# Этапы (задачи) ВКР и сравнительные способы их реализации

1. **Определение потребностей пользователей**

Решение задачи идентификации динамической системы одним из методов:

* 1. Идентификация на основе моделей: использование математических моделей для описания поведения системы и определения ее параметров на основе наблюдаемых данных.
  2. Идентификация системы с использованием сигналов: анализ входных и выходных сигналов от системы для определения ее поведения и параметров.
  3. Оценка параметров: использование методов оптимизации для оценки параметров системы на основе наблюдаемых данных.
  4. Идентификация на основе машинного обучения: использование алгоритмов машинного обучения, таких как нейронные сети или деревья решений, для моделирования поведения системы и определения ее параметров.
  5. Гибридные методы: объединение нескольких методов идентификации для повышения точности и надежности процесса идентификации.
  6. Идентификация на основе экспериментального проектирования: разработка экспериментов для сбора данных, которые могут быть использованы для идентификации системы, с учетом ограничений и требований системы.
  7. Статистические методы: использование статистических методов, таких как регрессионный анализ, для определения параметров системы на основе наблюдаемых данных.

1. **Выбор архитектуры приложения**
   1. Архитектура на основе клиент-сервер: клиентская часть выполняется в браузере пользователя, а серверная часть находится на удаленном сервере
   2. Архитектура на основе микросервисов: приложение разбито на независимые сервисы, каждый из которых отвечает за конкретную функциональность
   3. Архитектура на основе сервера API: фронтенд и бэкенд разделены, и взаимодействуют через API
   4. Архитектура на основе одностраничных приложений (Single-Page Applications, SPA): все данные загружаются один раз при загрузке страницы и последующие изменения происходят на клиенте без перезагрузки страницы.
2. **Проектирование функциональности:** реализация серверной части приложения, реализация алгоритмов идентификации динамических систем.

Для бэкенда самыми популярным языками являются:

* JavaScript,
* Python,
* Ruby,
* Java,
* C#,
* Go
* PHP.

Возможности получения переходной характеристики и построение передаточных функций в преобразовании Лапласа реализованы на Python.

1. **Проектирование интерфейса пользователя**

Интерфейс веб-страницы реализуется с помощью HTML, CSS и JavaSrcipt. Перед версткой можно использовать Figma для:

* Создания каркаса: создание простых эскизов или макетов для описания макета, размещения элементов
* Прототипирования: создание интерактивных прототипов для тестирования функциональности и дизайна интерфейса.

1. **Работа над безопасностью приложения**

Для обеспечения безопасности веб-приложения, решаются следующие задачи:

* 1. Проверка входных данных: обеспечение проверки пользовательских входных данных для предотвращения вредоносных атак, таких как внедрение SQL-кода (англ. SQL injection) или межсайтовый скриптинг (англ. Cross-Site Scripting).
  2. Аутентификация и авторизация: внедрение механизмов безопасной аутентификации, таких как хэширование паролей, и внедрение управления доступом на основе ролей для ограничения доступа к конфиденциальной информации.
  3. Шифрование: шифрование конфиденциальных данных, как при передаче, так и в состоянии покоя, для защиты от несанкционированного доступа.
  4. Регулярные обновления программного обеспечения: обновление веб-приложения и его компонентов с помощью последних исправлений безопасности для устранения известных уязвимостей.
  5. Тестирование на проникновение: проведение регулярного тестирования безопасности для выявления и устранения уязвимостей в веб-приложении.
  6. Мониторинг и протоколирование: мониторинг веб-приложения на предмет необычной активности и протоколирование всех событий для обеспечения своевременного обнаружения инцидентов безопасности и реагирования на них.

1. **Проектирование и реализация механизмов администрирования и аналитики**

Примеры механизмов администрирования:

* 1. Управление пользователями: добавление, удаление и управление разрешениями пользователей в веб-приложении.
  2. Управление доступом на основе ролей: назначение пользователям определенных ролей и определение действий, которые они могут выполнять на основе этих ролей.
  3. Ведение журнала действий: запись действий, выполняемых в веб-приложении, для облегчения аудита и устранения неполадок.
  4. Отчетность: создание отчетов по различным аспектам веб-приложения, таким как статистика использования и показатели производительности.
  5. Резервное копирование и восстановление: создание резервных копий веб-приложения и его данных и управление ими для обеспечения возможности восстановления данных в случае сбоя.
  6. Мониторинг: мониторинг производительности и доступности веб-приложения и оповещение администраторов о потенциальных проблемах.
  7. Управление конфигурацией: управление параметрами конфигурации веб-приложения для обеспечения его правильной и последовательной настройки.

1. **Тестирование, отладка и оптимизация**

Тестирование – проверка соответствия приложения указанным требованиям, выявление дефектов или проблем.

Существуют различные типы тестирования, в их числе:

* Модульное тестирование: тестирование отдельных компонентов или блоков приложения для проверки их поведения и функциональности.

Инструменты

Существует несколько инструментов, которые можно использовать для модульного тестирования веб-приложений, некоторые из них:

* + JUnit: фреймворк основанный на языке Java.
  + NUnit: фреймворк для .NET приложений
  + Mocha: основан на JavaScript фреймворке тестирования для приложений Node.js.
  + **PyTest**: фреймворк для приложений Python
  + PHPUnit: фреймворк для тестирования приложений на PHP
  + CppUnit: C++ фреймворк для модульного тестирования
* Интеграционное тестирование: тестирование того, как различные компоненты приложения работают вместе.
* Функциональное тестирование: тестирование функциональных требований приложения, чтобы убедиться, что они выполнены.
* Тестирование производительности: тестирование производительности приложения в различных условиях, чтобы убедиться, что оно соответствует требованиям к производительности.
* Тестирование безопасности: тестирование безопасности приложения для выявления потенциальных уязвимостей и обеспечения его защиты от атак.

**Отладка** включает в себя поиск и исправление дефектов или проблем в приложении. Это может включать использование различных инструментов, таких как отладчики, чтобы помочь определить источник проблемы и найти решение.

**Оптимизация** включает в себя повышение производительности, масштабируемости и надежности приложения. Это может включать улучшение используемых алгоритмов, уменьшение объема обрабатываемых данных или тонкую настройку конфигурации приложения.

1. **Установка приложения на сервер и настройка работы с ним. Запуск**

Деплой (deploy) — это развертывание и запуск веб-приложения или сайта в его рабочей среде, то есть на сервере или хостинге. Разработчик загружает приложение, написанное на локальном компьютере, в специальное пространство, из которого оно доступно в Интернете для пользователей.

Состоит из шагов:

* 1. Упаковка: Код приложения упаковывается в развертываемый пакет, который можно легко перенести в производственную среду (англ. Production environment).
  2. Перенос: Пакет переносится в производственную среду либо путем загрузки через веб-интерфейс, либо путем копирования непосредственно на сервер.
  3. Конфигурирование: Производственная среда конфигурируется для запуска приложения, включая настройку всех необходимых зависимостей, переменных окружения и баз данных.
  4. Тестирование: Приложение тестируется в производственной среде, чтобы убедиться, что оно работает так, как ожидается, и что все конфигурации верны (интеграционное тестирование).
  5. Запуск: приложение запускается и становится доступным для конечных пользователей. Это может включать перенаправление трафика на новую версию, запуск веб-сервера или предоставление доступа к приложению через API.
  6. Мониторинг: Приложение контролируется на предмет производительности, надежности и безопасности, чтобы убедиться, что оно продолжает удовлетворять потребности конечных пользователей.

В некоторых случаях развертывание может также включать в себя масштабирование приложения для обработки возросшего трафика или нагрузки, либо путем добавления дополнительных серверов, либо путем внесения других изменений в инфраструктуру. Точные шаги, связанные с развертыванием веб-приложения, зависят от конкретной технологии и используемой архитектуры

Инструмент для решения вышеописанных задач: **Kubernetes** - это платформа с открытым исходным кодом для автоматизации развертывания, масштабирования и управления контейнерными приложениями. Она предоставляет способ управления и оркестровки контейнеров, которые представляют собой единицы упакованного программного обеспечения, включающие все необходимое для запуска приложения, в том числе код, библиотеки и системные инструменты.

Аналоги:

* Docker Swarm: Docker Swarm является собственной системой оркестровки контейнеров и прямым конкурентом Kubernetes.
* Apache Mesos: Apache Mesos - это менеджер кластеров с открытым исходным кодом, который абстрагирует процессор, память, хранилище и другие вычислительные ресурсы от машин.
* Hashicorp Nomad: Nomad - это простой и гибкий кластерный менеджер, который может выполнять любую рабочую нагрузку или приложение на любой инфраструктуре.
* OpenShift: OpenShift - это платформа контейнерных приложений, построенная на базе Kubernetes и предоставляющая ряд дополнительных функций и инструментов для создания, развертывания и управления приложениями.
* Rancher: Rancher - это полная платформа для управления контейнерами, от разработки до развертывания, с акцентом на простоту использования, безопасность и масштабируемость.

1. **Техническая поддержка и поддержание в рабочем состоянии**

Выполнение регулярного обслуживания и обновлений для поддержания бесперебойной работы веб-приложения и устранения любых проблем.

Некоторые из инструментов и средств, которые могут быть использованы для обслуживания и обновления:

* + Системы контроля версий: Системы контроля версий, такие как Git, позволяют легко отслеживать и управлять изменениями, вносимыми в код с течением времени.
  + Инструменты автоматизированного тестирования: Инструменты автоматизированного тестирования, такие как Selenium, JUnit и TestNG, помогают убедиться, что приложение работает правильно после любых обновлений или работ по обслуживанию.
  + Инструменты мониторинга и протоколирования: Такие инструменты, как New Relic, Datadog и Logstash, помогают отслеживать производительность и работоспособность приложения и выявлять любые проблемы, требующие решения.
  + Инструменты непрерывной интеграции/непрерывной доставки (CI/CD): Такие инструменты, как Jenkins, TravisCI и CircleCI, автоматизируют процесс создания, тестирования и развертывания кода, облегчая управление обновлениями и обслуживанием.
  + Инструменты управления конфигурацией: Такие инструменты, как Ansible, Chef и Puppet, помогают управлять и автоматизировать конфигурацию приложения, облегчая внесение изменений и обновление приложения.
  + Инструменты контейнеризации: Инструменты контейнеризации, такие как Docker, позволяют легко развертывать и управлять приложением, облегчая его обслуживание и обновление.