Project Proposal: GameSwap

Team Details

Team Name:	MSP 14
Tutorial:	Tue 2:30 ATC325
Tutor:	Dr Kaberi Naznin

Background / Problem Description

GameSwap is a gaming store with a hybrid income model. They sell computer games and allow customers to book gaming sessions on site to try a range of games on one of the store's powerful computer setups. Currently, customers can pay for a certain number of hours at the front before being let into the gaming room.

GameSwap currently uses a fully paper-based system to track both their game sales and the hours spent by customers in the gaming room. They are having trouble tracking inventory and often think they have certain games for sale when they are out of stock. Furthermore, they are having increasing difficulty maintaining exactly how much time each customer spends in the gaming room especially as more and more customers come to use it.

GameSwap have commissioned a development team to create a digital solution to track their sales data, allow for better inventory prediction and adhere more strictly to the gaming time they sell to customers.

Scope

To address the main issues above the client has stated two primary objectives the software should fulfil:

Table 1. General Client Objectives

No.	Item	Description
1	Digitally Track Time Spent on Devices	Implement some sort of lease-control mechanism on the computers
2	Track Sales Data	Create reports and maintain a database of records tracking product inventory

To support this goal the following specific requirements also need to be met:

Table 2. Specific Objectives

No.	Item	Description
1	Creation of database	The database will hold membership records and can be queried to present details for the interface
2	Creation of an application layer	The application layer will provide the GUI for the company to interact with the database
3	Creation of network setup to control computer access	Front computer would control leasing system for gaming computers

Deliverables, Justification and Schedule

The physical deliverables will be in the form of an exe file and a web server with an internal webpage to control computers on the internal network. The exe will house the GUI. The server (hardware provided by client) will house the database and the portal for the leasing system which will be run through an internal webpage.

In preparing a list of backlog items, a justification must be provided to rank their necessity to the business and therefore help define the order of tasks to be done in sprints. A list of agreed factors and their justifications is shown in Table 1.

Table 1. Justification of Selection Factors

Factor	Justification
Feature Dependency	Feature dependency is the most important factor in determining priority, especially in the early stages of the project. Since the features developed in sprint 1 are likely to be the most fundamental, everything that comes later is likely to depend on them. This will be used as the primary deciding factor
Business Value	Business value is inherently important, as this is primarily defined by the client and will be what the client looks for when the project is delivered. That said, business value is harder to quantify than dependency leaving it as s secondary deciding factor.
Development Effort	Development effort is a minor consideration to be made when engaging in the sprints. Assuming feature dependency and business value are approximately equal, development effort could be used as a discriminator. In general, the task that takes less effort for equal value will be prioritised.
Risk	Risk is another minor factor which uses the same logic as development effort. All things equal, the item of lower risk will be prioritised.

As noted, the progression of factors in terms of weighted importance is:

Dependency -> Business Value -> Development Effort -> Risk

Timeline was not an included factor. This was on purpose as the agile development method does not place as much importance on a stringent timeline as other methods. Furthermore, it was agreed that feature dependency overlaps with timeline, in that the dependencies loosely define the chronological order in which tasks must be done.

The definition of high-low classification for each factor is summarised in Table 2.

Table 2. Definitions of High-Low Affinity of Each Factor

Factor	High	Med	Low
Feature Dependency	3 or more features require this as a prerequisite	Only 1 or 2 features require this as a prerequisite	No other feature requires this feature as a prerequisite
Business Value	The client has directly specified this feature as a requirement	The feature has not been specifically requested, but it is required to better meet another direct requirement	The client has not specifically requested the feature

Development Effort	Development effort is minimal in hours, implementation knowledge is to a high level	Development effort is average but there is some research or trial and error required to implement	Development effort is high in hours including time required to research implementation
Risk	Feature is low risk: Easily replaced and-or reimplemented, not critical to the function of the program.	Feature is medium risk: Somewhat replaceable, is required for program function but cannot be easily damaged.	Feature is high risk: Irreplaceable and critical to program function.

Using the following factors and justifications a more accurate selection can be made for the most important backlog items. Table 3 captures a summary of each backlog item, a justification for its importance, and a ranking according to the selection factors above.

Table 3. Product Backlog

No.	Item	Feature Dep	Business Value	Dev Effort	Risk	Justification
1	Create Sales Table	High	High	High	Low	Creating the sales table is critical for storing records.
2	Add Sales Record	High	High	High	Med	Adding sales records is critical to the business, all report tracking for sales derives from this.
3	Edit Sales Record	Low	Med	High	Med	Editing a sales record is less critical but important for correcting mistakes.
4	Display Sales Record	Med	High	High	Med	Displaying sales records is critical from a testing and user perspective so that operators can receive visual feedback on whether they have used the system correctly.
5	Create Product Table	High	High	High	Low	Creating the product table is critical for storing records.
6	Add Product Record	High	High	High	Med	Adding products is critical as the business needs to be able to add new games to its inventory.
7	Edit Product Record	Low	Med	High	Med	Editing product records is less critical as products are unlikely to change but useful for correcting errors.
8	Display Product Record	Med	High	High	Med	This is critical to show the user that they have used the system correctly.
9	Low Stock Warning Alert	Low	High	High	High	This warning is critical as it is defined as one of the client requirements that they would like

						to be aware of when they're running out of stock.
10	Weekly Sales Report	Low	Med	Low	Med	Weekly reports are not critical but desired by the client in order to better track sales.
11	Sales Report by Product	Low	Med	Low	Med	Sales by product breakdown are also not critical but useful to have in helping the company see sales trends such as whether they need to buy more of certain game during Christmas.
12	Sales UI	High	High	Med	Low	The sales UI is critical both to executing a sale and to making sure it is entered into the database correctly.
13	Product UI	Med	Med	Med	Low	The product UI is not as critical as it will not be used as much but is useful for adding product records without going into the database.
14	Lease Computer UI (from command computer)	High	High	Low	Low	This is critical as it will provide an interface for the staff to interact with the network page and open certain gaming computers for use.
15	Lease Computer UI (from gaming computers)	Med	Med	Low	Med	This is less critical but extremely useful for keeping to a strict timeline as this will show customers how much time they have left on their given gaming session.
16	Time Finished Popup	Low	Med	High	High	This is not so critical as the computer can simply be logged out at the end of the session, but an extra UI component will make it clear and fair to players when they're time is finished. Could even add a 5-minute leeway to finish up the current game.
17	Overall UI	High	High	Med	Med	This is critical for navigating to the other UI components
18	Login UI	Low	Low	Med	High	This is not critical as the front desk should always be manned, but in a landscape of increasing security, authentication is an important component.
19	Network Setup	High	High	Low	Low	This is critical in ensuring that all the gaming computers communicate well with the main computer. This should be mostly done already as the company

						already caters for LAN multiplayer games, it will be a matter of changing the architecture and command structure.
20	Create Lease Webpage	High	High	Med	Low	This is critical to ensuring that the computers can be logged into remotely (by the control computer) and unlocked to be used by customers for their allocated time.
21	Usage Report for Gaming Computers	Low	Med	Low	Med	Reporting on time spent on given computers is not critical but provides excellent feedback to the company that they can use.
22	Reporting UI	Low	Med	Low	Med	The reporting UI is useful but not critical.
23	Testing	Low	Low	Low	High	Testing is not critical to the company but is a critical step in the development process to ensure that the product works as intended.

Table 4. Item Release Schedule

No.	Item	Dependencies	Business Value	Release Schedule
1	Create Sales Table	-	10	Sprint 1
2	Add Sales Record	1	10	Sprint 1
3	Edit Sales Record	1	6	Sprint 1
4	Display Sales Record	1	8	Sprint 1
5	Create Product Table	-	10	Sprint 1
6	Add Product Record	5	10	Sprint 1
7	Edit Product Record	5	6	Sprint 1
8	Display Product Record	5	8	Sprint 1
9	Low Stock Warning Alert	2, 8	9	Sprint 2
10	Weekly Sales Report	4, 8	7	Sprint 2
11	Sales Report by Product	4, 8	7	Sprint 2
12	Sales UI	17	8	Sprint 1
13	Product UI	17	7	Sprint 1
14	Lease Computer UI (from command computer)	19, 20	10	Sprint 2
15	Lease Computer UI (from gaming computers)	19, 20	8	Sprint 2
16	Time Finished Popup	15	6	Sprint 2
17	Overall UI	-	7	Sprint 1

18	Login UI	17	5	Sprint 2
19	Network Setup	-	10	Sprint 1
20	Create Lease Webpage	19	10	Sprint 1
21	Usage Report for Gaming Computers	15	8	Sprint 2
22	Reporting UI	17	7	Sprint 2
23	Testing	-	-	Sprint 3

High Level Solution

The design is broken down into three layers:

The client layer (application layer), which provides a UI and a visual method of interacting with the system.

The logic layer, where business logic takes place and drives things like calculation and reporting.

The database layer, which houses the raw business data.

Together they create an architecture as follows:

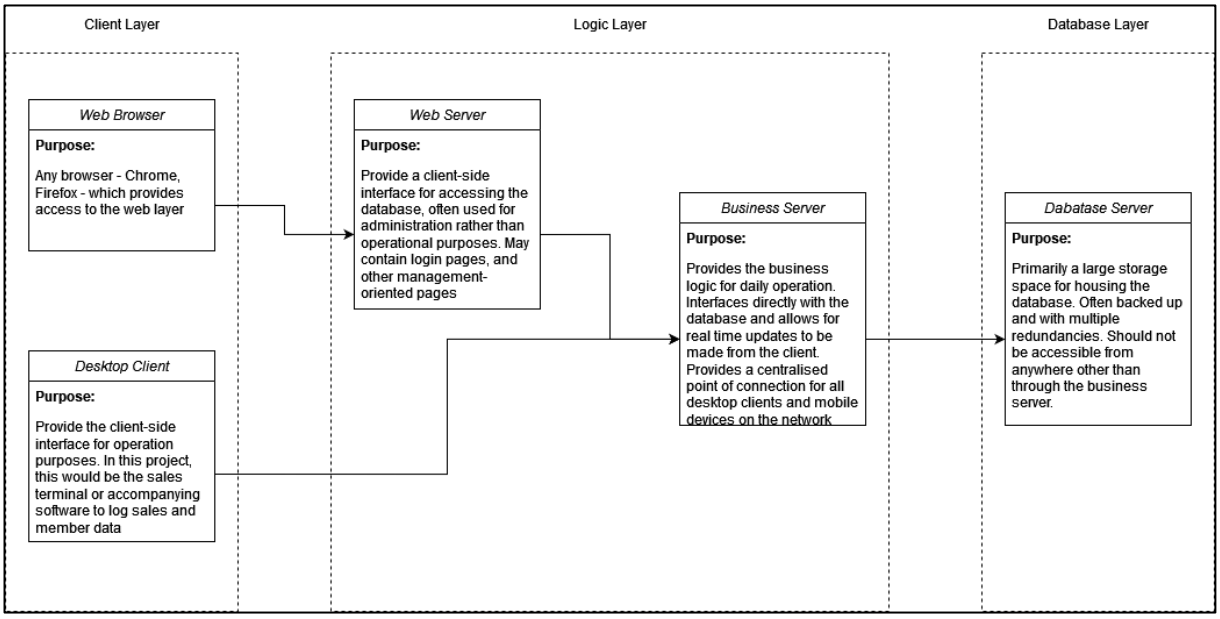

Figure 1: *Overall system design by layer.*

For this project, there the web server component would also house the internal site which handles the leasing of the computers. The high-level design of this page is shown in **Figure 2**.

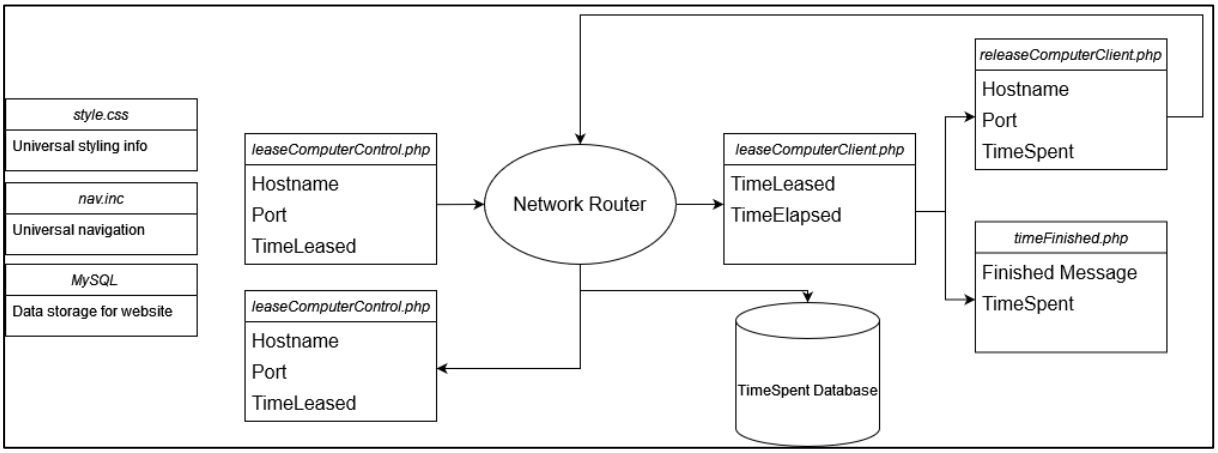

Figure 2: *Leasing webpage high-level functionality.*

Here, some network architecture design is required so that the control computer can send information to the correct computer as designated by the hostname and port. Each client computer will be listening to the network but only allow a login if it has been activated by the control computer. From there, the activate computer will keep a timer of how long it has been in use for and once it reaches the end of its designated time it sends a message back to the control computer asking to be released.

This demonstrates good software management as the control code for activating and deactivating the computers is only stored on the control computer which should be always attended by a staff member. This reduces the chances that a client trying to extend their time could do so on one of the client computers.

Definition of Done Requirements

Quality management on a software product relies on creating quantifiable test benchmarks and ensuring the software performs according or exceeding the criteria. Each item should be developed considering the user/client requirements.

Table 5. Definition of Done

Condition	Description
Functional Suitability	Functional suitability concerns the physical abilities of the software and if they satisfy the requirements. For this project the program must be able to perform the functionality explained, specifically: - Users must be able to add, delete, modify, and search for products using graphical interface. - Users must be able to add sales using graphical interface. - Users must be able to activate computers for a given timeframe using graphical interface. - Computers must automatically shut off after the allocated timeframe. - Users must be alerted when stock on a certain item is low or empty. - Able to run on computers with minimum system specs of 1GB of RAM and intel i3 or equivalent.
Performance Efficiency	Performance efficiency is a measure of how much time it takes for the program to perform key functions. No benchmark was specified by the client, but we can make a reasonable assumption to arrive at the following parameters: - Less than 2 seconds for the program to add a record to the database once the button is pressed. - Less than 2 seconds to display the result when searching for a product.

	- Less than 1 second to add a product to a sale. - Less than 5 seconds to compile and publish a weekly report. - Less than 10 seconds to compile and publish a product sales report over a timeframe of a year.
Compatibility	Compatibility refers to the program's ability to be run on machines with different architecture and operating systems: - No specific platform has been specified by the client. The program should therefore be functional on the 3 most common operating systems: Windows, Linux and Mac. - Does not noticeably slow down other programs running on the target machine.
Usability	Useability concerns the ease of use of the program considering the average training of the user. Though hard to quantify, the following parameters will be used to satisfy this condition: - Users should not be able to enter any incorrectly formatted product details (input checking). - Interface should appear like currently existing POS systems. - Users should be able to use the interface to the same degree as other aspects of their job. Any current GameSwap employee should be able to use the program interface.
Reliability	Reliability refers to the program's ability to perform its function correctly and accurately: - To consider the software successful it should have an internal error (not caused by user error) rate of no higher than 0.5%. - All database data should be preserved in the case of an outage or failure.
Portability	Portability refers to the ability of the software to be used in multiple environments and still perform as expected: - System must completely replace paper-based ordering system.

Software Design

The software design followed for this project (shown in **Table 7**) should adhere to best practice software design principles as shown in **Tables 6**.

Table 6. Software Design Principles

Principle	Description
Modularity	The measure of how well the software is broken down into individual components. The components must be separately maintainable from other components and cannot affect others.
High Cohesion & Low Coupling	**High Cohesion** refers to how a single classes function relate to one another. **Low Coupling** refers to classes having the least number of dependencies on other classes.
DRY (Don't Repeat Yourself)	All pieces of code are not repeated in other elements. For example, a function/definition is only defined once. This helps make maintainable and scalable coding.
MVC Design Pattern	Model View Controller (MVC) is a design principle that breaks down each functional component of an application into three separate components. These three components describe the interaction between a user and the backend of an application.

	A User uses the Controller which in turn manipulates the Model and updates the View that is seen by the User. This is essentially a feedback loop for user input and data manipulation.

Table 2. Justification of Design Principles

Principle	Justification
Modularity	Each UI component within the project is separate from another. Changing anything on the products screen cannot change or affect the performance of any other component. All product functions are defined on the product page and the same goes for the sales page. The leasing system, though accessed through the same UI is completely independent, with all code running on the business server. This protects it from being affected by anything stored and run on the client software.
High Cohesion & Low Coupling	Each object within this project is designed to perform a single action. Where data needs to be transferred between two objects, this is done so with a bridging function which is again, modularly built so that if edits need to be made to the data transmission that is the only function that needs to be consulted and not the functions generating or receiving the data. An example of this would be the php scripts calling for the computers to be leased and released. The control computer is the only one with the power to do this, while the client computers only keep track of how long their session has been engaged for and sent back a release request when time is up.
DRY (Don't Repeat Yourself)	This goes without saying. There not many places where this would be particularly relevant as the product entry UI and the sales entry UI are distinctly different SQL processes.
MVC Design Pattern	To follow this model well, the sales UI page should refresh after entering a new sale in order to show the user and client that is has gone through successfully. On the leasing side, the computers should have an unclosable popup active on the desktop which shows them how much time has passed. This is important for the MVC pattern, but also for ensuring that clients cannot be confused with how long they have spent in their session.

Sprint 1 Planning

The tasks that must be carried out for Sprint 1 are listed in **Table 8** with their time estimation method. Primarily these are the fundamental tasks upon which the higher-level UI components will be built. This follows the justification to use feature dependency as the primary determination in backlog order.

Table 8. Sprint 1 Backlog Items

Item	Time Estimated (hours)	Time Estimation Method
Create Sales Table	0.2	Delphi
Add Sales Record	0.8	Analogy
Edit Sales Record	1	Analogy
Display Sales Record	0.3	Size Estimation
Create Product Table	0.2	Analogy
Add Product Record	1.3	Analogy
Edit Product Record	0.2	Analogy
Display Product Record	0.3	Analogy
Sales UI	2.1	WBS
Product UI	2.2	WBS
Network Setup	5	Analogy
Create Lease Webpage	7.5	Size Estimation

Create Sales Table

The sales table creation requires exactly 1 SQL statement of relative simplicity. Using any common database manager (MySQL, PostgreSQL) this can be done graphically and even quicker. Using a Delphi estimation, it will take no longer than 20 minutes.

Add Sales Record

Adding a sales record is more complex as it requires 1 SQL statement as well as some processing code to read from an input field and translate this into an SQL statement. By analogy of similar statements, the SQL statement should take no more than 10 minutes to create. On the other hand, processing code depending on the UI software and scales with how many input fields there are. Fortunately, the POS system requires little error checking as every sale is done via barcode scanning. The only information that needs to be read into the table is the items involved and their quantities, all values that are hardcoded in the product table anyway. It also requires a timestamp which will be autogenerated correctly from the system.

With all this in mind, the code required to process a sale into a record should not take longer than 30 minutes, making this task take 40 minutes in total.

Edit Sales Record

Editing a sales record follows the same process as entering one, however it is not as safe as the sale is not being generated by barcode, so error checking is required. Furthermore, to edit a record, it has to be searched for and found in the sales table, which requires an additional SQL statement to retrieve the records. Two SQL statements will take 20 minutes.

Error checking on each field can be considered by analogy – having done a similar task – to take approximately 10 minutes per field. Since there are 4 potential fields to be edited, this would take 40 minutes, making this task take 1 hour in total.

Display Sales Records

Displaying a sales record requires a SELECT statement and a form of processing to populate a UI field. The SQL statement will take less than 10 minutes. Given the processing does not have to change or error check anything – as each record is assumed to be correctly formatted, it will take less time than going the other way (from UI to database). Using size-estimation against "Add Sales Record" this task should take a third of the coding time, yielding 10 minutes. This task will therefore take 20 minutes in total.

Create product Table

This task follows the basics of the "Create Sales Table" task. By analogy it will also take no longer than 20 minutes.

Add Product Record

Adding a product record is similar to adding a sales record but more complex again as there is no barcode to scan. The details of each product must be entered manually meaning that error checking has to be more stringent. The product table will have 4 columns, meaning that 4 fields need to be error checked. Using size estimation and analogy, this means the error checking alone will take 40 minutes. Adding the 30 minutes estimated for the processing code of "Adding Sales Product" (code will be similar), and adding 10 minutes for another SQL statement, this task will take a total of 1.3 hours.

Edit Product Record

Editing a product record is almost identical to editing a sales record, however the difference is that the code for error checking each field would have already been completed in the "Add Product Record" task. Time does not have to be allocated for this. Therefore it is as simple as an extra SQL statement to select the record in question, making this task (excluding UI) a 10 minute job.

Display Product Records

Displaying a product record will take the same time and effort as the "Display Sales Record": 20 minutes.

Sales UI

The sales UI is composed of multiple items which will appear on screen for the user to click and navigate through. The WBS is used to try and capture an adequate estimation for how long it will take. Most of the UI components are dependent on the software being used in how long it will take to create them. For this project, it is assumed that the Ignition software (used for the Goto-Gro project) will also be used here. That way, size estimation and analogy methods can be best utilised.

Table 9. Sales UI WBS

Item	Description	Justification	Est. Time
Overall UI and Navigation	The physical screen all the buttons are displayed on. Offering tabs at the top to alternate between screens.	The display page is simply a panel with a strip of 4 tabs at the top. It will be trivial to build.	0.1
Display Panel for Running Sale	The panel on the side where the scanned product data will be echoed back to the screen.	The display panel will draw from the "Add Sale" code and display it. Some designing will be required to	0.5

		show the data in an easily readable format.	
Confirm Sale ("Pay") Button **Card**	A button to trigger the payment popup using card.	A simple button which triggers a code event. Trivial in ignition.	0.1
Confirm Sale ("Pay") Button **Cash**	A button to trigger the payment popup using cash.	A simple button which triggers a code event. Trivial in ignition.	0.1
Link to POS Terminal for **Card** Payment	Code to activate the POS machine.	The linking code to make a POS terminal work requires more effort. Research shows there are resources which already do this, reducing the time needed to code it, however this is a technology the team may not have exposure to, a conservative estimate would take the upper bound of what is reasonable.	1
Popup to Enter Amount Paid for **Cash**	Code to deduct the price of the sale from the amount entered and calculate required change. Would also trigger opening the till and keep a daily record of how much cash should be in there by the end of the day.	This code is very simple, requiring a basic mathematical calculation and a popup to confirm whether or not this amount has been paid to the customer.	0.3
Total			**2.1**

Product UI

The product UI is composed of multiple items which will appear on screen for the user to click and navigate through. The WBS is used to try and capture an adequate estimation for how long it will take.

Table 9. Product UI WBS

Item	Description	Justification	Est. Time
Overall UI and Navigation	The physical screen all the buttons are displayed on. Offering tabs at the top to alternate between screens.	The display page is simply a panel with a strip of 4 tabs at the top. It will be trivial to build.	0.1
Display Panel for Current Products Table	The panel on where alphabetical product records will be displayed.	Some designing will be required to show the data in an easily readable format.	0.5
Entry Field Title	An input field.	A simple input field. Trivial in ignition.	0.1
Entry Field Category	An input field.	A simple input field. Trivial in ignition.	0.1
Entry Field Price	An input field.	A simple input field. Trivial in ignition.	0.1

Entry Field Series	An input field.	A simple input field. Trivial in ignition.	0.1
Confirm Entry Button	A button to push the field data into the database.	A simple button. Trivial in ignition.	0.1
Search Button	A button to activate a table search.	A simple button. Trivial in ignition.	0.1
Search Input Field	A small field for typing the name of the item to search.	A simple input field. Trivial in ignition.	0.1
Select a Record from View Panel	Functionality to click on a record in view panel and highlight it, populating the input fields.	Requires a bit of coding. By analogy, have done a similar task.	0.5
Delete Button	A button to delete selected record from table.	A simple button. Trivial in ignition.	0.1
Delete Confirm Popup	Confirm popup with two buttons saying cancel or okay to confirm delete action.	A more complex popup requiring multiple buttons and snippets of trigger code.	0.3
Total			**2.2**

Network Setup

The network setup is not strictly a coding task, but it is necessary to ensure the smooth operation of the product. Having experience with setting up networks, this may require access to the router on the company floor, as well as access to each computer on the network in order to set up remote login and/or group policy implementations.

Fortunately, given that the company is already running and that they offer LAN games, it can be safely assumed that the network is already set up, leaving the only complexity in setting up appropriate group policies in preparation for remotely logging in and logging out of the computers using the control computer. On a single computer this is not an easy processa dn requires multiple restarts. A conservative estimate would be 15 minutes per computer for an experienced operator. The company has 20 gaming computers so this would take 5 hours in total.

Create Lease Webpage

The lease webpage would be very similar to something like the networking router lease portal that Swinburne uses (smartrack). This is a single page but has a lot of embedded functions and dynamic code to update the page when a new rack is booked. Following this kind of functionality, the page would need to display a list of all the computers connected to the network and their status – green for available, red for occupied, yellow for out of order. Ideally these are displayed in some sort of geographic order so that their positions correspond to their positions in the gaming room to enhance usability. Then after selecting given computer the user would have to give a time in hours for which it is to be in use and confirm this with a button. Finally, some JavaScript function would have to send that code as a packet to the correct computer (taking its IP and port number from the graphical selection made) and activating the page on the computer in question. After this, some code would force a remote login and open the computer for lease. The script keeping track of how much time this session has been active for will also start at the same time.

To quantify how long this might take we can look at how many pages need to be implemented. There is one visual page requiring html, css, php, headers etc, and the visual componenet

alone would take up to 2 hours, with the graphical layout of the computers taking up a lot of this time. The code for this is not necessarily complex, but engages networking components and must be robust, at a minimum it would take approximately 2 hours as well. Then there needs to be a php listener on each of the gaming computers (but no visual components), which would be a small snippet of repeated code which also has to fire back that the computer is in use to the control webpage. This should take 1 hour. The graphical timer on the computers would take about 1 hour, using analogy of doing a similar display program on windows. A final component would be the deployment of the interface and listener on each client computer which at 5 mins per computer would take 1.5 hours.

Conclusion

The task presented by GameSwap involves a hybrid architecture of using a client program (exe) and a webserver with business code to track their sales and allow for strict digital control over their leased computers. This document presents the project planning, high level software design, and management plan for Sprint 1.