец пространства;
- Редактирование пространства: владелец/редактор;
- Удаление пространства: владелец.

Создание пространства

1. Перейдите во вкладку "Пространства" левого меню;
2. Нажмите на кнопку "Новое пространство";
3. Введите код - уникальный идентификатор, и описание;
4. Нажмите на кнопку "Создать".

Редактирование пространства

Для редактирования доступно только поле "Описание".

1. Перейдите во вкладку "Пространства" левого меню;
2. В строке пространства нажмите на кнопку "Редактировать";

3. Внесите изменения;
4. Нажмите на кнопку "Сохранить".

Удаление пространства

Удаление пространства доступно без предварительного удаления репозиториев, веток и др.

1. Перейдите во вкладку "Пространства" левого меню;
2. В строке пространства нажмите на кнопку "Удалить";
3. Подтвердите удаление.

Добавление участников в пространство

1. Перейдите во вкладку "Пространства" левого меню;
2. В строке пространства нажмите на кнопку "Участники";
3. Нажмите на кнопку "Добавить участника";
4. Введите ФИО или почту пользователя;
5. Выберите роль);
6. Нажмите на кнопку "Добавить".

После добавления пользователь получит доступ к просмотру репозиториев и файлов пространства. Права на редактирование, создание и удаление данных будут автоматически определены в соответствии с назначенной ролью.

Изменение роли участника пространства

1. Перейдите во вкладку "Пространства" левого меню;
2. В строке пространства нажмите на кнопку "Участники";
3. Нажмите на кнопку "Изменить" в строке с именем участника;
4. Выберите роль;
5. Нажмите на кнопку "Сохранить".

После изменения роли права на редактирование, создание и удаление данных будут автоматически переопределены.

Удаление участника пространства

1. Перейдите во вкладку "Пространства" левого меню;
2. В строке пространства нажмите на кнопку "Участники";

3. Нажмите на кнопку "Удалить" в строке с именем участника;
4. Подтвердите удаление.

После удаления пользователя из пространства его доступ к репозиториям и файлам этого пространства будет немедленно отозван.

Управление репозиториями

Необходимые права: Владелец пространства/Редактор

Создание репозитория

1. Перейдите во вкладку "Пространства" левого меню;
2. Нажмите на название целевого пространства;
3. Нажмите на кнопку "Создать репозиторий";
4. Введите код - уникальный идентификатор;
5. Нажмите на кнопку "Создать".

После создания репозитория ветка main будет создана автоматически.

Редактирование репозитория

1. Перейдите во вкладку "Пространства" левого меню;
2. Нажмите на название целевого пространства;
3. В строке репозитория нажмите на кнопку "Редактировать";
4. Внесите изменения;
5. Нажмите на кнопку "Сохранить".

Удаление репозитория

Удаление пространства доступно без предварительного удаления веток и др.

1. Перейдите во вкладку "Пространства" левого меню;
2. Нажмите на название целевого пространства;
3. В строке репозитория нажмите на кнопку "Удалить";
4. Подтвердите удаление.

Управление ветками

Необходимые права: Владелец пространства/Редактор
Создание ветки

1. Перейдите во вкладку "Пространства" левого меню;
2. Нажмите на название целевого пространства;
3. Нажмите на название целевого репозитория;
4. Нажмите на кнопку "Создать ветку";
5. Введите код - уникальный идентификатор, выберите родительскую ветку и коммит;
6. Нажмите на кнопку "Создать".

После создания ветки код из выбранного коммита будет загружен в ветку. Версионирование ветки начинается с 0-го коммита - выбранного при создании. Полный граф веток и их коммитов доступен для просмотра в окне "Коммит" -> "Версии"

Публикация ветки

Публикация - процесс внесения изменений последнего коммита текущей ветки в другую (обычно main).

1. Перейдите во вкладку "Пространства" левого меню;
2. Нажмите на название целевого пространства;
3. Нажмите на название целевого репозитория;
4. В строке ветки нажмите на кнопку "Опубликовать";
5. Выберите целевую ветку для публикации изменений;
6. Нажмите на кнопку "Слить".

При возникновении конфликтов портал уведомит пользователя и предложит их решить.

Слияние веток

Слияние - процесс внесения изменений целевого коммита ветки в текущую.

1. Перейдите во вкладку "Пространства" левого меню;
2. Нажмите на название целевого пространства;
3. Нажмите на название целевого репозитория;
4. В строке ветки нажмите на кнопку "Слить";
5. Выберите целевую ветку для публикации изменений;
6. Нажмите на кнопку "Слить".

При возникновении конфликтов портал уведомит пользователя и предложит их решить.

Удаление ветки

Удаление ветки доступно без предварительного удаления файлов.

1. Перейдите во вкладку "Пространства" левого меню;
2. Нажмите на название целевого пространства;
3. Нажмите на название целевого репозитория;
4. В строке ветки нажмите на кнопку "Удалить";
5. Подтвердите удаление.

Управление файлами и папками

Необходимые права: Владелец пространства/Редактор

Создание директории

1. Перейдите в целевую ветку репозитория;
2. Нажмите на кнопку "Создать директорию";
3. Введите имя директории;
4. Нажмите на кнопку "Создать".

Для создания файлов внутри директории нажмите на название директории в списке и следуйте инструкции по созданию файлов. Для перемещения файлов в/из директорию(и) нажмите на название файла, удерживайте кнопку мыши и перетащите его к названию целевой директории.

Создание файла

1. Перейдите в целевую ветку репозитория;
2. Нажмите на кнопку "Создать файл";
3. Введите имя файла и выберите формат;
4. Нажмите на кнопку "Создать".

Удаление файла

Удаление файла доступно без предварительного удаления его содержимого.

1. Перейдите в целевую ветку репозитория;

2. Нажмите на название файла в проводнике;
3. Нажмите на кнопку "Удалить";
4. Подтвердите удаление.

Удаление директории

Удаление директории доступно без предварительного удаления её содержимого.

1. Перейдите в целевую ветку репозитория;
2. Нажмите на название директории в проводнике;
3. Нажмите на кнопку "Удалить";
4. Подтвердите удаление.

Управление коммитами и версиями

Необходимые права: Владелец пространства/Редактор

Внесение изменений в файл и создание коммита

1. После внесения изменений в файл сохраните изменения, нажав CTRL+S. Диаграмма/документ будет скомпилирован и отображен справа.
2. Убедитесь, что изменения сохранены в каждом из измененных файлов.
    - Если в файле есть несохраненные изменения, то в редакторе отображен кружок рядом с названием вкладки.
3. Нажмите на "Коммит" в левой панели.
4. Введите описание коммита (commit message).
5. Нажмите на кнопку "Сохранить".

Публикация изменений

1. Перейдите в список веток репозитория.
2. Рядом с веткой для публикации (мердж) нажмите на кнопку "Опубликовать".
3. Выберите ветку для слияния.
4. Нажмите на кнопку "Слить".

При возникновении конфликтов, отобразится Diff и платформа предложит их разрешить.

Управление диаграммами

Платформа предоставляет возможность через контекстное меню

- Открыть диаграмму на весь экран
- Скопировать ссылку на текущую/последнюю версию диаграммы
- Скачать диаграмму в формате png

Положение элементов на диаграмме возможно изменить через drag&drop. Их позиции будут сохранены и отображены так же при следующем открытии диаграммы.