1. Strona tytułowa
2. Streszczenie pracy – tzw. Abstract – krótki opis założeń i rezultatów pracy
3. Podziękowania
4. Spis treści
5. Rozdział pierwszy - wstęp
   - kontekst pracy - krótko o systemach plików i metadanych.
   - przedstawienie koncepcji Hybrydowego systemu plików VHFS – porównanie z systemem plików hierarchicznym.
   - przedstawienie narzędzi do gromadzenia i przeglądania baz metadanych – porównanie z proponowaną koncepcją.
   - krótkie omówienie celu pracy (implementacja jest tzw. „proof on concept", badawczy charakter pracy)
   - krótkie omówienie wykorzystywanych narzędzi
   - omówienie osiągniętych rezultatów oraz ich znaczenia praktycznego oraz teoretycznego
   - omówienie organizacja pracy z akcentem na najważniejsze elementy
6. Rozdział drugi – dokładne omówienie problemu, którym zajmuje się praca. Stanu wiedzy o problemie w odwołaniu do źródeł, ogólna charakterystyka podejścia lub rozwiązania zastosowanego w pracy.
   - dlaczego problem jest istotny?
   - dlaczego **wirtualny** system plików?
   - kompatybilność z tradycyjnym api systemu plików
     - opis api systemu plików z uwzględnieniem kontekstu stosowania w wirtualnym systemie plików
     - rozpoznanie dokładnej semantyki wszystkich teraźniejszych narzędzi do obsługi systemu plików – sama analiza semantyki api systemu plików nie wystarczy
     - współpraca z powłokami – poprawna obsługa „wild cards" w Bashu.
     - dodatkowe narzędzia zarządzania wynikiem zapytań o system plików:
       - konstrukcje:
         - $ ls /hybridFS/files/limit 10/
         - $ ls /hybridFS/files/order by modification_date desc/limit 10
         - $ ls /hybridFS/files/group by creation date/
         - $ ls /hybridFS/files/name like %test%/
         - $ ls /hybridFS/files/name *.mp3$/
         - $ ls /hybridFS/files/(/name is *.mp3$/OR/name is *.ogg$/)
   - rozszerzony interfejs dla hybrydowego systemu plikow.
     - interfejs programistyczny, który umożliwia dowolne stosowanie operatorów logicznych
     - graficzny interfejs – propozycja, bez implementacji na którą na pewno zabraknie czasu.
7. Rozdział trzeci: omówienie narzędzi, metodologii, teorii, języków, systemów, itd. zastosowanych przy realizacji pracy:
   - Narzędzia i systemy:
     - system operacyjny GNU/Linux (tylko ten ze względu na FUSE).
     - edytor tekstów
     - edytor diagramów (UML i prawdopodobnie inne)
     - język Python – zalety, wady, przyczyny wyboru
     - PyFUSE – omówienie API
       - `getattr(path)`
         * `st_ino`
         * `st_dev, st_blksize`
         * `st_mode`
         * `st_nlink`
         * `st_uid`
         * `st_gid`
         * `st_rdev`

```
        * st_size
        * st_blocks
        * st_atime
        * st_mtime
        * st_ctime
```

- `readlink(path)` – serce wirtualnego systemu plików
- `mknod(path, mode, rdev)` – prawdopodobnie nie ma potrzeby implementować
- `mkdir(path, mode)` – dodaje etykietę
- `unlink(path)` – usuwa etykietę lub zdejmuje etykietę z określonego obiektu
- `symlink(target, name) – albo semantyka identyczna z`
- `rename(old, new)`
- `link(target, name)`
- `fsinit(self)`
- `*open(path, flags)*`
- `create(path, flags, mode)`
- `read(path, length, offset, fh=None)`
- `write(path, buf, offset, fh=None)`
- `fgetattr(path, fh=None)`
- `ftruncate(path, len, fh=None)`
- `flush(path, fh=None)`
- `release(path, fh=None)`
- `fsync(path, fdatasync, fh=None)`

- PyInotify
  - klasy: WatchManager, Notifier, ThreadedNotifier, EventsCodes, ProcessEvent
  - api:

```
class PTmp(ProcessEvent):
    def process_IN_CREATE(self, event):
        print "Create: %s" %  os.path.join(event.path, event.name)

    def process_IN_DELETE(self, event):
        print "Remove: %s" %  os.path.join(event.path, event.name)
wm = WatchManager()
mask = EventsCodes.IN_DELETE | EventCodes.IN_CREATE
wdd = wm.add_watch('/tmp', mask, rec=True)
```

  - wspierane zdarzenia systemu plików:
    - `IN_ACCESS`
    - `IN_ATTRIB`
    - `IN_CLOSE_NOWRITE`
    - `IN_CLOSE_WRITE`
    - `IN_CREATE – zdarzenie obsługiwane przez Trigger`
    - `IN_DELETE`
    - `IN_DELETE_SELF`
    - `IN_DONT_FOLLOW`
    - `IN_IGNORED`
    - `IN_ISDIR`
    - `IN_MASK_ADD`
    - `IN_MODIFY`
    - `IN_MOVE_SELF`
    - `IN_MOVED_FROM`
    - `IN_MOVED_TO`
    - `IN_ONLYDIR`
    - `IN_OPEN`
    - `IN_Q_OVERFLOW`
    - `IN_UNMOUNT`
- PyMySQL – po krótce o Python DB-API
- MySQL – dlaczego MySQL, dlaczego RDBMS, wybór metody składowania danych – MyISAM lub InnoDB – rozważenie sensu użycia silnika Memory.

- Metodologia:
  - krótki opis powstawania pracy teoretycznej:
    - rozpoznanie tematu
    - projekt aplikacji (w tym projekt bazy danych)

        - wpływ implementacji na wygląd pracy (weryfikacja i ewentualna modyfikacja projektu aplikacji pod wpływem doświadczeń implementacyjnych)
    - implementacja:
        - zastosowane wzorce projektowe
        - wykorzystane paradygmaty programowania
        - testy jednostkowe
8. Rozdział czwarty: omówienie rozwiązania, algorytmów
    - algorytmy wyszukujące wpływ i przyczyny wyboru organizacji metadanych
    - algorytmy proponujące zbiory podobnych etykiet – tzw. „Query Refinements”, proponowane etykiety raczej nie powinny pochodzić z przodków, chociaż na pewno byłoby to rozwiązanie wydajniejsze.
9. Rozdział piąty: omówienie rozwiązań implementacyjnych bez zbytniego wnikania w szczegóły:
    - modele danych
        - podstawowe encje w systemie
            - plik – podstawowy obiekt, relacja zawiera:
                - unikalny identyfikator
                - pełną ścieżkę do katalogu w systemie plików gdzie znajduje się plik (rezygnujemy z traktowania katalogu jako pliku)
                - nazwę pliku
                - całą strukturę uzyskiwaną poprzez *fstat()* ale rozbitą na kolumny (na niektóre z nich założone indeksy – data utworzenia, modyfikacji itp.)
            - etykieta – obok pliku kolejny podstawowy obiekt:
                - identyfikator etykiety
                - nazwa etykiety (może być dowlnym ciągiem znaków nie zawierającym znaku '/' dla prostoty rozwiązania
            - etykieta_etykieta – tabela złączeniowa zawierająca:
                - identyfikator pierwszej etykiety
                - identyfikator drugiej etykiety
                - pierwsza etykieta jest przodkiem etykiety drugiej
            - plik – link symboliczny
                - ścieżka do pliku (linka)
                - identyfikator pliku docelowego
                - te informacje w strukturze uzyskiwane poprzez *fstat()* na linku, które nigdy nie pokrywają się z informacjami uzyskanymi poprzez *fstat()* na pliku źródłowym.
            - relacja zawierająca kod triggerów (w trakcie implementacji zweryfikuję czy takie rozwiązanie jest słuszne – w przypadku triggerów napisanych w języku skyptowym nie ma przeciwskazań, zweryfikować jak to będzie w przypadku języków kompilowanych, wydaje się, że też w porządku)
            - stałe konfiguracyjne – dostępne z poziomu: */hybridFS/__config*
    - triggery – programy wyciągające metadane z plików dzięki którym możliwe jest stworzenie rozbudowanej struktury etykiet bez duże ingerencji użytkownika, w katalogu głównym */hybridFS/files* będzie znajdował się główny trigger, który uruchamia inne triggery w zależności od rozpoznanego typu.
10. Rozdział szósty: omówienie trudności przy realizacji pracy, zalet i wad przyjętego rozwiązania, planów rozwojowych i potencjalnych zastosowań pracy.
11. Podsumowanie: krótkie omówienie wyników pracy z podaniem osiągnięć i niedostatków. Zwrócenie uwagi na rozbieżność między założeniami a wykonaną pracą.
12. Spis prac cytowanych.
13. Dodatki: słownik, dokumentacja techniczna programu, dokumentacja użytkowa.