**Báo cáo thực tập**

**Version 0.4**

**Working Draft**

**Produced for:**

**Produced by:**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 11/11/1999 | 0.1 | Initial Draft | Donald Firesmith |
| 12/13/1999 | 0.2 | Updated essential abstractions.  Started operational requirements verification. | Donald Firesmith |
| 1/9/2000 | 0.3 | Updated the auction package. Updated the correspondence package. | Donald Firesmith |
| 2/13/2000 | 0.4 | Updated the document to be consistent with the new content and format standard. | Donald Firesmith |

Table of Contents

1 Giới thiệu 5

1.1 Hiện trạng ứng dụng portal 5

Định nghĩa 5

Các chức năng 5

1.2 Các vấn đề 5

1.3 Giải pháp 6

1.4 Build Table 7

1.5 Build popup 7

1.5.1 Thông báo loading: 7

Vấn đề 7

Giải pháp 7

1.5.2 Tối ưu vị trí hiện thị popup: 7

Vấn đề 7

Giải pháp 7

1.6 Làm việc nhóm 7

2 Kết quả 8

2.1 Đạt được 8

2.2 Hạn chế 8

3 Module 2: Tổng quát hóa phân tích dữ liệu 9

Table of Figures

# Giới thiệu

* VNTT
* Dịch vụ Cloudbackup:

## Hiện trạng ứng dụng portal

Trước khi bắt đầu xây dụng dashboard thì các tính năng về tạo tài khoản (manager, admin), các tính năng về backupset, các nội dung và report được hoàn thành.

Tuy nhiên portal vẫn chưa có trang chủ. Do đó, dashboard được xây dụng như trang chủ của tất cả các loại tài khoản. Đồng thời, dashboard đóng vai trò như một bản tổng hợp để người dùng kiểm tra sự hoạt động của hệ thống cloudbackup một cách tiện lợi, nhanh chóng.

Định nghĩa

* Dashboard là bảng gồm các hàng và các cột.
  + Mỗi ô trong bảng là tổng hợp trạng thái của một tài khoản manager (admin hoặc backupset) trong một thời gian cố định (một ngày hoặc một giờ).
  + Các ô được hiển thị với màu sắc khác nhau tương ứng với trạng thái của tài khoản (backupset) đó trong đơn vị thời gian. Những trạng thái này được xét theo thứ tự ưu tiên nếu có nhiều backupjob trong cùng đơn vị thời gian.
  + Mỗi dòng trong dashboard tương ứng với một tài khoản (manager, admin) hoặc backupset.
* Dashboard được thể hiện ở 2 dạng:
  + Dạng theo ngày: tổng hợp dữ liệu theo ngày, được hiển thị thành một tháng được chọn (manager, admin, vnttadmin, vnttdirector)
  + Dạng theo giờ: tông hợp dữ liệu theo giờ, được hiển thị thành một ngày tính từ thời điểm hiện hành, dạng này được sử dụng bởi tài khoản vnttagent.

Hình 1 Dashboard dạng ngày



Hình 2 Dashboard dạng giờ

* Dashboard là một tính năng mới, chưa xuất hiện trong console. Dashboard cho phép user có cái nhìn tổng quát về trạng thái của tài khoản (manager, admin) hoặc theo dõi trạng thái của khách hàng (vnttadmin, vnttagent, vnttdirector). Đây là điều rất quan trọng đặc biệt là đối với user mới vì có thẻ theo dõi được hoạt động của tài khoản. Đồng thời, công ty VNTT cũng có thể theo dõi tình trạng sử dụng của khách hàng một cách dễ hàng để có thể hỗ trợ nhanh chóng, nâng cao chất lượng dịch vụ.

Các chức năng

* Hiển thị trạng thái tổng hợp theo ngày (giờ) của user hiện hành (áp dụng đối với manager và admin).
* Hiển thị trạng thái tổng hợp theo ngày (giờ) của toàn bộ tài khoản khách hàng (áp dụng đối với vnttadmin, vnttagent và vnttdirector).
* Hiển thị popup tổng hợp thông tin khi rê chuột vào các ô trên dashboard (dữ liệu theo ngày hoặc giờ).
* Hiển thị popup tổng hợp thông tin khi rê chuột vào các ô header trên dashboard (dữ liệu theo ngày hoặc giờ).
* Hiển thị popup thông tin tài khoản, backupset khi rê chuột vào ô tương ứng.
* Tạo link liên kết đến các trang setting, backupset, report…
* Cho phép collapse và expand để xem chi tiết.
* Cho phép thay đổi thứ tự manager của dashboard với dạng giờ.
* Cho phép chọn tháng hiển thị (dạng ngày) hoặc thay đổi ngày dạng giờ).

## Các vấn đề

* Lượng dữ liệu lớn: Số backupset trong một tài khoản manager là

Trong đó:

n là số lượng account vguard pro

m là số lượng account vguard lite (mỗi tài khoản chỉ có tối đa một backupset).

ai là số backupset của tài khoản vguard pro thứ i

Với n = 5, ai = 5 ∀i, m = 5. Ta có S = 30.

Với 200 manager thì có 30 x 200 backupset = 6000 backupset.

Nếu mỗi backupset thực hiện backup mỗi 2h thì số backupjob là 6000 x 12 = 72000 backupjob. Một con số rất lớn!

* Tốc độ obs: Thời gian trả lời của obs khi truy xuất các record là khá lâu và API của Obs chỉ hỗ trợ truy xuất backup Job trên từng tài khoản admin vào một ngày xác định. Ngoài ra các thông tin cung cấp bởi chức năng truy xuất trên không thể cung cấp đầy đủ thông tin cho các backup job mà đòi hỏi với mỗi backupjob cần có một câu truy vấn obs khác để lấy thông tin. Tất cả dẫn đến thời gian truy vấn trực tiếp trên Obs để lấy dữ liệu về backupJob là rất dài.
* Sự thay đổi: API của cloudbackup cung cấp chức năng rất hạn chế để có thể cập nhật dữ liệu backupjob. Việc cập nhật dữ liệu chỉ có thể thực hiện thông qua nhiều truy vấn nên thời gian rất chậm. Đồng thời, các backupjob có trạng thái là chưa hoàn thành phải đượ cập nhật lại. Có thể những backupjob kéo dài sẽ dẫn đến việc phải cập nhật dữ liệu lại nhiều ngày. Do đó, cần phải xác định một giới hạn thời gian để có thời tránh trường hợp backupjob không kết thúc hoặc kéo dài quá lâu.
* Tần suất xem: Các thông tin về tài khoản (manager, admin) , backupset được xem với tần số thấp và có thể được thay đổi trong một phiên nên không được lưu trữ trên web mà được truy vấn với mỗi khi có yêu cầu xem từ phí người dung.
* Vấn đề bất đồng bộ

## Giải pháp

Kiểm soát sự thay đổi:

* Bắt đầu cập nhật từ ngày đầu tiên có backupjob chưa hoàn thành.
* Nếu ngày này trước thời gian hiện tại quá 7 ngày thì sẽ bắt đầu lúc 7 ngày trước ngày hiện tại.
* Nếu không có backupjob nào chưa hoàn thành thì sẽ cập nhật từ ngày hiện tại.

Cache (Database, Web):

* Database: Lưu trữ toàn bộ dữ các record backupjob trên database, do đó giảm số lượng truy vấn trên obs.
* Web: Dữ liệu tổng hợp của các ô được cache lại trên web để giảm thời gian truy vấn, tổng hợp theo ngày (giờ) của toàn bộ các admin khi được yêu cầu.

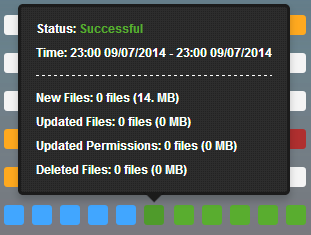
Bất đồng bộ + Detect tác vụ không có chủ đích + Loading stutus + Error

Thay đổi thứ tự manager: Dùng lệnh update trong SQL để cập nhật dữ liệu. Dựa trên các giá trị vị trí cũ và mới để cập nhật vị trí mới.

Build Tree: Dựa trên cấu trúc phân cấp từ cao đến thấp lần lượt là manager, admin, backupset. Khi đó sử dụng javascript và jquery để xử lý việc đóng mở các node trên cây. Với những nút đang mở thì ẩn tất cả các con của nó. Đối với những nút đang đóng thì hiện tất cả những con của nó.

## Build Table

## Build popup

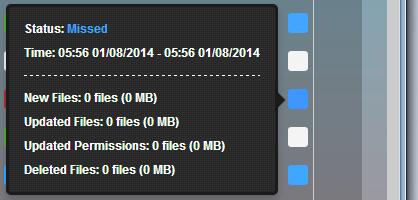
Mục đích: hiển thị thông tin về backup job.

Vấn đề: cần xây dựng một popup hiển thị đầy đủ thông tin cần thiết. Tuy nhiên cũng cần phải tối ưu về mặt truy vấn dữ liệu để cải thiện tốc độ đáp ứng người dùng từ đó cần phải giải quyết một số vấn đề.

### Thông báo loading:

Vấn đề: khi người sử dụng truy vấn nhưng hệ thống không thể đáp ứng ngay thì cần phải hiển thị thông báo đang tải cho người dùng. Tuy nhiên nếu thời gian đáp ứng nhỏ (dưới 3 giây) thì không nên hiển thị vì sẽ gây cảm giác hệ thống chậm chạp. Ngược lại, nếu đã đủ thời gian để hiện thị thì cần phải hiện thị ít nhất một khoản thời gian nhất định vì nếu không sẽ gây hiện tượng khó chịu cho người dùng.

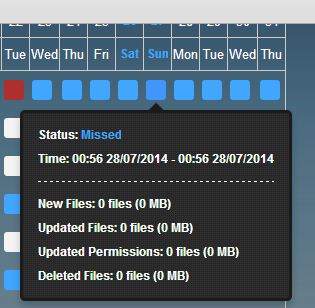
Giải pháp:



### Tối ưu vị trí hiện thị popup:

Vấn đề: khi hiển thị popup cho người dùng, nếu như vị trí hiện thị gần các biên (trên, dưới, trái, phải) thì cần phải tính toán cách hiển thị vị trí sao cho hợp lý.

Giải pháp:



## Làm việc nhóm

* Team Foudation

# Kết quả

## Đạt được

## Hạn chế

# Module 2: Tổng quát hóa phân tích dữ liệu

1. Định nghĩa

Các dữ liệu trả về ban đầu là khối lượng lớn dữ liêu thô, không có tổ chức và được lấy từ nhiều nguồn khác nhau, do đó cần một quá trình phân tích, biến đổi tổng quát các dữ liệu này thành các đối tượng để sử dụng một cách thuận tiện nhất trong các tác vụ khác liên quan đến dữ liệu.

1. Vấn đề

* Toàn bộ dữ liệu của chương trình được truy xuất thông qua các Obs server, dữ liệu gồm thông tin về users, backup set, backup job và các thông tin về hệ thống. Tuy nhiên các thông tin này được chia làm các API khác nhau và các services được viết trên các API tạo nên 32 services khác nhau. Bên cạnh số lượng services lớn mỗi service lại có các thông số đầu vào (paramaters) riêng và trả về các dữ liệu riêng. Dẫn đến việc tạo riêng các paramaters và các objects khác nhau để cho từng service sẽ tốn một chi và thời gian lớn. Ngoài ra, điều này sẽ làm quá trình kiểm tra tính đúng đắn dữ liệu khó khăn và khó kiểm soát trong các tác vụ liên quan đến dữ liệu.
* Dữ liệu các servers Obs cung cấp ở dạng dữ liệu thô XML với số lượng thuộc tính lớn, do đó việc này sẽ gây khó khăn trong quá trình kiểm tra lại dữ liệu khi truy xuất. Ngoài ra cấu trúc XML khá phức tạp gồm nhiều tập hợp và đối tượng ở các node phía trong và mỗi API lại trả cấu trúc khác do đó không thể áp dụng được các công nghệ serialization và deserialization XML của C#.
* Ngoài những khó khăn trên, trong dữ liệu trên Obs có một số dữ liệu cần được truy xuất thường xuyên, nếu không được xứ lý thích hợp chi phí và thời gian bỏ trong quá trình truy xuất lên Obs sẽ gây ra ảnh hưởng lớn tốc độ của chương trình.

1. Phân tích

* Tuy dữ khối lượng các thông số đầu vào và dữ liệu trả về của service rất lớn nhưng nó có nhiều thuộc tính chung và có một số service lấy dữ liệu cho chung một đối tượng.
* Dữ liệu XML trả về phức tạp nhưng cấu trúc nó chỉ lặp lại ở 1 số dạng cấu trúc nhất định, có thể xử lí linh hoạt để đọc dữ liệu.
* Các dữ liệu thường xuyên được sử dụng nhưng không phải lúc nào thông tin đó cũng thay đổi nên ta có thể sử dụng một số kĩ thuật để kiểm tra và truy xuất khi cần thiết.

1. Giải pháp

* Phân tích tổng quát thành các đối tượng chính để dùng chung các services có các dữ liệu trả về chung cho một đối tượng,ví dụ backup set, user profile,… . Tương tự với dữ liệu đầu ta sẽ tạo một lớp gồm các đối tượng chung và các lớp khác sẽ thừa kế lại những lớp chung này. Tuy nhiên khi tạo các lớp này số lượng các thuộc tính vẫn còn lớn để đảm bảo chất lượng không sai xót khi tạo các lớp này, có một giải pháp khá hữu ích là viết một chương trình đơn giản để tự động lọc ra các tên thuộc tính của các API do Obs cung cấp và chuyển thành định dạng thuộc tính trong C#. Việc này giảm thời gian đáng kể và chất lượng các tên thuộc tính đảm bảo.
* Sau khi phân tích các dữ liệu XML gồm ba cấu trúc chính Object, Collection,Primitive và kết hợp của chúng, do đó ta tách ra làm 3 hàm con để xử lý riêng cho các dạng dữ liệu. Các generic method được viết để tăng tính linh hoạt phù hợp với tất cả các dữ liệu trả về. Bên cạnh đó một số kĩ thuật về tạo đối tượng bằng Activator Class để giải quyết vấn đề của các Objects và Collections.
* Các dữ liệu cần thiết ví dụ như thông tin Users cần được update kiểm tra liên tục, do đó chúng ta sẽ tạo ra cache để lưu lại danh sách users và chúng ta sẽ phân tích tính toán thời gian hợp lý để update lại danh sách. Do đó danh sách này chỉ được cập nhật khi User được kiểm tra chưa nằm trong danh sách hoặc sau khoảng thời gian nhất định. Ngoài ra, chúng ta có một phương thức force update để dành cho một số trường hợp như tạo tài khoản.

Unit Test.

1. Định nghĩa.
   * Unit Test là quá trình kiểm thử các thành phần của chương trình như các phương thức, các lớp đối tượng hoặc các đơn vị khác trong code. Unit Test sẽ giúp developers có khả năng để xác định chức năng của họ hoạt động đúng như mong đợi.
2. Lợi ích của Unit Test.

* Unit Tests giúp xác định giải thuật, logic không đúng hoặc các lỗi trong quá trình lập trình từ sớm để sửa chữa và nâng cao chất lượng code.
* Unit Tests giúp có thể kiểm tra lại code một cách nhanh chóng và chính xác khi thay đổi một phần nhỏ trong chương trình.
* Unit Tests chứng minh rằng code của bạn đang hoạt động.
* Unit Tests giúp cải thiện thiết kế code mà không phá vỡ cấu trúc code.
* Test-first buộc developers phải có kế hoạch trước khi lập trình.
* Chi phí và thời gian hiện thực phần mềm giảm, bên cạnh đó chất lượng code được nâng cao và giảm rủi ro của từng chức năng một trong phần mềm.

1. Giới thiệu một số Framework Unit tests trên các ngôn ngữ khác nhau.
2. C#
   * + Visual studio: Visual Studio Unit Testing Framework lần đầu tiên được bao gồm trong Visual Studio Team System 2005, nơi nó được tích hợp với IDE, nhưng không có sẵn trong hầu hết được sử dụng Standard Edition. Từ Visual Studio 2008 nó có sẵn cũng Professional Edition. Bắt đầu với Visual Studio Express 2013, nó được bao gồm các phiên bản với Visual Studio Express.
3. Java
   * + JUnit là một framework đơn giản để viết bài kiểm tra lặp lại. Nó là một hiện thực của kiến ​​trúc xUnit cho các khuôn khổ kiểm tra đơn vị.
4. C/C++
   * + Parasoft C/C++ test: Mô phỏng con đường thực hiện ứng dụng khả thi trong nhiều chức năng và dữ liệu là rất quan trọng cho các ứng dụng nhúng trong đó phân tích chi tiết cho thời gian chạy là không hiệu quả hoặc không thể. C / C ++ kiểm tra cũng cung cấp một đường dẫn đầy đủ cho mỗi khiếm khuyết tiềm năng trong IDE và qua các liên kết nó với code, cho phép người dùng nhanh chóng chuyển đến bất kỳ điểm nào trong đường dẫn phân tích nhấn mạnh. Các lỗi được phát hiện bao gồm sử dụng chưa được khởi tạo hoặc không hợp lệ bộ nhớ, null dereferencing con trỏ, mảng và tràn bộ đệm, chia cho không, bộ nhớ và tài nguyên bị rò rỉ, và các lỗi khác nhau làm code không hoạt động.
5. JavaScript
   * + QUnit: là một framework Javascript unit testing mạnh mẽ và dễ dàng sử dụng Nó được sử dụng bởi các dự án jQuery, jQuery UI và jQuery Mobile và có khả năng kiểm tra bất kỳ mã JavaScript.
6. Unit tests trong portal.

Để đạt độ chính xác cao, tiết kiệm chi phí, thời gian và nâng cao tính chuyên nghiệp trước mỗi đoạn code được viết đều được thiết kế và hiện thực các test cases tự động trước. Các dữ liệu trong database được sử dụng kĩ thuật mock để giả lập cho các unit tests để đảm bảo hạn chế lỗi tối đa khi chạy trong môi trường thực tế. Bên cạnh đó, số lượng test cases nhiều đảm bảo các chức năng chạy đúng như mong muốn do đó số lượng bug được tìm ra và sửa chữa sớm. Trong quá trình sửa chữa, các test cases luôn được chạy lại để đảm bảo cấu trúc của chương trình không bị phá vỡ.

1. Công việc thực hiện.

Bước 1: Tạo một kế hoạch thử nghiệm   
Bước 2: Tạo các test cases và test data   
Bước 3: Nếu có thể tạo ra các scripts để chạy các test cases.   
Bước 4: Chạy test cases cho mỗi khi hoàn thành code.   
Bước 5: Sửa chữa các lỗi nếu có và chạy lại test cases.   
Bước 6: Lặp lại chu kỳ test cho đến khi "đơn vị" không còn lỗi.

# 