# ПРИВЕТСТВИЕ

Уважаемая комиссия и все присутствующие.
Хочу представить вашему вниманию дипломный проект на тему «Файловый менеджер с функцией автоматического распределения файлов»

# ВВЕДЕНИЕ

В настоящее время миллиарды пользователей каждый день создают, скачивают, перемещают со съёмных носителей огромное количество файлов. Для одного пользователя этот процесс растянут во времени и со постепенно он забывает, что для определённого типа файлов у него уже создана директория. Так на диске начинают появляться целый список временных директорий, директории с огромным количеством файлов, которые лежат вперемежку, наборы одинаковых директорий на разных разделах диска и так далее. Когда разобраться в таком нагромождении файлов становиться очень трудно, пользователь тратит час\два на то что бы вручную привести всё в порядок, обещает себе, что уж теперь то он будет сохранять всё куда нужно, и со временем всё становиться на круги своя.

Уменьшить проблему можно, если воспользоваться приложениями, позволяющими пользователю существенно сократить время, затрачиваемое на ручную очистку диска (программы-сортировщики, программы, удаляющие файлы дубликаты или временные файлы и так далее).

Полным же решением этой проблемы может стать автоматизация процесса распределения файлов, попадающих на диск компьютера пользователя. Обратите внимание на **плакат №1**.

Автоматическое распределение файлов можно организовать по следующему принципу: файлы разных типов (видео, аудио, фото и документы) попадают в папку на жёстком диске пользователя из 3 источников:

1. Скачиваются из интернета.
2. Передаются через внешние носители.
3. Создаются самим пользователем.

Файловый менеджер автоматически определяет, что отслеживаемая папка изменилась и распределяет новые файлы по правилам, которые пользователь задал заранее.

Таким образом пользователю достаточно один раз определить где должны храниться файлы определённого типа, а всю работу по их распределению приложение будет делать самостоятельно.

# ГЛАВНОЕ

При разработке данного приложения учитывались три главных требования:

1. Приложение должно быть кроссплатформенным или максимально легко переносимым.
2. Приложение не должно мешать пользователю и другим приложениям пользователя выполнять их повседневные задачи.
3. Приложение не должно быть слишком сложным в освоении.

Архитектура приложения была построена исходя из первого требования. Прошу обратить внимание на **структурную схему**.

В приложении можно выделить 10 блоков, каждый из которых отделён от остальных интерфейсами взаимодействия и может быть изменён независимо. Блоки в свою очередь можно сгруппировать в 3 модуля:

1. Модуль клиента приложения, к которому относится только блок интерфейса пользователя.
2. Модуль монитора файловой системы, к которому также относится только один блок – блок мониторинга файловой системы.
3. Модуль обработки событий файловой системы, к которому относится всё остальное.

Каждый из модулей приложения реализован в виде отдельной программы. Взаимодействие между программами происходит путём передачи сообщений заранее определённого формата.

Такой подход позволяет обеспечить выполнение первого требования. Поскольку модуль обработки событий файловой системы написан на языке Python, он безболезненно переносится на любые операционные системы, на которые можно установить интерпретатор этого языка. Монитор переписывается по необходимости, в зависимости от файловой системы и имеющихся инструментов операционной системы по отслеживанию событий в ней. Клиент в данный момент написан на Python в виде кроссплатформенного консольного приложения, но в дальнейшем можно создать родные клиенты для каждой операционной системы (Linux, Windows, MAC OS).

Порядок действий приложения от появления файла в отслеживаемой директории, до момента его распределения описан на **схеме программы**.

1. Событие считывается монитором, конвертируется в формат сообщения и пересылается модулю обработки.
2. Модуль обработки конвертирует сообщение в объектный формат.
3. Получает максимум данных о файле (имя, расширение, тип контента, размер, специфичную для данного контента информацию, к примеру название

группы и жанр музыкального файла).

4. Получает правила распределения, заданные пользователем для файлов такого типа.

5. Применяет эти правила, обрабатывая отдельно ситуации, в которых файл нужно игнорировать или удалить.

6. Сохраняет информацию о действии над файлом, для возможности дальнейшего восстановления состояния.

Архитектуру приложения с точки зрения классов можно увидеть на **диаграмме классов.** Каждый блок структурной схемы в ней представлен отдельным классом со строго заданным интерфейсом. Все остальные классы являются обслуживающими (такие как сервисы по сохранению данных или классы потоков, выполняющие задачи на заднем плане) или представляют собой сложные структуры данных (правило распределения, состоящее из действия и ограничения для файлов, результат анализа файла, объекты сообщений и команд и так далее).

На данном чертеже показан пример взаимодействия между состояниями приложения, и видно время, которое тратиться в каждом из состояний на обработку одного события (**диаграмма последовательности**). Особое внимание стоит обратить на то, что состояние, тратящее наибольшее количество времени запускается асинхронно и выполняется в рамках отдельного легковесного потока. Время взаимодействия с базой данных так же минимально, потому что для уменьшения времени считывания применяется кэширование, а запись идёт очень быстро так как это одно из главных преимуществ выбранной базы данных (Mongo DB). Таким образом, за счёт распараллеливания и сокращения времени обращения к базе данных было выполнено второе требование – приложение работает незаметно как для пользователя, так и для других приложений.

## ЗАКЛЮЧЕНИЕ (**Плакат №2**)

В итоге, использование приложения пользователя сводится к четырём шагам:

1. Устанавливаем программу.
2. Настраиваем программу под свою рабочую машину.
3. Задаём правила распределения файлов.
4. Забываем про то, что она существует, и наслаждаемся результатом.

Преимущества программы:

1. Экономия времени пользователя.
2. Быстрота портирования на другие платформы (за счёт модульности).

3. Удобство использования на нескольких ПК (за счёт переноса правил и настроек).

4. Расширяемость (поле непаханое).

Недостатки программы:

1. Много времени необходимо для первоначальной настройки.

2. На данном этапе, несмотря на высокую надёжность и отказоустойчивость, серьёзные ошибки пользователю придётся откатывать вручную.

У меня на этом всё. Спасибо за внимание!