READY-TO-GO WEB SALES & SMALL BUSINESSES KIT

COMP2013/2014 Group 10

"A low power, always-on, thin web services solution"

Jing Li, Horace Li, Eeren Tan, Weiwei Liang, Tim Szeto

Contents

Analysis

Overview

The project task was to create a PoC for a complete computing solution (client and server) for businesses, targeting small companies that would not normally use much technology. Our product should empower them to be more effective, and assist them in their dealings rather than be an administrative burden. It should be easy to set up, use, and require minimal software maintenance.

The briefing requirements are as follows:

a) Enable users to store their deployed company work on the cloud with branding.
b) Presents opportunities for hosting a website and taking payments, e.g. via Paypal.
c) A HTML5 apps panel that provides business services, such as product inventory, buyers listings, P & L sheets, tax filings and due reminders.
d) A suite of office and project management tools installed including tax filing and accounting.
e) Dynamic generation of webpages from sales models. Marketing and branding opportunities e.g. on products that are selling well or not well, to enable Search Engine Optimisation of special deals on auto-generated WordPress webpages.

Requirements

The first step was naturally to identify the hardware and software requirements. The server requirements would depend on the applications we decide to implement, so that will be explored after a list of packages/software has been decided on.

Raspberry Pi

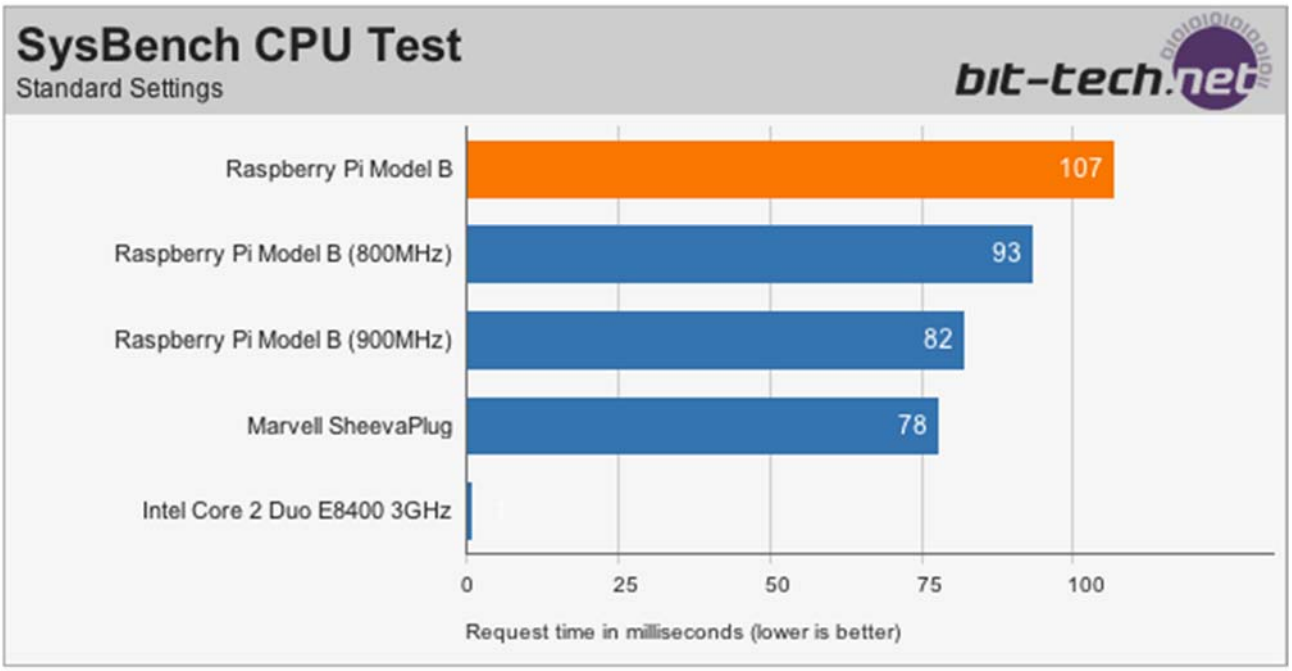
In our attempt to develop a low power always-on, thin web services solution for small businesses, which traditionally do not use much technology, we considered Raspberry Pi as the target hardware. Raspberry Pi is a low-cost credit-card-sized single-board computer developed by Raspberry Pi Foundation. The technical specifications of our target hardware, Raspberry Pi Model B are as follow:

Connectivity

According to the benchmark tests done by bit-tech.net, the connectivity of Raspberry Pi is beyond expectations and it is fast enough. Having installed the miniDLNA media streaming software, the testers were able to turn Pi into a NAS(Network Attached Storage) and stream 1080p HD content to a Playstation 3 without glitch or hiccough. However, the nature of the system-on-chip does introduce a potential network performance bottleneck. The Raspberry Pi does not come with wireless capabilities but it is compatible with USB-connected wifi dongles.

Compute Performance

To test the compute performance of Raspberry Pi, a Pi running Debian Linux OS and having SysBench benchmark suite installed is used. The SysBench CPU test runs tests such as calculating prime numbers up to a selected maximum, using 64-bit integer mathematics. That immediately puts Pi on a back-footing since it has a 32-bit processor on board. The benchmark test is ran with few other devices for comparison purpose. In conclusion, the scores proved that the Broadcom BCM2835 is a remarkably capable chip.



Graphics Performance

Its graphical capabilities such as fullscreen 1080p30 playback and hardware accelerated 3D graphics have been proven possible but they are not programmed to use due to the immature development of operating systems developed for the Raspberry Pi. Besides, to get the best performance of 3D acceleration, a bigger proportion of the RAM should be used for the acceleration. The Raspberry Pi Foundation recommends a 50:50 split between the graphics acceleration and hosting operating system.

General Purpose Performance

To test whether the Raspberry Pi could be used as a day-to-day machine, we installed the Debian Linux distribution with the lightweight LXDE desktop environment. It takes approximately thirteen seconds of text-based terminal boot time and eight seconds to load up desktop environment. Loading the Midori web browser takes around six seconds. Scrolling on complex pages or running complex programs will push Raspberry Pi to its limits. Generally, Raspberry Pi can be used as a general purpose machine but would not be able to run smoothly when it has to deal with heavy processing.

Overclocking

The config.txt located on the boot partition of SD card, represents a traditional BIOS. The .txt file can be modified to overclock and overvolt the memory, CPU and GPU. When Raspberry Pi is overclocked to 800Mhz, and the RAM is clocked to 420Mhz instead of 400MHz, the benchmark test reported a drop from 106.72ms per request to 93.07ms. 800MHz is reportedly the hard limit for Raspberry Pi. Any higher clocking than 800MHz is dangerous and could potentially brick the Raspberry Pi.

Ultimately, Raspberry Pi is a suitable candidate, because it is cheap, portable, capable to run the Arch Linux OS and access web-based services using browser.

Client

The project briefing noted that our solution should consist of a web thin client. This will reduce the likelihood of problems occurring clientside, keep hardware costs down, and allow scalability of

numbers with minimal administrative costs. The client computer will not require any heavy processing power (e.g. graphics rendering) because we are building the solution for a small (non-tech-oriented) business, so the thin client model can be adopted pretty much in full. For the purposes of this project, the downsides of a web operating system (e.g. single point of failure) will not be considered since that requirement was part of the original briefing.

Our target hardware is the Raspberry Pi since it is one of the lowest specced and priced computers available yet is capable of running a fully-fledged operating system. With a clock speed of 700MHz (ARMv6), 512MB of RAM, a GPU, and external storage support, it's hardware specifications are comparable to that of modern first-gen smartphones (utilizing the ARMv6 architecture), despite being significantly cheaper. It also has capabilities that one would find on a contemporary low-end PC, including ethernet, USB, HDMI, and 3.5mm sockets.

It is important to note at this point, however, that our project will not be exclusively limited to successful operation of the solution on the Raspberry Pi. We are also targeting devices of a similar specification (clock speeds, ports, etc) in other architectures, notably the x86 family. This will allow usage of our solution software on most non-ARM-based computers on the market today, such as a low end desktop that one would typically find in an office or at home. Indeed, while we are ultimately aiming for a production solution on the RPi, because the most part of the initial testing will be done in a virtual machine on our team members' machines, the initial development architecture will likely be x86 (x86-64).

Platform

We started off from the outset with Windows, Mac OS, and Linux since they had the largest ecosystems for development and usage/troubleshooting. We also considered Chrome OS since its operating model is broadly similar to what we're trying to achieve.

From then on, we tried out a few of the most popular (and active) distributions to use for the code base for our project, attempting to run them on a local virtual machine to examine its ease of use and how closely it resembles our ideal solution. A relatively popular distribution would guarantee a sizable amount of online resources to work from, and allow more compatible addons in the future. However, it may not necessarily have the most appropriate software, and there's the risk of a bloated operating system slowing down the client.

Subsequently, a few operating systems were eliminated due to lack of hardware support or maturity. This included Chromium OS, which fails to run on a 'typical' x86 computer (without special driver support), and Android (AOSP), whose x86 ports are still experimental. Likewise, we ruled out MacOS since it doesn't support ARM-based processors.

Although Windows RT supports ARM, its hardware requirements exceed what is offered by Raspberry Pi. There is an older alternative, Windows Embedded Compact 7, which runs on x86 and ARM processors. However, a number of points were clearly not in its favour:

- It is closed source, so if there's anything that Microsoft can't provide, it will not be easy to change.

- Unlike linux which has a large developer community, Windows CE is more oriented towards commercial entities that require an OS for their hardware product (e.g. ATMs, flight status trackers).
- Windows CE require licences to be bought, which causes a plethora of licensing, legal, and administrative issues.
- Software which has been ported onto linux/ARM greatly outnumbers that which has been ported to WindowsCE/ARM.
- There's no pre-existing stable port for Windows CE 7 on the Raspberry Pi.

Moving on to review linux, the obvious distro was Ubuntu. However, (quoting from the official Raspberry Pi FAQ) "because of issues with newer releases of Ubuntu and the ARM processor [the Raspberry Pi's] are using, Ubuntu can't commit to support Raspberry Pi at the moment". This also rules out other distros that are heavily based on Ubuntu, notably Jolicloud and Linux Mint.

With the elephant out of the room, we figured there were three main ways we could build our own linux distributions: stripping down larger mainstream distros (e.g. OpenSUSE, Debian, Fedora), building up from lightweight distros (e.g. Slackware, Arch Linux), or taking 'in-the-middle' distros such as Damn Small Linux. After evaluations of the various distros, we concluded that we would try to build from the ground up, and would therefore be working from Arch Linux because of its pre-existing compatibility for both Raspberry Pi (ARM) and x86. There are multiple reasons for taking this approach:

- It's very time consuming researching individual packages, checking their dependencies and capabilities, then stripping them out without breaking the rest of the distro.
- It's time/CPU-cycle consuming for the computer to work with bloat that could be avoided.
- There will be a tendency for us to use preinstalled software instead of looking for better alternatives elsewhere.
- It will ensure that we only install the bare minimum software required (and only the relevant dependencies), allowing the system to run as fast as it potentially can.

Listing

Office Tools

1. To generate legal copies, letterhead, letters, memos, and reference documents
2. Enable business to calculate, sort and analyze data. Track employees, production and finances. Tabulate and display data in a graphical way
3. Create business presentation for proposals, business plans etc.

Enterprise Resource Planning

Accounting

1. Financial accounting tools to record general ledger, fixed asset, payables, receivables, cash management.
2. Management accounting tools to record budgeting, costing, cost management, activity based costing

Human Resources

1. Manage employees' accounts and details

2. Manage recruitment and trainings
3. Manage payrolls
4. Handle benefits, retirement and separation

Manufacturing

1. Record and handle manufacturing orders
2. Handle order planning and scheduling
3. Record products and bill of materials
4. Provide support for manufacturing process, projects and flow
5. Product life-cycle management

Supply Chain Management

1. Support for supply chain planning and scheduling
2. Manage purchasing and inventory
3. Manage claim processing

Project Management

1. Support project planning and resource planning
2. Create and form groups of members working on projects
3. Record project costing, bill, time and expenses
4. Tasks management
5. Collaboration support over files

Sales & Payment

1. Sales management system to record sales orders, invoices etc.
2. Point of sale system that logs the user and connects to the inventory database

Customer Relationship Management

1. Sales force management system to record all stages of sales
2. Contact management system
3. Support sales forecasting and order management
4. Marketing information system for marketing decision making
5. Customer service supports

Web Hosting for Branding

1. A friendly tool that allows user to create and design website with ease
2. Provide hosting service
3. Create a business page for the company to provide information to potential customers, partners and suppliers

Browser

1. To access online desktop
2. To allows the usage of web-app
3. Safe browsing and sandboxing feature to protect the business against malicious attacks
4. Ability to view pdf files

Enterprise Social

1. Build a network of useful contacts

2. Provides easy access to potential recruitment candidates

Conference Software

1. Ability to carry out video conference with business partners
2. Manage online contacts of business partners, employees and others
3. Messaging system

Cloud storage

1. To allow employees to store their work on the cloud and so they could be accessed anywhere when needed
2. Sharing of files between employees over internet

Research

Interviews

Since our system is to be designed for small businesses, naturally, we tried to ask the UCL Entrepreneurs Society and UCL Advances for their advice on software that would be applicable for our target clients. Although the former failed to give enough help to be of much use within our project, we managed to arrange an interview with Dr Alastair Moore from UCL Advances.

Dr Moore is a project leader within UCL Advances, focusing on digital enterprise startups, as well as an active member of the British Computer Society Entrepreneurs Specialist Group. As such, he was well-informed to answer our questions in this area, with both a background in technology and a rich experience with businesses.

We were provided with a list of software, software houses, and resources which provided further lists of applicable software relating to finance, stock, and sale (POS) management, bookkeeping, and scheduling software. These eventually formed the bulk of our starting selection of software packages for each respective area of business. However, a few points he made were particularly notable:

- All businesses have to have a web presence to make an impact.
- There is a surprising amount of 'specialized' work that are actually done just on spreadsheets, even when dedicated software exist for such work.
- Although there are a few generic software requirements such as office suites and bookkeeping requirements, different (types of) businesses will, by their nature, have greatly varying software requirements, and it will be impossible to satisfy them all.
- There is an emerging trend to allow employees to work on the move through mobile apps or mobile-optimized websites.

Web desktop

A web desktop is a desktop environment which can be opened from a web browser. Web desktops provide an environment similar to that of Windows, Mac, or a graphical user interface on Unix and Linux systems. In a web desktop, the applications, data, files, configuration, settings, and access privileges reside remotely over the network. Much of the computing takes place remotely. The browser is primarily used for display and input purposes.

Particularly, our team has researched into two web desktops: Lucid desktop and qWikiOffice Web Desktop.

Lucid desktop

Lucid is a free, open source web desktop. It provides user a portable, online workspace that user can use to store files, play media, and manage office documents.

Below is the process we set up Lucid desktop:

Step1: Download wampserver2.2e-php5.4.3-httpd2.2.22-mysql5.5.24-x64 from http://www.wampserver.com/en/.

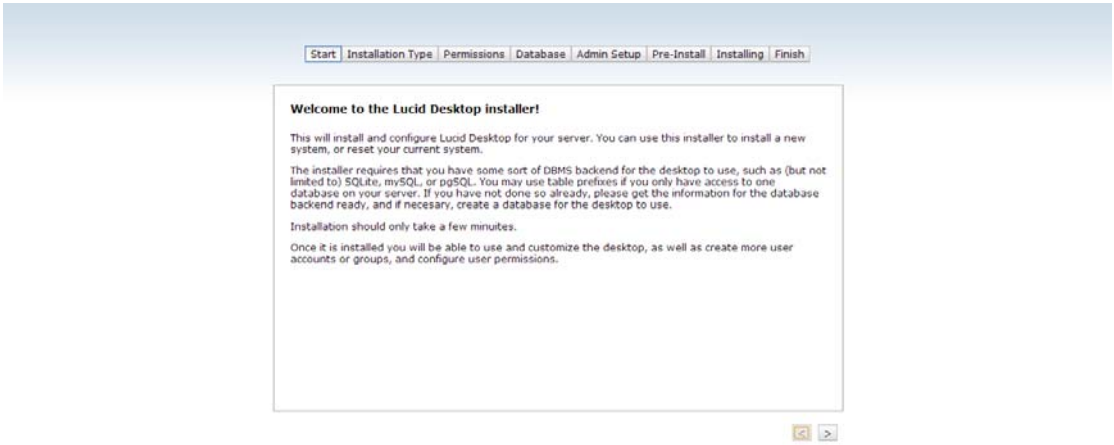
Step 2: Download Lucid-1.0.1.zip from http://www.lucid-desktop.org/download/.

Step 3: Install WampServer.

Step 4: Open phpMyAdmin, create a new database named 'lucid' in WampServer.

Step 5: Unzip Lucid-1.0.1.zip to WampServer server root directory (C:\wamp\www).

Step 6: Open browser, and type http://localhost/lucid-1.0.1/install/, then the installation begins.



Step 7: After finishing the installation, create an account to login to Lucid desktop.

Step 8:

Lucid desktop setup completed.

Advantage

Lucid is extremely customizable. Lucid has everything needed to write powerful applications with ease. Applications can be written completely in Javascript using the Dojo Toolkit, and can even have custom widgets. Also, any files on Lucid can be accessed from mobile phone, using the mobile file browser.

Disadvantage

Hosting company need to take responsibility for the security problem.

qWikiOffice Web Desktop

qWikiOffice Desktop is an open source web based desktop. It combines Ajax technology with Ext-JS Library to offer an environment which is similar to the most popular operating system, Microsoft Windows.

Advantage

Because it has a similar interface to Microsoft Windows, users will feel familiar when work on it. As it is a web-based system, it does not require much of processing power, storage or other hardware which is suitable for raspberry pi.

Disadvantage

As stated on its official website, 'It has not been released as stable or documented. It is currently available for download for developers who wish to investigate, nothing more.'

Conclusion of research on web desktop

We have considered using one of the popular web desktops as our base system and develop applications from there. However, the web desktop is overkill for our thin web services solution. In order to keep the design simple and focused, we decided to create our own web platform. Inspired by the web desktop interface, our web platform is designed to have a similar desktop-look app panel.

Besides, our local client, the archlinux desktop is way more capable than web desktop. If user do need desktop functionalities, we can always implement them on the client desktop.

JavaScript framework

Jquery

jQuery is an open source multi-browser JavaScript library. jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. It has good documentation, regular and well maintained update and visual online sample. jQuery library allows the creation of powerful dynamic web pages and web applications.

Dojo toolkit

Comparing with the most popular JavaScript framework jQuery, Dojo is a more comprehensive toolkit that offers those same features, but also includes many other pieces that facilitate building rich web applications, such as patterns for code organization, tools for abstracting data syncing between server and browser. However, Dojo seems difficult to learn and get started with. Besides, Dojo has been criticized for its “incomplete, scattered, and outdated documentation”. Hence, our team chooses to use jQuery instead of Dojo toolkit.

Ext JS

Ext JS is another JavaScript application framework developed by Sencha Inc. It offers a wide range of user interface widgets; high performance scalable grids, trees, menus, and more. It has comprehensive, well maintained learning resources and API documentation. Ext JS also has good cross platform browser compatibility.

Comparing with jQuery, Ext JS is a professionally developed by a for-profit company. It offers extensive testing and a higher-order, rigorous development of the controls. Since we are developing a thin web solution for small start-up business, jQuery suits better for us.

Our team decided to use jQuery as our JavaScript framework given its light-weight, rich function, free open-source and easy-to-use features.

Web development tools

Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level—including adding support for source-control systems (like Subversion and Visual SourceSafe) and adding new

toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).

Visual Studio supports different programming languages by means of language services, which allow the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C/C++ (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), and F# (as of Visual Studio 2010). Support for other languages such as M, Python, and Ruby among others is available via language services installed separately. It also supports XML/XSLT, HTML/XHTML, JavaScript and CSS. Individual language-specific versions of Visual Studio also exist which provide more limited language services to the user: Microsoft Visual Basic, Visual J#, Visual C#, and Visual C++.

In our project, we choose to use Microsoft Visual Studio Ultimate 2012. Along with numerous new features and improvements, this release adds support for our most advanced platforms. With built-in support for Windows 8 and Windows Azure user can easily develop across devices, services, and the cloud. It supports .NET 4.5 framework, HTML5 and CSS3.

Supported Operating Systems

- Windows 7 SP1 (x86 and x64)
- Windows 8 (x86 and x64)
- Windows Server 2008 R2 SP1 (x64)
- Windows Server 2012 (x64)

Supported Architectures

- 32-bit (x86)
- 64-bit (x64)

Hardware Requirements

- 1.6 GHz or faster processor
- 1 GB of RAM (1.5 GB if running on a virtual machine)
- 10 GB of available hard disk space
- 600 MB of available hard disk space (language pack)
- 5400 RPM hard drive
- DirectX 9-capable video card running at 1024 x 768 or higher display resolution

Ajax Control Toolkit

The Ajax Control Toolkit contains a rich set of controls that developer can use to build highly responsive and interactive Ajax-enabled Web applications. The Ajax Control Toolkit contains more

than 40 controls, including the AutoComplete, CollapsiblePanel, ColorPicker, MaskedEdit, Calendar, Accordion, HTML Editor Extender, and Watermark controls. Using the Ajax Control Toolkit, developer can build Ajax-enabled ASP.NET Web Forms applications by dragging-and-dropping Toolkit controls from the Visual Studio Toolbox onto an ASP.NET Web Forms page.

CSS framework

Bootstrap

Twitter's Bootstrap is an excellent set of carefully crafted user interface elements, layouts, and JavaScript tools, freely available to use in web design project.

Our project chooses to use Bootstrap as our base CSS framework along with our custom designed CSS for each webpage.

Database management tools

Microsoft SQL Server 2012 Express

The Microsoft SQL Server Express is a powerful and reliable free data management system that delivers a rich and reliable data store for lightweight Web Sites and desktop applications. Designed for easy deployment and rapid prototyping, this download includes support for Sysprep, Microsoft's System Preparation Utility for Microsoft Windows operating system deployment.

SQL Server Management Studio

This tool is used to manage SQL Server instances, including LocalDB, SQL Express, SQL Azure, etc.

System requirements

Supported operating systems: Windows 7, Windows Server 2008 R2, Windows Server 2008 Service Pack 2, Windows Vista Service Pack 2

- 32-bit systems
  - Computer with Intel or compatible 1GHz or faster processor (2 GHz or faster is recommended.)
- 64-bit systems
  - 1.4 GHz or faster processor
- Minimum of 512 MB of RAM (2 GB or more is recommended.)
- 2.2 GB of available hard disk space

Development

Our workload was divided between the setting up of the servers, the creation of an automated Arch Linux installation script which set up the client-side applications and packages, and evaluating potential software applications for the business to use.

Client

Having evaluated various operating systems, Jolicloud and Chrome OS most closely resembled what we were trying to achieve, a web-heavy thin client operating system. Throughout the development process, they were effectively what we were trying to emulate.

We had little success getting Chrome OS running in our virtual machines, and as a result could not examine its operation fully. Furthermore, it is far too different from conventional linux builds and many aspects of it either depended on having a reliable Googlenet behind it or exhibited behaviours that could not be reproduced an a bare-bones linux system without heavy integration with Google services (such as offline document editing, synchronization).

With Jolicloud, since it was based on a 'conventional' linux distro, we knew that most areas of it could be reproduced in a similar fashion with relative ease. The source code for each component of Jolicloud was available on github, but it was of limited use due to the lack of documentation. Instead, we worked on emulating its behaviour as perceived by the end user rather than copying sections of its code base.

We tried to ensure that all packages we used were available from the official Arch repositories. This would allow us to port our installation to the Raspberry Pi on Arch Linux ARM more easily since the majority of packages on the x86 repositories also have an equivalent ARM build on the Arch ARM repository.

Official ISO

The official ISO includes support for x86 and x64 architectures, accessed through two different images from the boot menu. Their operation is identical, except for the different architectures of the packages used. The vast majority of packages available in the official repository are either multi-architecture compatible or have two versions for each architecture. During development, we used the x64 architecture since the setup is marginally more complicated than that for an x86 environment.

By default, booting either image on the live CD provides a command line interface (CLI). The images on the CD don't provide a graphical user interface (GUI) to keep the image size small, so the initial part of the installation must be done through a CLI, or for certain programs, a text user interface (TUI) via a curses-wrapped version of the command line version.

Arch linux formerly came with an installation wizard, known as the Arch Installation Framework (AIF). However, this was recently removed from the official ISO release due to lack of maintenance.

Partitioning

Since our solution would have a thin client, we set the minimum storage size to be 4GB. This is the 'standard' capacity of SD cards for the Raspberry Pi, and was a good compromise between size (more than adequate for an installation of Arch Linux complete with drivers and a basic GUI framework) and cost (cheaper = better). Given that the majority of our applications would be cloud-based, the RAM requirements should be minimal, and thus a swap partition was not considered a necessity.

It is generally considered good practice to mount / (root) and /home on separate partitions to separate user files from system files. In the case of our client, we decided against it since the end-user should have minimal direct interaction with the file system (and hence the primary argument for separating the partitions wouldn't stand) and using a single partition would allow us to utilise the space more effectively.

For our test virtual machine, we set the hard disk capacity to 4GB and created a single partition formatted as ext4. Ext4 was chosen because it is known to be stable and is recommended over the ext3 system. There are alternatives available, but they didn't have any particular advantages over ext4

for general usage, and ext4 arguably has the most widespread usage (and therefore support), as well as compatibility and development activity.

To future-proof the client software for future, we decided on using GPT partitioning as opposed to the legacy MBR. Although the much-touted benefits of GPT (multiple and larger partitions) don't' apply in our case, it is known to be more resilient and has been recommended as the successor of MBR.

Despite that, many computers today still use BIOS (instead of the newer UEFI) to boot, and a protective MBR would still be required to boot GPT partitions on older computers. Luckily, the partitioning software we used (gdisk) includes this automatically as part of the standard partitioning process.

Bootloader

The three recommended bootloaders for Arch was GRUB2, LILO, and Syslinux. Grub is often recommended because of its customizability and maturity, but since we were using GPT partitions, there were complications involved, requiring a separate boot partition.

We eventually settled on Syslinux, because it supported GPT partitions without further fiddling and included a script that automatically installed and marked the main partition as active. There were minor hiccups during testing, as the GPT support required the installation of gptfdisk, and on various occasions, this package was left out and thus the bootloader could not be configured (on our GPT partitions). In these circumstances, rebooting into the live CD, installing gptfdisk, and reinstalling the bootloader, easily remedied the error.

Locales

The character set was set to UTF-8 to enable maximum compatibility with other scripts/languages. Naturally, we set the locale to en_GB as opposed to the default en_US, and the time zone to Europe/London. The differences are generally quite subtle, but the different representation of dates DD-MM-YY vs MM-DD-YY will be confusing for UK-based end users.

They keyboard layouts of a few of our members' laptops are American, so we decided against setting the keyboard to UK in case it caused more problems than it was intended to solve during development. If it turns out that the majority of production systems will use UK keyboard layouts (as would be expected), this can be easily set in the initial installation script.

Networking and Shared Drives

The drivers for both wireless and wired connections are installed by default. In the future, we may install each driver depending on what adapters are available to cut down on unnecessary drivers and overhead, but it was felt that installing both drivers would ensure maximum compatibility and connectivity.

By default, the computer will attempt to start the daemons for both wired and wireless connections during the boot cycle. On our test systems within the virtual machine, we found that the wireless daemon startup failed in the absence of a (virtual) wireless adapter, so the RAM/CPU overhead incurred by the extra daemon and drivers are minimal if the hardware is unavailable; the benefits of better connectivity outweigh the extra storage space that the drivers require.

Printing

Likewise, to ensure maximum compatibility, we installed a set of drivers such that a majority of printers would work. Like networking, we could potentially make the installation of these drivers optional or customizable, but before we scripted a menu for the end user to make a choice, we felt that compatibility took precedence over the little bloat this would cause.

The drivers installed included CUPS (the standard daemon), fonts (in the form of gsfonts), and a few more vendor-specific packages (hplip, splix etc), and related generic packages (foomatic etc). Although not directly related to printing, cups-pdf was also installed for pdf file creation.

User Accounts

This was one of the areas where it was difficult to emulate the Chrome OS or even Jolicloud. In an ideal world, the OS login would be integrated with the user account login for web services. However, this would require a lot of backend work capturing the inputs to the display manager (aka login screen), 'pre-logging in' the user, then launching a browser-based panel with the user already logged in.

In Chrome OS, this is handled as part of Chrome Sync. For Google, they have merged the concepts of the browser and the OS, and thus when the user logs in, he is logged into both the OS (for security and access) and the browser (for customizable synced areas such as themes and bookmarks).

Jolicloud works rather differently, in that the browser is considerably more lightweight, and the customization is focused on the launcher panel (presented as a website). Upon login to the OS user session, a full screen browser page of the Jolicloud panel is presented, where the user is required to login to the Jolicloud website (or displays the launcher panel if the user is already logged in). This secondary login provides the customizations and integration with the APIs of other web services, while the first login is purely to prevent unauthorized access to the local machine. Its usual purpose of owning files and handling user customizations within the OS (e.g. wallpaper, desktop environment) is irrelevant since everything is done through the web, and if multiple users were using the computer, they would use the same local account, but different web logins.

We decided to work towards Jolicloud's model because we found it worked well and offered adequate user customization in our evaluation of Jolicloud, and it was unfeasible to implement something as integrated with the OS as is the case with Chrome OS. On the local client, we would have a single user (imaginatively named 'user') with a nominal password and root, which should only be used during development or for debugging. Like Jolicloud, the user will see a further web-based login page once the desktop loads, and that secondary login will lead to the app launcher and other services/applications.

Audio

The default Arch Linux installation includes the ALSA driver (though the configuration for it, alsa-mixer, had to be installed separately). The alternative is OSS, but ALSA is generally expected to work with most hardware.

The audio is one area where relatively little work was done on. The installation script automatically sets the audio level of most devices to 0 dB, but during initial testing, the audio failed to work. Due to time constraints, we couldn't figure out whether the problem was related to the configuration of the

virtual machine or hypervisor, or whether there were missing packages within the guest OS. Until we can identify the problem, we are currently just using the ALSA driver.

Video

The Arch Linux installation guide recommends drivers for the four main driver vendors: Intel, AMD/ATI, Nvidia, and SiS. In addition, there are custom drivers for use with the vmware virtual machine, which were noted separately. In the installation script, there are a set of nested if statements, which detect the hardware to be installed on, and installs the appropriate drivers accordingly.

The SiS drivers were not included because of its rarity in the wild. Although the same could be said for the vmware drivers, they were included as the installation was being developed within a vmware guest environment. Where there was a choice between proprietary and open source drivers, the former were chosen as they tended to be more actively developed and supported by the hardware vendor and performance would therefore be expected to be marginally better. The usual advantages of using open source drivers don't apply in this case as we are not bound to use drivers released under certain licenses, nor are we expected to make any changes to the code base of the drivers.

Window System

The X Window System was naturally chosen for its maturity, stability, and compatibility with the most popular desktop environments. There weren't any alternatives available which matched X11 in terms of maturity or usage – the most popular desktop environments, GNOME and KDE, are both designed to be run on top of X11, and no other windowing system exist for (Arch) linux which has a comparable user/developer base.

Desktop Environment

We opted to use a standalone window manager instead of a complete desktop environment to cut down in unnecessary bloat, such as text editors and bundled web browsers. It was also decided that we'd use relatively mainstream packages so the compatibility and support available would be greater. Having whittled down our shortlist to KDE, GNOME, Xfce, and LXDE (and their window manager standalone equivalents), the first two were ruled out due to their notoriously large number of dependencies and speed issues.

Xfce and LXDE are both similar in terms of developer/community support, feature set, memory footprint, and extensibility. Although it was decided that we would be installing the file, window and display managers separately as standalone packages, consideration was given for installing packages together that would otherwise have been part of the same desktop environment (e.g. Xfwm4 and Thunar) for reasons of compatibility and available support/documentation since they are often installed together.

With regards to the display manager, the natural choice for the LXDE-based group was LXDM, whilst the equivalent for Xfce was SLiM (as suggested by the documentation on xfce.org; the alternative that was suggested, LightDM, was not available on the official repositories). SLiM, however, proved too minimalist as it didn't have preconfigured buttons to shutdown/restart at the login screen – those actions required the users to type in 'halt' and 'reboot' respectively into the username field.

After testing of both the Xfce and LXDE packages, we settled on using LXDE-based packages:

- The Arch wiki documentation on Xfwm was virtually non-existent (the closest match was on Xfce), as opposed to a very comprehensive page on setting up Openbox. This proved to be the biggest hurdle to installing Xfwm, the make-or-break factor in choosing Openbox over Xfwm.
- The LXDM documentation was plentiful and allowed a relatively large number of customization options compared to SLiM.
- The OS used xdg-open directly within Openbox to manage default applications, whilst Xfce required the use of exo-open which added another layer of complication.

With Openbox decided, for reasons noted above, we went with its usual associated packages, LXDM and PCManFM.

Drive Mounting

The Arch wiki recommends that udev, udisks, and their associated wrapper scripts be used to automatically mount USB and external drives. Going with the philosophy that packages from the official repository are preferred, we arrived at a shortlist of udevil, udiskie, and a combination of pmount/udisks2/spacefm. The last option was frowned upon because it required the installation of spacefm, a relatively immature file manager.

We tested both udevil and udiskie on our guest machines, however, they both showed similar problems, in that integration with our selected file manager (PCManFM) was limited, so users couldn't always unmount a drive from the file manager graphically. Furthermore, getting it to automatically mount an external USB drive became a hit-or-miss, for reasons unknown.

Before we ventured further with those packages, we decided on using the automount within PCManFM so the problem of allowing the end user to graphically unmount a disk was solved. Having installed the prerequisite GVFS, the reliability of the automount was still worrying, and problems weren't limited to mounting external partitions – the rubbish bin frequently gave error messages.

After much investigation, we pinpointed the problem to the dbus. The dbus allows different applications to talk to each other, in in our case, it appears that the file manager couldn't talk to the relevant applications responsible for mounting the drives because a common dbus session was not available. By first using 'dbus-launch pcmanfm' instead of 'pcmanfm', we were able to ensure that pcmanfm had a dbus to talk to, and the USB automounting subsequently worked flawlessly. Our final fix was to initialize dbus at the same time as launching the desktop session, when the instance of the window manager (openbox) was launched. Any applications opened after openbox would therefore be guaranteed to have a valid dbus session to work with, without having to worry about initializing it.

Web Apps

In our trials of Jolicloud, each web app we launched appeared in a full screen window of its own, rather than inside a conventional web browser window. Even though the apps were web based and were run within Jolicloud's browser of choice (nickel-browser, a variant of chromium), the apps appeared like a traditional desktop app with little sign that it was essentially an interactive website.

Since it was unfeasible for us to use nickel-browser due to lack of documentation, it was decided that chromium would be a suitable option since it also offers the same functionality through the use of the "--app" flag. (Kiosk mode was considered, but that would be unsuitable for a multi-windowed multi-tasking business computing environment, since it would be impossible to resize the home launcher

and the taskbar/desktop panel would be hidden.) By launching chromium using a command in the form of “chromium --app=http://*” we can open a web site in app mode, allowing chromium to load the website without the usual browser visual overhead such as navigation buttons and address bar. When the OS boots up, a chromium window in app mode is automatically opened, showing a login or launcher screen. Also, the right click event handler in the desktop has been changed from the default applications menu to running a new instance of the home launcher in chromium app mode.

Desktop Applications

In our tests of Jolicloud, we found some differences between the web-based version and the ‘desktop’ version – the latter contained options for changing settings related to the device the instance of JoliOS was being run on. The JoliOS browser has a modified user agent string, from which the server identifies is running on JoliOS and displays links to change the device settings. Upon clicking these links, the browser launches the relevant desktop apps. This functionality is achieved by running a local server on each JoliOS instance – when a hyperlink to a local application is required, the link (re)directs to a local loopback address, which the localhost server then picks up and launches the required applications or commands.

Xdg-open Handlers

During the development process, we found a local server written in ruby that would listen for certain requests and execute applications locally by executing the appropriate commands through xdg-open. However, this package was dropped in favour of our own custom solution which turned out to be more straightforward. We also had the choice of using the original daemon used within Jolicloud itself, but the lack of documentation meant that examining and implementing the source code of the appropriate modules would be excessively time consuming.

Our initial solution for opening websites within the home launcher in app mode took advantage of xdg-open, creating a new handler entry called ‘chromiumwebapp.desktop’. We then defined a new protocol called ‘webapp’, and set chromiumwebapp to handle any links with the mime type x-scheme-handler/webapp. Any links within the home launcher that should open a website as an app was prepended with “webapp:” – chromium would be at a loss over the handling of this protocol type, and would pass it to xdg-open. Having defined the default handler for x-scheme-handler/webapp, the custom-defined chromiumwebapp would strip the protocol name from the hyperlink (passed in as the argument), and open a new instance of chromium in app mode for the website in question.

We were unable to implement a similar mechanism for opening other desktop apps such as the file manager or settings due to time constraints. However, it isn’t expected to be of much difficulty – we would simply define another custom protocol and xdg-open handler, which could be used to manipulate the argument (from the hyperlink) and execute commands and scripts as appropriate. Since we have the freedom of executing any script in the handler, we could indeed define a generic handler that could execute any script or binary based on the argument passed into it from the hyperlink, though security could become an area of concern. If we decided to include a general browser (instance of chromium) within our launcher panel, complete with the full browser experience and address bars etc, we could implement a handler that executes ‘chromium’ without the switches in this way.

Desktop Panel

To enable the end user to multi-task on the computer intuitively, a taskbar/desktop panel would be required. How useful this would be in terms of the number of concurrently open windows are disputable, given the small amount of RAM that devices such as the Raspberry Pi possess, but it was felt that the ability to multi-task should be present.

The primary area of consideration was the availability of a system tray (to enable the display of status icons for networking) and a small memory footprint. We eventually settled on tint2, a minimalist desktop panel with a package size an order of magnitude less than the default LXDE option (LXPanel). One of the main downsides that limit tint2 from more widespread usage is the lack of a launcher in the standard repository build, however since we'd expect the end user to access their required apps through the home launcher panel web interface, a panel launcher is of limited use. Setup was straightforward – the only problem that occurred was that it failed to automatically generate the configuration file upon initial launch; this was remedied by storing a copy of an example configuration file online and saving it into the appropriate directory during the OS installation.

Custom Installation Script

To simplify the installation process for mass-installs, we developed a script to automate the fetching of packages, installation, and configuration of the system. Initially, we tested each update of the script by remastering the default x64 image and remastering the live CD. However, this eventually proved to be too time consuming as updates to the installation script became increasingly frequent, and remastering the CD consisted of extracting the ISO, extracting the live file system, placing the required scripts into the file system, and rebuilding the live file system and ISO.

In subsequent iterations, the script was only updated on github – each installation required booting from the official (or remastered versions of the) live CD, cURLing the latest script from github to a local file, chmodding it to executable, and running that. Whenever certain configuration files were required to be set up, they were either generated on the fly using 'echo * >' or a predefined configuration file was stored on the github repository and cURLed to the local machine.

```
sgdisk /dev/sda -Z -o -n 0:0:0 -g
mkfs.ext4 /dev/sda1
mount /dev/sda1 /mnt
pacstrap /mnt base
```

/dev/sda is wiped and a new partition filling all available space is created. In typical usage, /dev/sda is the first disk available and is where the user intends to install the OS. There is, however, the risk that if multiple disks are attached, the wrong disk will be wiped. In the automated script, there are no menus or options available to the user, since all prompts are suppressed. Some of the flags could be modified to allow prompts if required, so that the user will have to manually confirm which disk he wishes to install the OS.

Having created a single partition spanning the entire disk (sda), it's assumed that the first partition will be sda1, and it's subsequently formatted to the previously discussed ext4, and the partition mounted to /mnt. At this point, the partition is empty.

Since interactive installer (AIF) was removed from the official Arch ISO, it has been replaced with a set of smaller hands-off scripts known as 'Arch Install Scripts'. The most notable part of this is pacstrap.in which installs 'core' packages enable a basic Arch system to boot, and optionally (depending on arguments called), 'development' packages to enable the installation of unofficial packages and more developer-oriented software.

```
genfstab -U -p /mnt >> /mnt/etc/fstab
```

After the installation of the core packages, another of the official automated scripts is called to generate the fstab automatically. This file keeps track of any mounted partitions while the OS is running; at this point only one entry is expected to be in the file (sda1). The documentation recommends that this file is verified manually after generation, but in our testing, this proved unnecessary.

```
sed -Ei 's/^#+(en_GB.UTF-8)/\1/' /mnt/etc/locale.gen
arch-chroot /mnt locale-gen
echo LANG=en_GB.UTF-8 > /mnt/etc/locale.conf
ln -s /mnt/usr/share/zoneinfo/Europe/London /mnt/etc/localtime
arch-chroot /mnt hwclock --systohc --utc
```

The locale is first set in locale.gen by uncommenting en_GB.UTF-8, then applied throughout the system by running locale-gen. A wrapper script provided by Arch is used to chroot into the newly installed OS and run commands within that environment. The language is also set in /etc/locale.conf manually to make it persist after a reboot. We suspected that some of these commands may be redundant, but they are not CPU/time-intensive to run and further testing would be required to prove if any of them are not required.

Time zones are set to Europe/London, and the hardware clock time set as appropriate. On the Raspberry Pi, a hardware clock doesn't exist, so further research in this area would be required, possibly resetting the OS clock to the correct time after every boot cycle.

```
arch-chroot /mnt hostnamectl set-hostname "archclient"
echo "archclient" > /mnt/etc/hostname
```

The hostname is set to 'archclient', and defined in /etc/hostname to allow it to persist after reboot. We had considered appending a random string to the hostname to differentiate multiple computers running on the same network, but after noting that this would be connected to the cloud rather than to a local network, use of the same hostname on different installations wouldn't be an issue, since user sessions are defined by the user logged on rather than traditionally, the physical computer.

```
arch-chroot /mnt pacman -S wireless_tools wpa_supplicant
wpa_actiond dialog sudo syslinux gptfdisk alsa-utils xf86-video-
vesa xf86-input-synaptics xorg-server xorg-xinit xorg-server-
utils xorg-twm xorg-xclock xterm chromium gamin cups cups-filters
ghostscript gsfonts wicd-gtk ifplugd preload gutenprint foomatic-
db foomatic-db-engine foomatic-db-nonfree foomatic-filters hplip
splix ntfs-3g smbclient lxdm openbox-themes obconf ttf-liberation
tumbler pcmanfm oxygen-icons gvfs tint2 --noconfirm
```

Packages are then installed using Arch's default package manager, pacman. The flag at the end suppresses any prompts, including confirmation messages related to dependencies, compatibility, download, and installation. For the packages defined, compatibility was not found to be an issue, and dependencies are resolved automatically. The definition of some packages are redundant since they would be required by other packages anyway as a dependency, but inclusion of them wouldn't affect the ultimate result, and so was left within the command.

```
arch-chroot /mnt systemctl enable net-auto-wireless.service
arch-chroot /mnt systemctl enable net-auto-wired.service
arch-chroot /mnt systemctl enable preload.service
arch-chroot /mnt systemctl enable dhcpcd@eth0.service
arch-chroot /mnt systemctl enable wicd.service
arch-chroot /mnt systemctl enable cups.service
arch-chroot /mnt systemctl enable lxdm.service
```

Several services are enabled to run at boot time. Networking services for both wired and wireless connections are enabled, as is dhcp support. Preload is program recommended by the Arch documentation to be used to speed up the OS startup. This was not tested by us, so whether or not it has a noticeable impact is debatable.

LXDM is enabled to load a GUI at startup, which will then load the main desktop (X11/Openbox) upon successful login. Wicd is started as a daemon, which when paired with wicd-client, allows a logged-in user to modify network settings and connections. CUPS was not tested due to a lack of hardware, but the daemon is required for CUPS to function correctly. If an interactive menu is implemented, the installation of CUPS or enabling of cups.service could be made optional for computers that are not intended to be used for printing.

```
echo -e "[multilib]\nSigLevel = PackageRequired\nInclude =
/etc/pacman.d/mirrorlist" >> /mnt/etc/pacman.conf
arch-chroot /mnt pacman -Sy --noconfirm
```

Multilib libraries are enabled to support 32-bit applications on 64-bit installations. Whether this is actually used/required hasn't been tested in detail, but the multilib repository is recommended to be enabled if the need arises. This is not expected to impact pacman on 32-bit installations. The local repository package database is then synchronized to include the additional packages.

```
arch-chroot /mnt useradd -m -g users -G
wheel,storage,power,uucp,sys,scanner,optical,network,lp -s
/bin/bash user
echo -e "password\npassword" | arch-chroot /mnt passwd
echo -e "password\npassword" | arch-chroot /mnt passwd user
sed -Ei 's/^#+\s*(%wheel ALL=\(ALL\) ALL)$/\1/' /mnt/etc/sudoers
```

A new user called ‘user’ is created, and included in groups as appropriate to allow him to access the respective types of hardware and OS settings. The password for both user and root is set to be the same (‘password’), given that it is there nominally rather than to differentiate between different local users. Going with a rather liberal security policy, the new user is also allowed to use ‘sudo’, though it is expected to be only used during development and not by the end user.

```
echo "exec dbus-launch openbox-session" >>
/mnt/home/user/.xinitrc
sed -Ei 's/^#*\s*(session=).*$/\1"dbus-launch
\/usr\/bin\/openbox-session"/' /mnt/etc/lxdm/lxdm.conf
sed -Ei 's/^#*\s*(autologin=)\w*$/\1user/'
/mnt/etc/lxdm/lxdm.conf
sed -Ei 's/^#*\s*(disable=)[0-9]?$/\11/' /mnt/etc/lxdm/lxdm.conf
sed -Ei 's/^#*\s*(lang=)[0-9]?$/\10/' /mnt/etc/lxdm/lxdm.conf
rm /mnt/usr/share/xsessions/openbox-kde.desktop
/mnt/usr/share/xsessions/openbox-gnome.desktop
/mnt/usr/share/xsessions/openbox.desktop
```

All openbox sessions should be run with dbus-launch as discussed previously. This is appended to .xinitrc so it can be run directly from console (via startx), bypassing the display manager if required. If the display manager is used, the default session is also set to run openbox-session through dbus-launch. Certain options (such as languages) in the display manager are disabled intentionally, either by changing the configuration file or removing the appropriate *.desktop session files. Autologin is also enabled within the configuration file and set to the only non-root user. As part of the system design, the authentication is primarily at the cloud/network level, and thus we can afford to be relatively lax in this area for the local machine.

```
curl
https://raw.github.com/horaceli/comp2013/master/chromiumwebapp.de
sktop > /mnt/usr/share/applications/chromiumwebapp.desktop
echo -e "!/bin/bash\nchromium --app=\${1:7}" >
/mnt/usr/local/bin/chromiumwebapp
arch-chroot /mnt chmod +x /usr/local/bin/chromiumwebapp
```

The custom handler used to handle ‘webapp’ links is available online on our github repository. Since it has to be defined in a certain format and we couldn’t easily enable it to substring the argument to remove the ‘webapp:’ preceding the actual link, this was outsourced to a short script called chromiumwebapp within /usr/loca/bin.

```
echo 'gtk-icon-theme-name = "oxygen"' >> /mnt/etc/gtk-2.0/gtkrc
mkdir -p /mnt/home/user/.config/openbox/
/mnt/home/user/.config/tint2/
/mnt/home/user/.local/share/applications

[…]

arch-chroot /mnt chown -R user:users /home/user
```

In one of our earlier builds, we found that PCManFM wouldn't display icons as expected. Here, it's defined explicitly. Also, certain directories are made at this point to enable the creation of user-based configuration files (notably for tint2 and openbox), and chowned to 'user' so they can be edited at a later stage.

```
cp /mnt/etc/xdg/openbox/rc.xml
/mnt/home/user/.config/openbox/rc.xml
sed -Ei 's/<action name="ShowMenu"><menu>root-
menu<\/menu><\/action>/<action name="Execute"><command>chromium -
-
app=http:\/\/smartbusinesscloud\.azurewebsites\.net<\/command><\/
action>/' /mnt/home/user/.config/openbox/rc.xml
sed -Ei 's/<number>4<\/number>/<number>1<\/number>/'
/mnt/home/user/.config/openbox/rc.xml
```

The global openbox configuration file is copied to create a user-based version (which takes precedence over the global version). This is then edited so right-clicking the desktop will open the web-based launcher panel instead of an application menu. The number of desktops is also reduced from the default 4 to 1, to simplify the UI/UX.

```
curl https://raw.github.com/horaceli/comp2013/master/tint2rc >
/mnt/home/user/.config/tint2/tint2rc
```

As discussed earlier, a predefined version of the tint2 configuration file is downloaded and placed in the appropriate directory. This ensures that the system tray is enabled, and uses the default theme as is listed on the official tint2 website.

```
echo -e "[Default Applications]\nx-scheme-
handler/webapp=chromiumwebapp.desktop" >
/mnt/home/user/.local/share/applications/mimeapps.list
echo -e "tint2 &\nwicd-client --tray &\nchromium --
app=http://smartbusinesscloud.azurewebsites.net" >
/mnt/home/user/.config/openbox/autostart
```

The default handler for 'webapp' links is defined for the user (not globally) in mimeapps.list, and the network manager (wicd), desktop panel (tint2), and web-based launcher panel (chromium in app mode) is set to run when the window manager session loads.

```
arch-chroot /mnt syslinux-install_update -i -a -m
sed -Ei 's/(APPEND root=\/dev\/sda)3/\11/'
/mnt/boot/syslinux/syslinux.cfg
```

The automated syslinux setup script is then run. Curiously, the default boot partition is set to sda3 – this is remedied running a sed regex replacement on the configuration file.

```
for i in Master Headphone PCM Front Surround Center LFE Side CD
Video Phone Aux; do
    arch-chroot /mnt amixer -c 0 set "$i" playback unmute 0dB
done
```

Since the audio devices on different computers will vary, this will loop through and enable the sound for some common audio device names. Where a device with such name doesn't exist, an error message will be seen, but is safely ignored as the loop progresses onto the next device.

```
lspci | grep VGA | tr "[:upper:]" "[:lower:]" | grep nvidia
if [ $? -eq 0 ]; then
arch-chroot /mnt pacman -S nvidia lib32-nvidia-utils --noconfirm
else
  lspci | grep VGA | tr "[:upper:]" "[:lower:]" | grep "intel
corporation"
     if [ $? -eq 0 ]; then
          arch-chroot /mnt pacman -S xf86-video-intel lib32-
intel-dri --noconfirm
     else
          lspci | grep VGA | tr "[:upper:]" "[:lower:]" | grep
"advanced micro devices"
          if [ $? -eq 0 ]; then
               arch-chroot /mnt pacman -S catalyst-dkms lib32-
catalyst-utils --noconfirm
          else
               lspci | grep VGA | tr "[:upper:]" "[:lower:]" |
grep "vmware"
               if [ $? -eq 0 ]; then
                    arch-chroot /mnt pacman -S xf86-input-
vmmouse xf86-video-vmware svga-dri --noconfirm
               fi
          fi
     fi
fi
```

These are the previously mentioned nested if statements which runs lspci to list the system hardware, grep to detect if the VGA component is created by any of the tested brands, and installs the required packages. A fallback video driver is already included for all installations (xf86-video-vesa), but performance with a driver matching the actual hardware will be much improved.

```
reboot
```

The script is intended to install everything automatically and reboot, so the user should see the installed OS with the web launcher open when everything completes. The only area yet to be explored or implemented is how to run the script automatically when the live CD is installed, or whether the user should be required manually initiate the script (by typing a command or confirming a menu prompt).

Cloud Panel

Setup Process

- Install Microsoft SQL Server 2012 Express with SQL Server Management Studio
- Install Microsoft Visual Studio Ultimate 2012
- Install Ajax Control Tool Kit from VS package manager
- Include jQuery and Bootstrap in project

JavaScript framework	jQuery

CSS framework	Bootstrap
Web development tools	Microsoft Visual Studio Ultimate 2012 Ajax Control Toolkit
Database management tools	Microsoft SQL Server 2012 Express SQL Server Management Studio

Minimum hardware specification for development environment

Processor	For 32-bit, 2 GHz For 64-bit, 1.6 GHz
RAM	1 GB (1.5 GB if running on a virtual machine)
hard disk	12.2 GB
hard drive	5400 RPM
video card	DirectX 9-capable running at 1024 x 768 or higher display resolution

Version control

For web application source code version control, we have explored two ways: Microsoft Team Foundation Server and Git.

Since we use Visual Studio as development tool, MS TFS comes as the first choice since it is the default version control tool in Visual Studio. It offers source control, data collection, reporting, and project tracking, and is intended for collaborative software development projects. In order to use TFS, we need to install the TFS server and it requires a paid license. In addition, we have tried TFS in VS and found it very hard to use.

Therefore, we tried out Git and see if it can be an alternative. Below is the process of setting up Git in Visual Studio:

1. Install Git for windows (download from : http://git-scm.com/download/win)
2. Install Git Source Control Provider in VS

   Step 1

   - Launch Visual Studio, Select menu Tools | Extension Manager
   - Search online gallery for Git Source Control Provider
   - Download and install

   Step 2

   - Go to Tools | Options
   - Select Source Control in the tree view
   - Select Git Source Control Provider from the drop down list, and click OK
   - Verify the paths to Git tools are setup correctly

3. User configuration from git bash
4. Add project to Git version control from VS

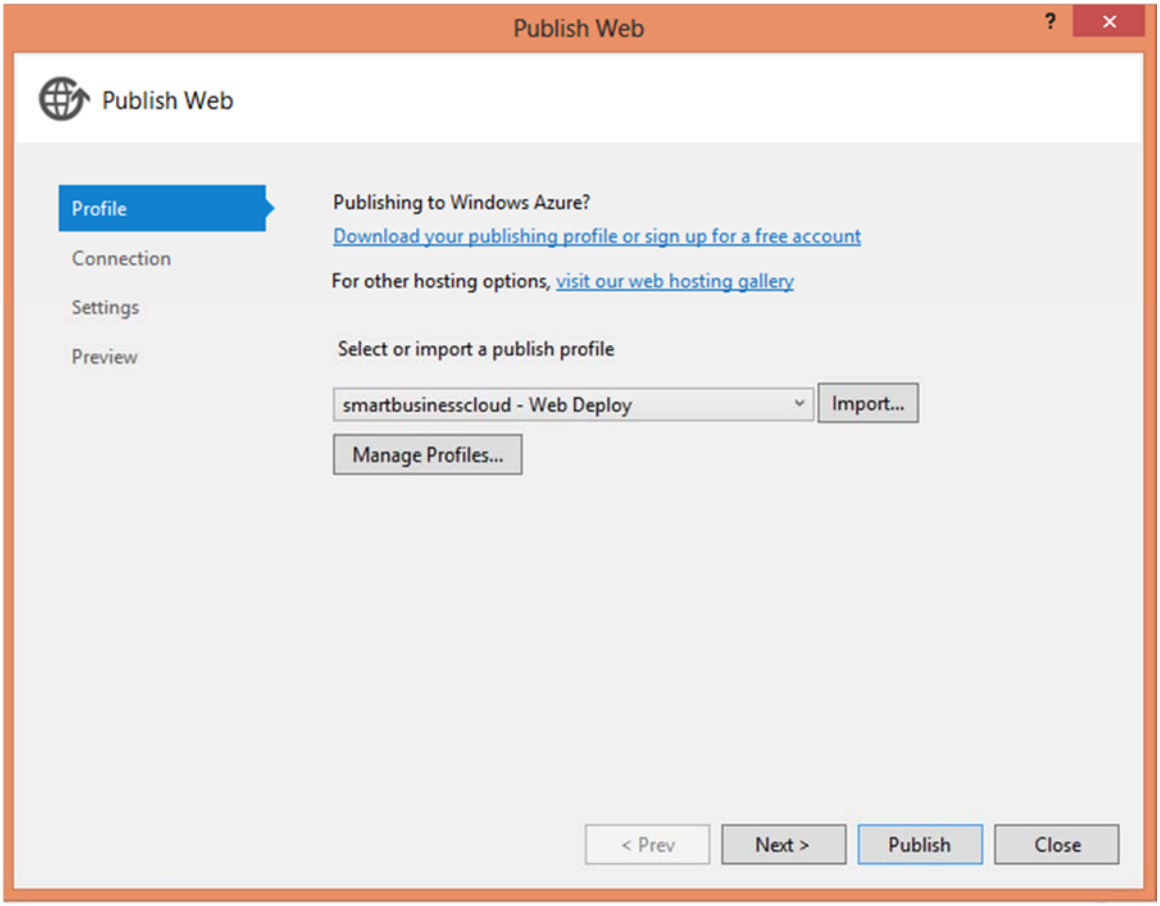
After trying out, we found that Git is easy to use with Visual studio since it provides a plug-in (Git Source Control Provider). Comparing to TFS, it is simple to manage, fairly shorter learning curve and also free. Hence, we decided to use Git as our version control method.

Publish Site

Windows Azure Website provides three ways to publish site: FTP, Git clone and publish from Visual Studio. We choose to use the last method. Below is the process of publishing from VS:

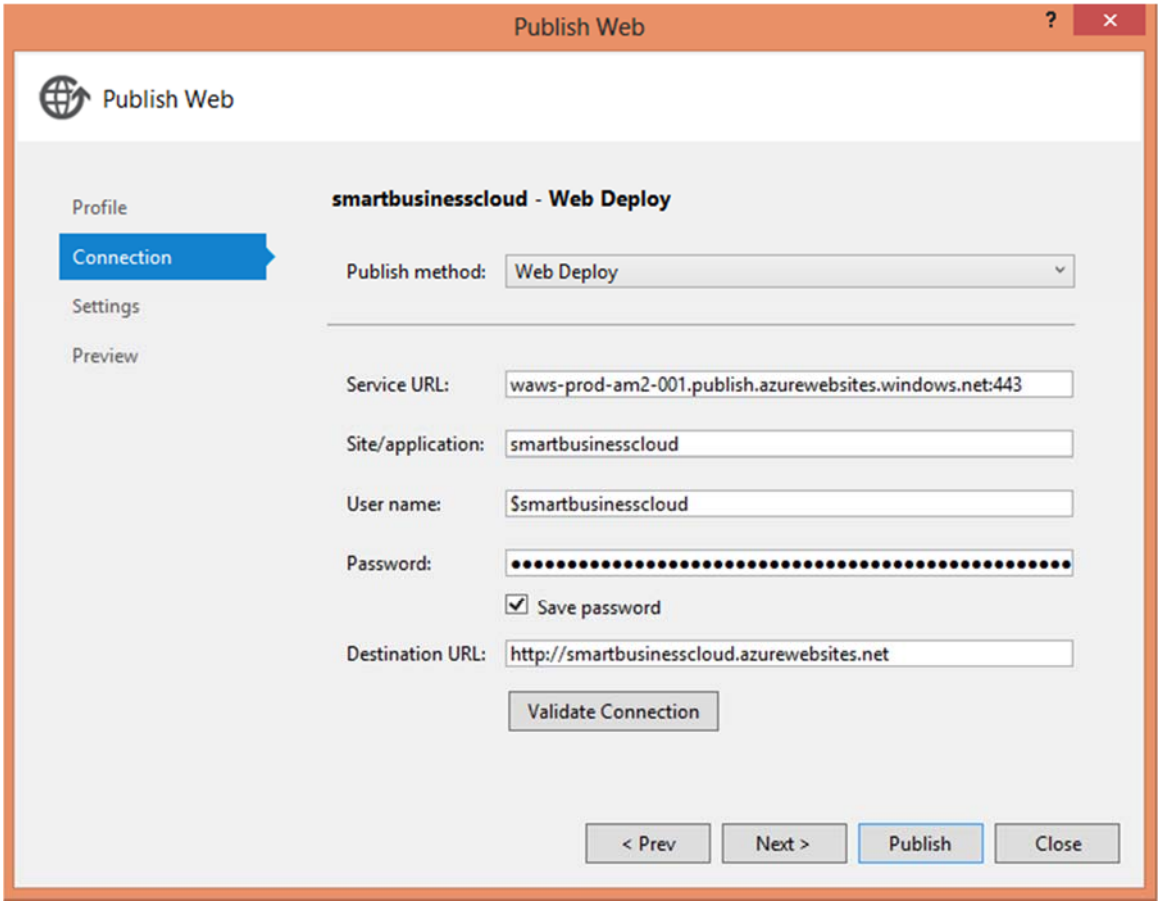
The process includes creating a publish profile, configuring the connection tab, configuring the settings tab and finally publish the site.

1. Creating a publish profile
    - Download publish profile from Windows Azure Website
    - In VS Solution Explorer, right-click the project and select Publish Web Site to open the Publish Web wizard
    - Import the downloaded publish profile from wizard

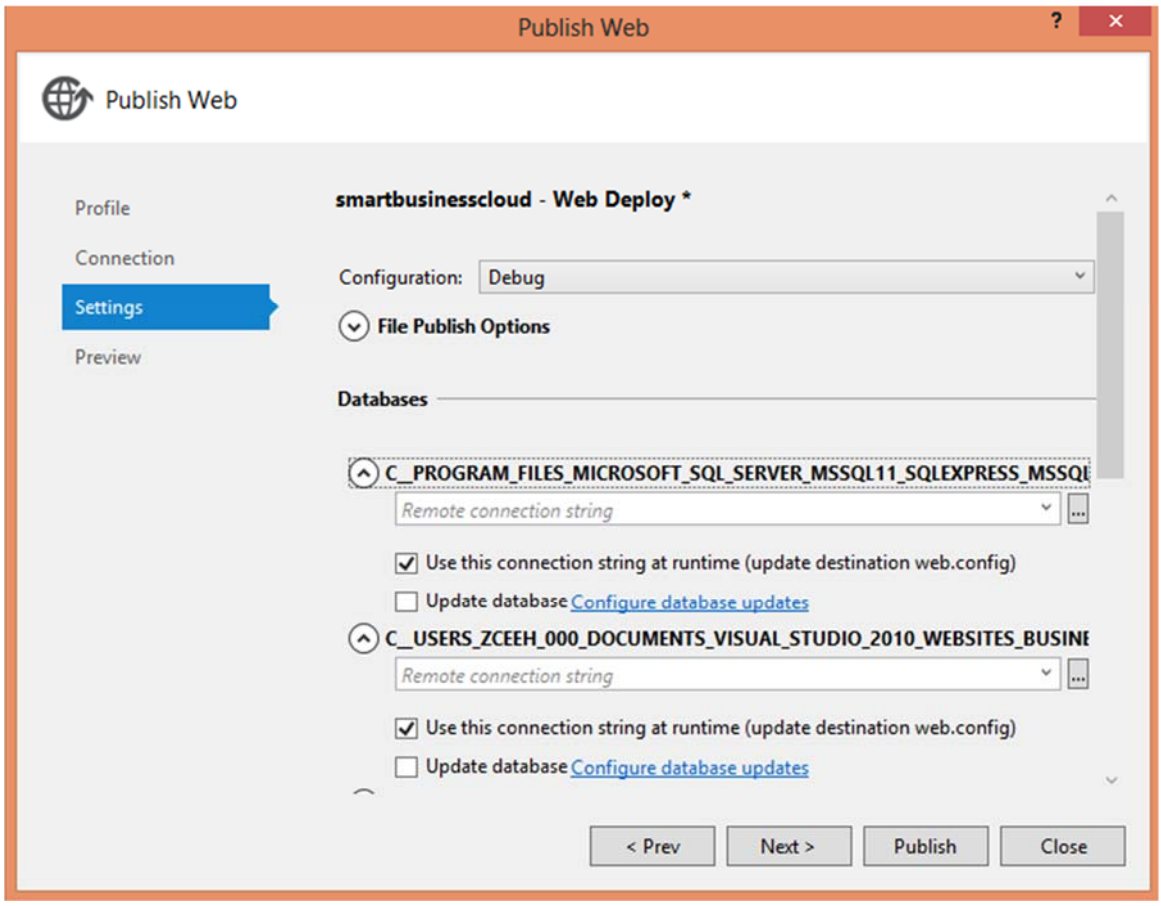


2. Configuring the connection tab
    - Select Web Deploy for Publish method
    - Copy the site URL from Windows Azure and paste into Service URL box
    - Enter Site/application name
    - Enter username and password for Windows Azure Website

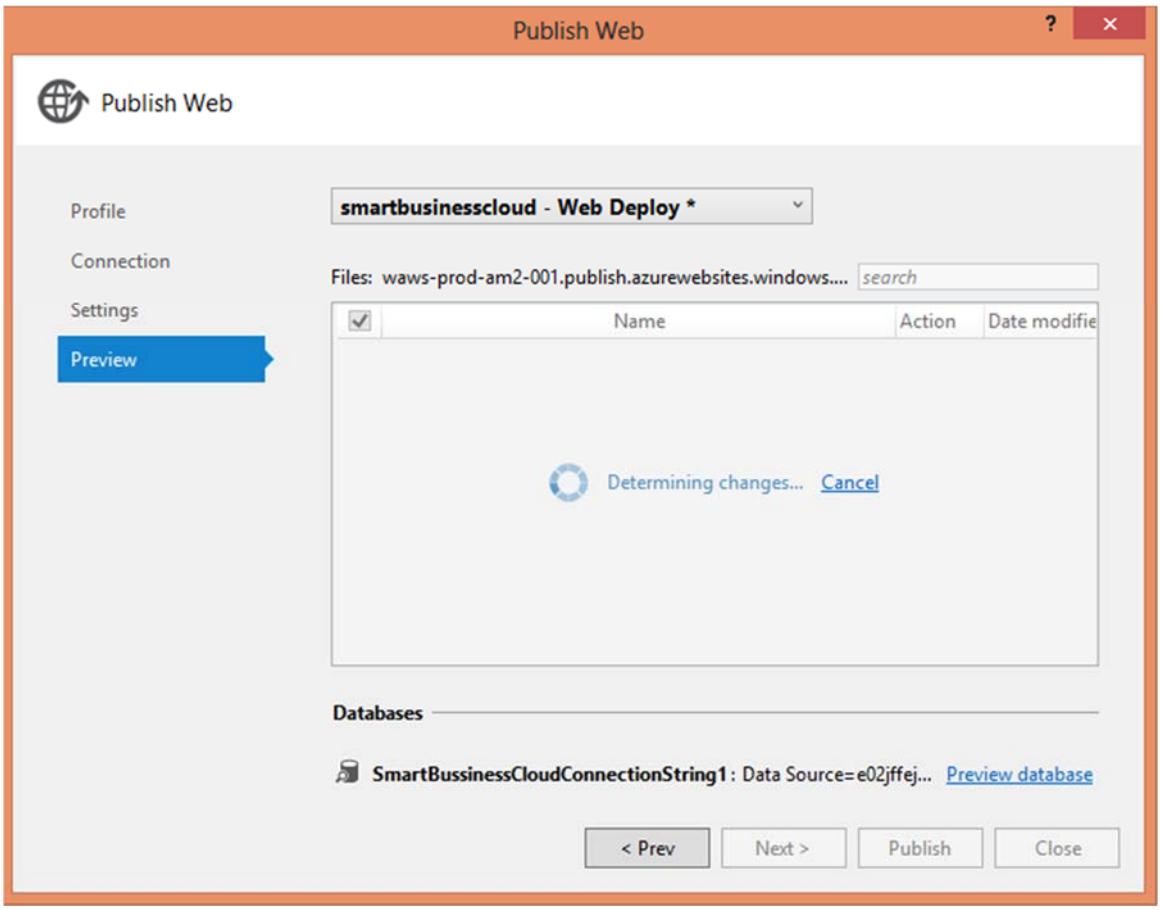
- Enter Destination URL



3. Configuring the settings tab
    - In the Configuration drop-down list, choose the build configuration to deploy
    - one entry for each database that the project uses is listed below after selecting the Web Deploy publish method
    - Check the connection string that want to use during run time, also check update database



4. Previewing changes and publish the website
    - Preview changes to website and database
    - Publish website



Business Cloud

So far, we have developed a basic version of an online business cloud. It is a platform provides business user with a desktop-look app panel where various kinds of enterprise apps such as accounting and tax filings are integrated.

The business cloud is now hosted on Windows Azure. The front-end is written in HTML5, CSS3 and JavaScript whereas the backend is written in C#. Our database is Microsoft SQL 2012.

We have stored a large amount of enterprise software in our database. The app panel webpage is dynamically generated from our database.

We have implemented the features as below:

- Register, login, logout
- Store personal information of each user in database
- Change desktop wallpapers
- Add/Remove Apps, each user has their own set of apps stored in database
- User can create their own app and add into their desktop
- More details will be demonstrated in video report.

OpenERP

OpenERP is an open source integrated enterprise resource planning (ERP) software manufactured by OpenERP.

We have installed OpenERP server on the virtual machine hosted in Windows Azure. The process involved of installing PostgreSQL (the database engine used by OpenERP), Python and then installing OpenERP server.

This software is also integrated to our business cloud app panel.

Packages

To identify the appropriate packages for the target platform, we have first outlined the main features, which are necessary and feasible, to be incorporated in the system to be developed. The main features are listed below together with information on chosen packages, web-based applications or existing open source softwares which are capable of providing pragmatic and direct functionalities to perform and present the features.

Office suite

Google docs

- Part free/Part proprietary, no subscription fee
- Platform independent
- Suitable to less complicated documents and worksheets
- General feature includes collaborative editing

Microsoft Office Web Apps

- Monthly subscription fee of $6 is needed to maintain the license
- Compatible with Windows, Linux and Mac OS platform
- It has familiar look and feel with desktop version of Microsoft Office
- General features include address book, collaborative editing, rss feeds and spam filter

LibreOffice

- Free to use software
- Linux compatible
- It is on par with most of the commercial competitors
- Does not have web integration
- Builds diagrams and sketches for data

Google docs is chosen because target customers do not need such complete documents editing functions. Besides, Google docs is distributed for free and free to use, also is being support on Linux. Microsoft Office Web Apps requires users to pay for each month. Although LibreOffice is free, it does not support web integration. Because we are aimed to create a web based software, therefore, we have to turn down the use of LibreOffice.

Business management software

OpenERP

- Linux compatible full-featured, fully-integrated business management system.
- Manages general ledger, bank journal, fix assets management, tax calculation, invoices, subsidiary ledgers of suppliers and clients and journal entries
- Displays employees' personal timetable, expense and leaves and allocations, recruitment and contacts of other employees

- provides plans and orders of manufacturing, bills of material for products, inventory control, management of orders and products of receive and deliver
- Able to plan projects and assign tasks and with clear categories
- Provides the product purchase and invoice control, the up-to-date sales information, invoicing and call for sales, PoS management, connects to inventory database
- displays data in a graphical way
- employee meal and vehicle management
- Free and full version

Compiere

- Linux compatible full-featured, fully-integrated business management system
- Open source for both individuals and companies
- Manages general ledger, account recoverable and payable, cash-based accounting, bank journal, balance sheet, profit and loss, statement of cash flows
- Provides tracking on the financial progress of projects, accounting transactions and documents associated with the project
- Manages advanced manufacturing planning and scheduling, MRP, production control, cost management, reporting, supports the product and invoice control, multi-warehouse management, has personal design warehouse management
- Has PoS management, transaction supports various methods, supports e-commerce, costumer management, tracking potential customers, arrange purchase order, manages invoices and payments
- Has two editions, community edition and enterprise edition. Enterprise edition have more functions including manufacturing and warehouse management
- Lacks human resources management

Both have similar functions and are able provide enough functionality for small startup businesses. However, we choose OpenEPR because it provides a full and free version and the Compiere only provides a 30-day trial.

Cloud Storage

Google Apps

- A service provided by Google allowing clients to customize their control panel with name and brand logo
- Features office suites, Gmail, Google Groups, Google Calendar, Talk, Docs and Sites
- Additional option to install other Google Apps from Google Apps Marketplace

Zoho

- An online web-based solution for professionals, businesses as well as individuals
- Includes collaboration applications such as chat, commentbox, docs, discussions, mail, meeting and projects
- Includes business applications such as assist, books, bug tracker, campaigns, creator, CRM, invoice, marketplace, people, recruit, reports, sites and support
- Includes productivity applications such as calendar, notebook, writer, show, sheet and Zoho Office for Microsoft SharePoint

- Free to use

Zoho is chosen because of its more complete and business oriented features. Besides, work done and uploaded can be stored online. It includes core applications for businesses such as invoice, CRM, reports etc. It is also free to use. Google Apps is a more general tool, not a business-targeted package.

Building and hosting websites

Xtgem

- A visual mobile site building tool
- Clean and simple interface
- Free service can be chosen, with ads posted on users' website or pay $5/month for an ad-free website
- Store locator feature
- Website hosted on xtgem domain

1&1

- Business websites templates to choose from
- Create industry-specific text, images and attractive layouts
- User friendly and does not require prior web design or software knowledge
- Customize the layouts and appearance of client's website easily
- Offer hosting and onsite domain registration service

WiX

- Online web-app
- Secure, reliable and dedicated web hosting
- User-friendly drag n' drop website builder
- 24/7 support center
- Set up online store
- Get social by integrating services such as Facebook, twitter

WiX is chosen as the main hosting service to incorporate to the system. It has more beautiful designs, more stable and easier to use hosting service, a 24/7 support service and social websites features can be added to the client's website easily. Xtgem is a tool that is required charges, therefore, this does not make it more suitable for small startup business. Although WiX and 1&1 both provide similar services, WiX is cheaper than 1&1 for 50 GB Web space which should be enough for a small startup business.

Browser

Google Chrome

- Most widely used by users
- One of the fastest browser
- Lots of Google Apps could be used alongside as a traditional web browser
- Intelligent start page and has its own task manager
- Currently have the best support for HTML5
- Google Chrome for Linux is still Beta
- Variant and open source version - Chromium is highly customizable

Mozilla Firefox

- Ahead of Chrome in offering some graphics hardware acceleration
- Highly Customizable
- Mac, Windows and Linux compatible
- Lacks of Chrome's integrated support for Adobe Flash and PDF

Opera

- Turbo feature which makes slow connection faster
- Built-in mail and bittorrent clients
- Live tiles on start page
- Mouse gestures allow user to navigate faster
- Mac, Windows and Linux compatible
- Lacks of Chrome's integrated support for Adobe Flash and PDF

Google Chrome is selected due to its more complete features for daily or business use. It is fast and free. Besides, updates are pushed automatically from time to time. Chrome also has better support in both HTML-5, Adobe Flash and PDF than both Mozilla Firefox and Opera.

Enterprise social

LinkedIn

- Set up company page to be viewed
- Allows company to display company's information to other professionals as well as other companies

Xing

- Supports more languages compared to LinkedIn
- Current membership count is still a lot lower compared to LinkedIn
- Most user and companies come from European
- Allow create a company page with pictures and pdf

LinkedIn is chosen because it is the largest companies and users professional social network around in the world. Most of users are with advanced degree, and it would be a better platform for recruiting potential employees and establishing a brand image.

Video Conferencing Software

Skype

- A cross-platform conference application which is widely used by individuals
- Quality of service is stable
- Skype-to-Skype voice and video calls are free
- Free instant messaging
- Share screen feature
- Call quality variable due to decentralized nature of network.

Google talk

- Connects to Gmail, support Google account- to -Google account and chats log can automatically save to the online Gmail mailbox

- Supports Google account to phone
- Able to connect with other instant messaging software services
- Audio and instant messaging only

Skype is chosen because it is more commonly used by individuals and companies. It supports instant messaging, audio and video conference, with good quality. Although Google talk has a similar functionality as Skype, Google talk is less popular that makes most people to not be as familiar as Skype.

Cloud storage

Dropbox

- Cross-platform application widely used, including mobiles.
- Support group share folder
- Free to storage starts from 2GB, can be expanded up to 18GB through referrals (500MB per referral)
- Maximum limit of uploading file is 300MB

Microsoft SkyDrive

- Maximum limit of uploading file is 2GB
- Free to have maximum 7GB storage for individuals
- Connects with hotmail
- Integration with MS Online Office

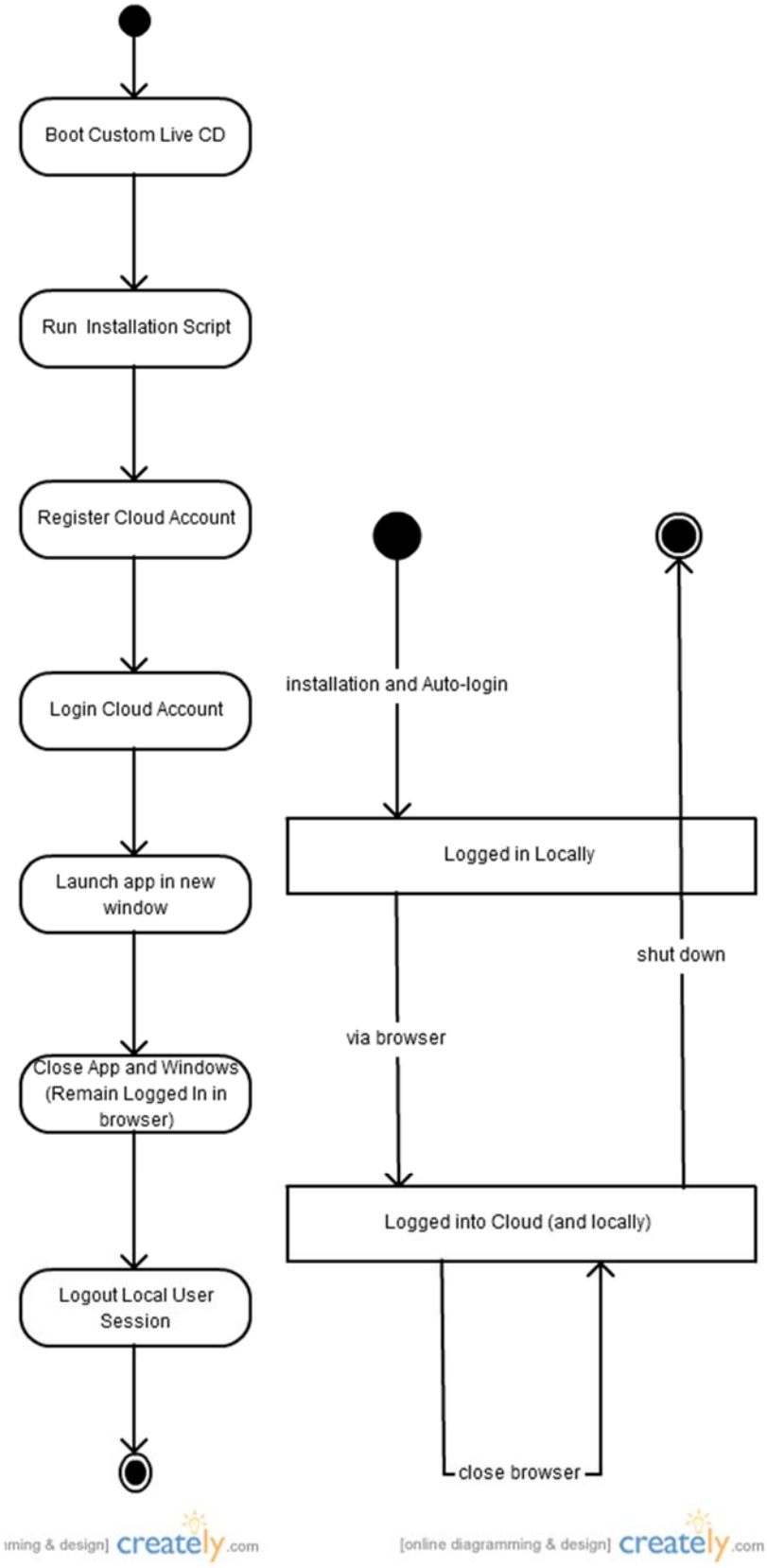
Google Drive

- Cross-platform application widely used, including mobiles.
- Maximum limit of uploading file is 10GB
- Free to have maximum 5GB storage for individuals
- Supports Google doc

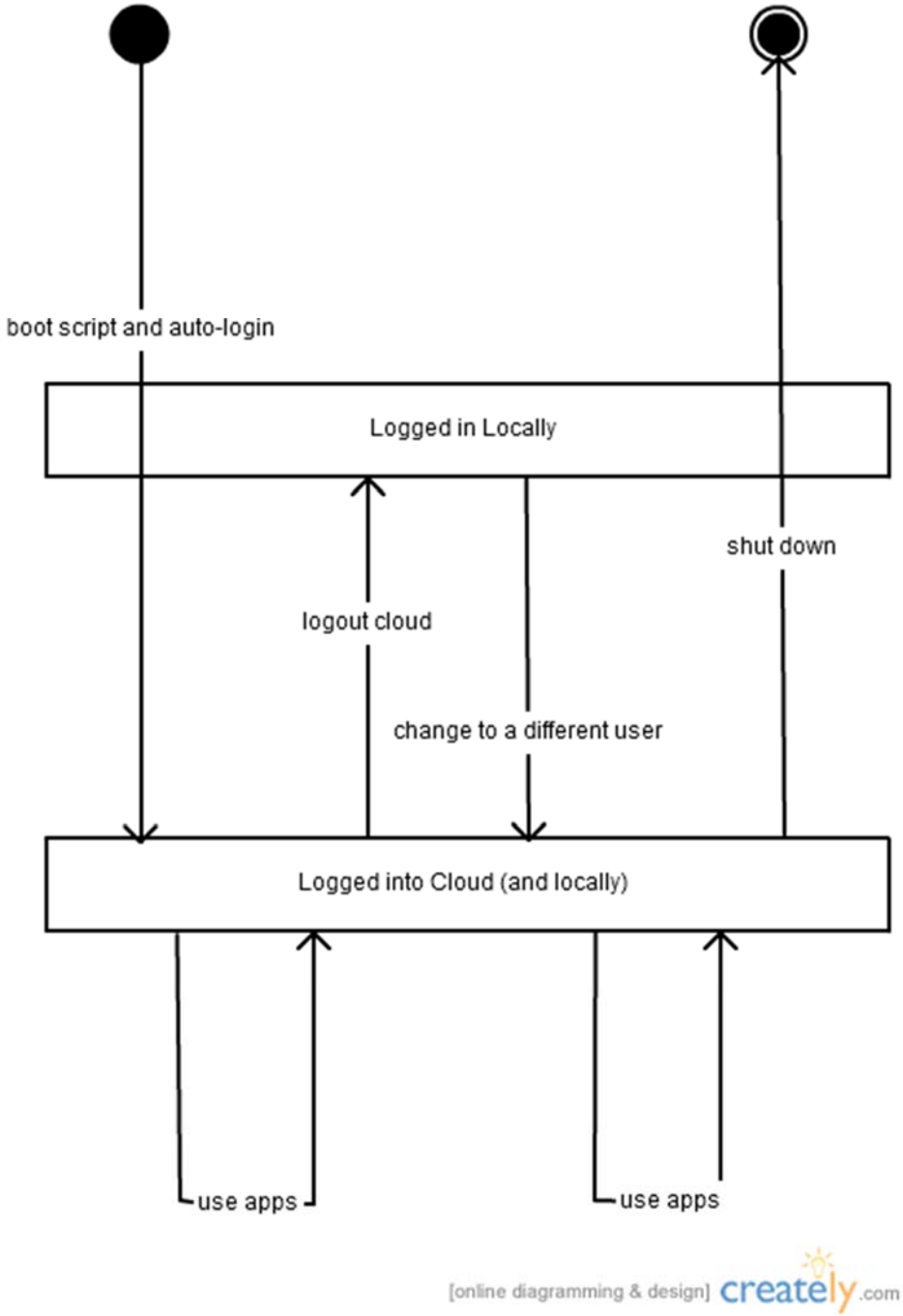
Dropbox is chosen of its great storage amount for individuals that employees can share more files with small groups for free. It can also share files in Android, iOS and Blackberry phones. Although Microsoft SkyDrive has a larger starting storage, it is not as much as an after-expanded Dropbox though referrals. Google Drive neither has a smaller starting storage than Microsoft SkyDrive, nor is not expandable.

Operation Models

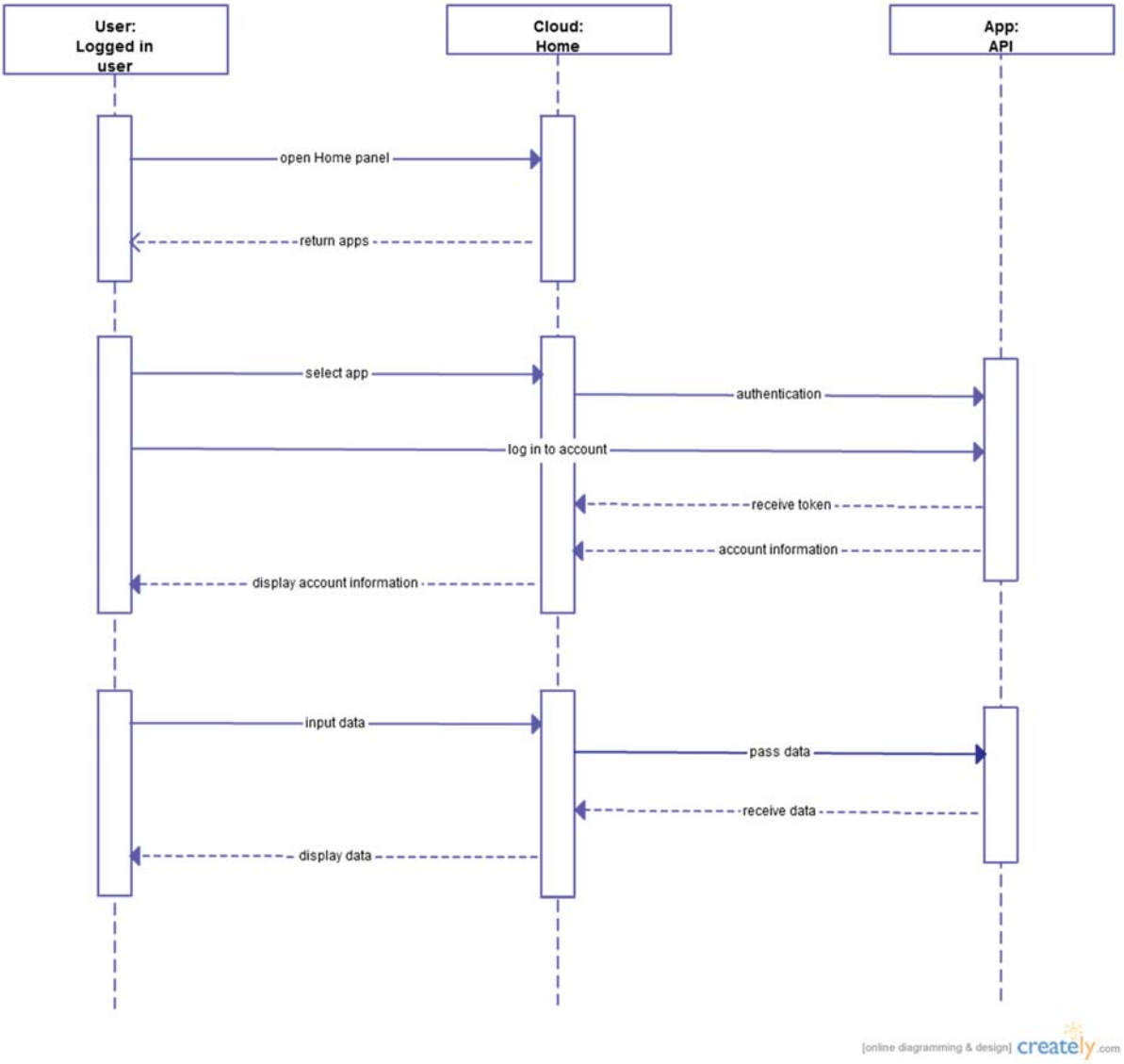
Initial Installation and Login



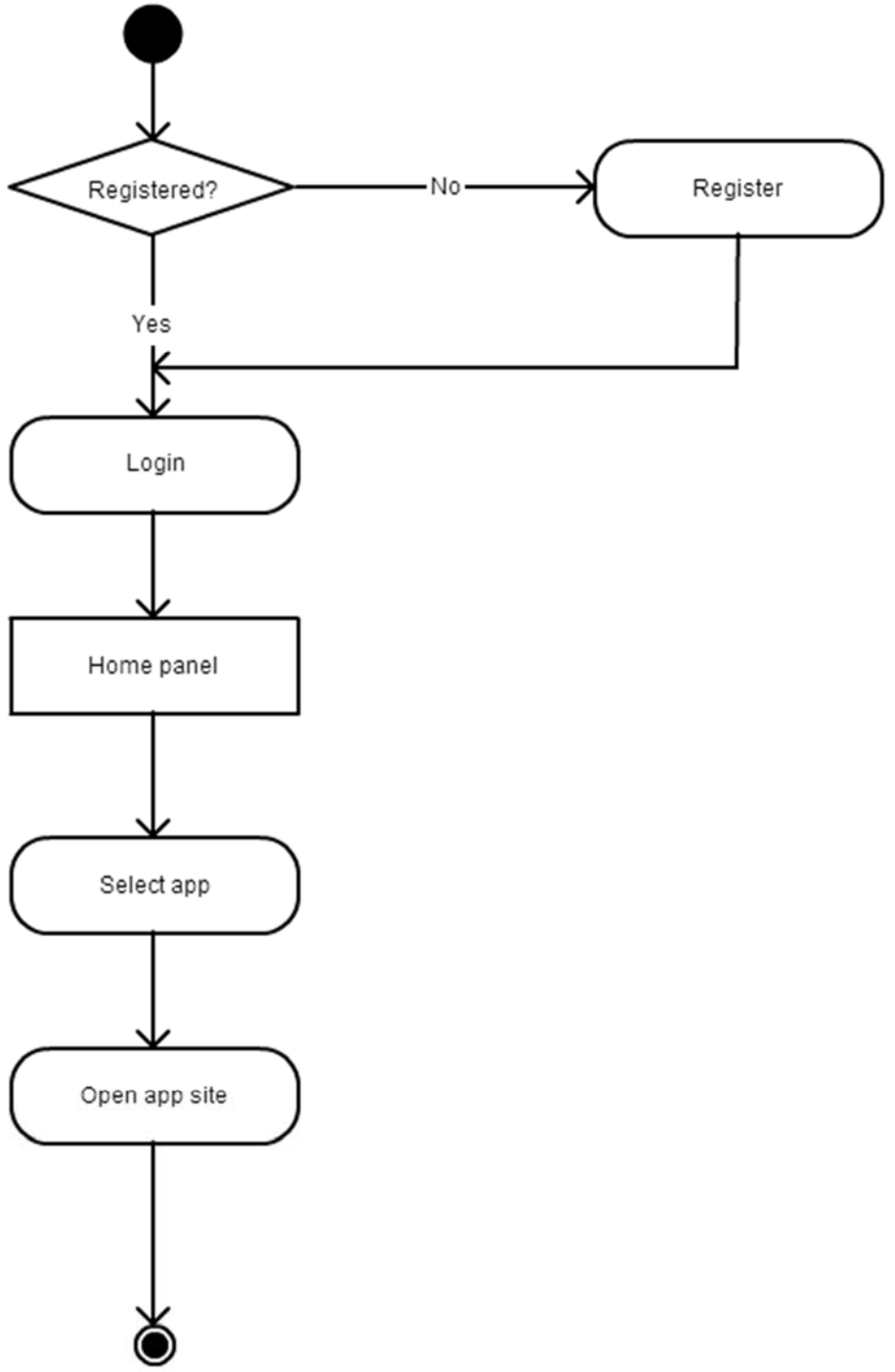
Changing Cloud Account



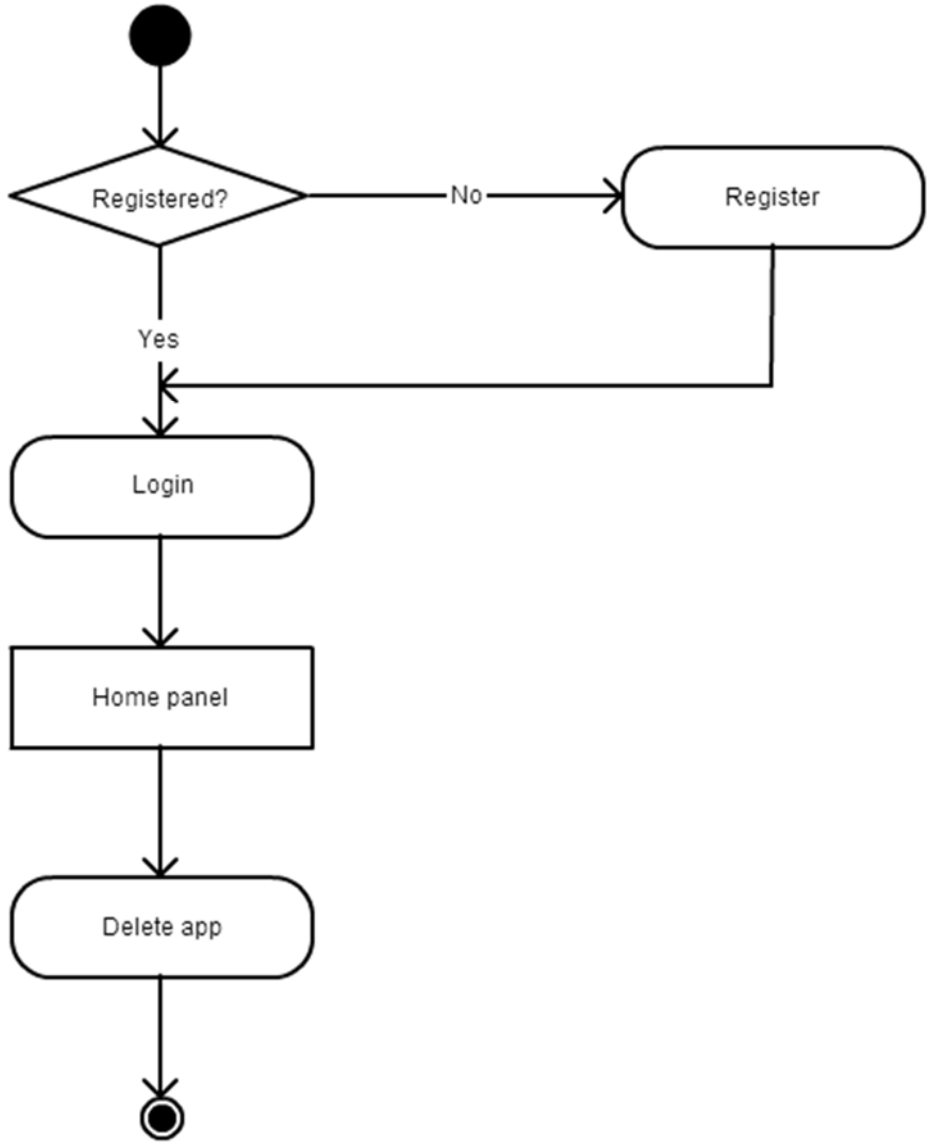
Integration with Apps' APIs (Proposed)



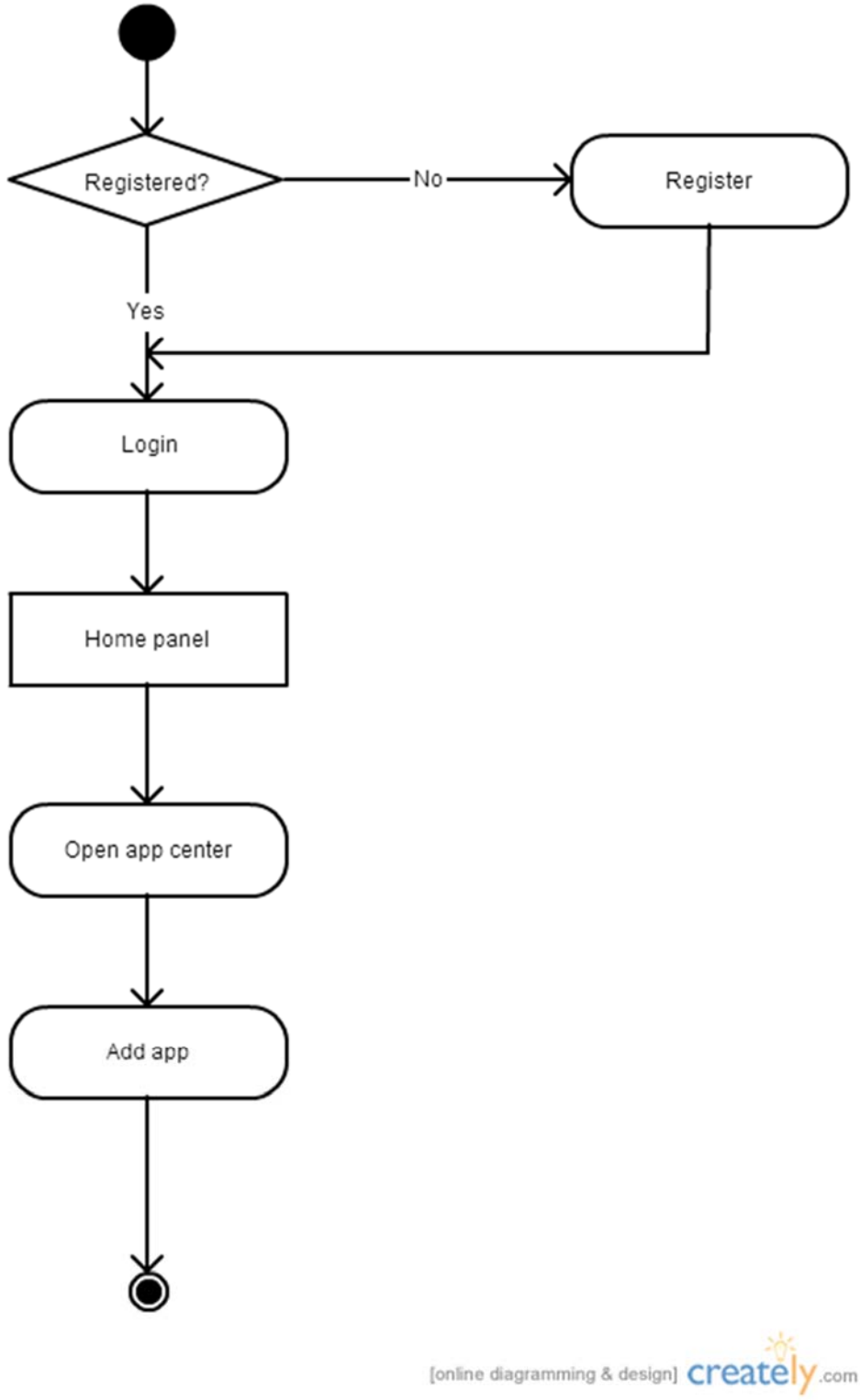
Apps Usage



App Deletion



App Addition in Home Panel



Testing

The testing of our implementation focus on testing web applications. Our development includes building a web desktop which comprise of other web based applications whether hosted on the official web application server or our own server.

Test Categories

Functionality Testing

- Test for all the links in webpages - *testing all outgoing links, testing internal links, testing links jumping on the same pages, test to check if there are any orphan pages, check for broken links*
- Test database connection - *testing data consistency, testing data integrity and errors during DB operations, validate and verify queries, check data retrieval*
- Cookie testing - *test application by disabling and enabling cookies in browsers' configuration, test if cookies are encrypted before writing to user's machine, test session cookies, check effect on application if cookies are deleted*

Usability Testing

- Test for navigation - *test how user surfs the webpages, test for friendliness, ensure main menu bar appears on each page for consistency purpose*
- Content checking - *keeps content logical and easy to understand, check for spelling errors, check for colour schemes, test for anchor text links, check position and sizes of images*
- Other user information for user help - *search options, sitemap, help should exist in website*

Interface Testing

- Test for two main interfaces - *web server and apps API*
- Check if all interactions are being executed correctly, check error handling, check if error messages are being sent and display if necessary correctly, check for effect on application if user interrupts, check for effect if connection to web server is reset

Compatibility Testing

- Browser compatibility - *website coding should be browser platform independent, test on different browsers, such as safari, chrome, firefox, opera, internet browser*
- OS compatibility - *test web application on target OS platform, linux*
- Printing compatibility - *ensure fonts, page alignment, page graphics are printed correctly*

Performance Testing

- Web load testing - *test the effect when many users are requesting or accessing the same page, site should handle simultaneous requests, simultaneous connection to DB*

Security Testing

- Test by pasting internal urls to address bar, internal pages should not open without logging in
- Validation and verification through invalid user inputs
- Web directories and files should not be accessible unless given download option
- Test CAPTCHA for automates scripts logins
- All transactions, error messages, security breach attempts should be logged in log files

Testing Tools

Telerik Test Studio

tests for cross browser compatibility, create and maintain automated tests for all types of html applications, allows for synchronization when AJAX is called for AJAX testing, supports, Javascript

invocation and validation directly from code, data driven testing, html popups and browser dialog support

SmartBear TestComplete

automate tests, functional GUI testing, keyword-driven testing, regression testing, data-driven testing, unit testing, white-box testing, object driven testing, manual testing, automated HTML5 application testing, integration with source control systems and Visual Studio, test scripts types, test scripts editor, test scriptsdebugger, test run reporting

Selenium

Testing IDE, test automation, downloadable plugins, complete documentation, easy record and playback, walkthrough tests, debug and set breakpoints

Watir Webdriver

open-source tool for automating web browsers, write tests which are easy to maintain and read, models human's interaction with the webpages, check for results, supports multiple browsers on multiple platforms, lightweight and easy to use

Pylot

test web performance, runs http load tests, generates concurrent load, verify server responses, produce reports with metrics, http and https support, automatic cookie handling, response verification with regular expressions, real-time stats, results reports in graph, GUI or shell mode, cross platform

Testing tools for app layer

The app layer of project COMP2014 is a web application written in HTML5, CSS3, JavaScript and C#. The IDE is Microsoft Visual Studio 2012 Ultimate.

To cover as much testing points as possible and at the same time automate testing with VS, we choose to use tools below for respective testing point:

Load and web performance test

Visual Studio 2012 integrated load and web performance testing framework provides the easiest but robust testing for website developed in VS. MSDN provides comprehensive online documentation including step by step walkthroughs to help get started with the tools.

Backend unit test

For the same reason as web performance test, we choose to use the VS integrated unit testing framework to test the C# code at server side.

Security test

Websecurify is a very easy-to-use and open source tool which automatically identifies web application vulnerabilities by using advanced discovery and fuzzing technologies. It can create simple reports (that can be exported into multiple formats) once ran. The tool is also multilingual and extensible with the add-on support.

Conclusion

The objective of the project was to create a proof of concept for a potential thin client solution for a business. The demonstrator worked as expected, with the automated installer taking care of the necessary drivers and packages, and rebooting into a ready-to-use state for the end user. A prototype of the web home panel was made, which provided rudimentary user account capabilities, as well as the basis for a 'launcher panel' that could be used to replace the desktop.

Going from what we have achieved so far, we could either integrate the specialized apps into our panel to provide a more coherent experience through the use of APIs, or host the apps on our own server and integrate them into each other (using custom modules). On the client side, the straightforward approach at this point would be to further reduce overhead by making packages optional, integrating it into the web panel further, and ensuring that it can be installed and run on a larger number of devices.

A number of problems were, however, identified. To integrate the server and client as well as Chrome OS has done will require a more radical overhaul of the solution design and the rewriting of a lot more code. Jolicloud, which has been working towards a similar solution for a long time, has yet to provide a solution in this area. However well they've optimized the experience after login, a nominal local login is still required, these difficulties expected to also apply to our solution.

Upon reviewing our solution as a whole, it appears that the solution appears to be oriented more for the individual rather than for a business. The most evident issue is the lack of a central control panel, where a single (or subset of) users could manage other user accounts, connected devices, and common and individual files and data. Despite apps such as dropbox possessing user-based sharing facilities, something that could replace the traditional shared network drive has yet to be implemented. By using a pre-existing framework which already has user or file management capabilities, this could partially fixed, although it would still require substantial work on integration and testing.

Stepping back from the project, we were working with a fixed model and target hardware too early on. Before we'd started much of the work, we'd already decided on using Raspberry Pi as our target hardware and implementing a system with one cloud server and multiple thin clients. This did not take into account businesses that would normally have resources partitioned amongst different stores/branches, ones that would require specialized software that would only run on a more powerful machine (and wanted to integrate that into the network), or indeed, businesses which would want a more flexible model supporting a multitude of devices and wanted a solution that could entirely replace their 'legacy' IT infrastructure with all the usual expected capabilities.

Despite achieving the essential tasks and proving that the concept is feasible and works in practice, we could look in other directions to follow on this project. The PoC itself has potential to be further developed into a fully-fledged system, but we should allow the client to be more flexible (not limited to our installation of Arch linux and allowing specialized desktop software as appropriate), greatly expand the services offered by the our custom cloud framework, and integrate the individual apps and home screen to create a more capable product with a more coherent experience.