|  |
| --- |
|  |
| Network Computers Manager |
| פרויקט גמר בסייבר |
|  |
| **שם התלמיד: אריאל וסטפריד**  **ת"ז: 209368570**  **מנחים: מיכאל צ'רנובילסקי וגד רוזנטל**  **בית ספר: אורט גוטמן נתניה**  **כיתה: י"ב 8** |
|  |

**‏מאי 16**

|  |
| --- |
|  |

# תוכן עניינים

[תוכן עניינים 1](#_Toc451269529)

[רקע 2](#_Toc451269530)

[תיאור המוצר המוגמר 5](#_Toc451269531)

[מבט אישי על העבודה ותהליך הפיתוח 7](#_Toc451269532)

[תיאור הממשק למשתמש 9](#_Toc451269533)

[סביבת העבודה 14](#_Toc451269534)

[תיאור המודולים 15](#_Toc451269535)

[האלגוריתמים המרכזיים בפרוייקט 17](#_Toc451269536)

[תדפיס של הקוד 19](#_Toc451269537)

# רקע

**תקציר ורציונל הפרויקט**

תארו שאתם אחראיים על התפקוד של כל המחשבים ברשת אחת. בדרך כלל רשת כזו מונה מאות מחשבים, ואם יש בעיה תפקידכם הוא להגיע אל המחשב הספציפי ולטפל בו. בעיה יכולה להיות דבר פשוט בקבצים או מחשב שלא נדלק. האם באמת צריך לעבור ממחשב למחשב כל הזמן כדי לפתור בעיות ולפקח על פעילות המחשבים?

פריוקט זה מנסה לתת פתרון נוח עבור ניהול מחשבים ברשת על ידי מתן כלים לניהול מרחוק. בזמן שבו יש מחשבים בכל מקום ועיקר העבודה מתבצעת מהם יש צורך לפקח עליהם. כך, המנהל יכול לראות אילו מחשבים פועלים, אילו תוכנות רצות וגם לעבור על עץ הקבצים. כך מנהלים יכולים לראות מה קורה במשרד או הטכנאי יכול לעבור על המחשבים.

לדוגמה: אחד העובדים לא כיבה את המחשב בסוף יום העבודה. התוכנה מספקת אפשרות לכיבוי אוטומטי של מחשב או כמה בזמן רצוי.

דוגמה נוספת: ישנה פגיעה במערכת הקלט של אחד המחשבים. המערכת מאפשרת גישה אליו מרחוק כדי לבצע פעולות אחרונות (לדוגמה להוריד ממנו קובץ) לפני כיבוי לשם תיקון.

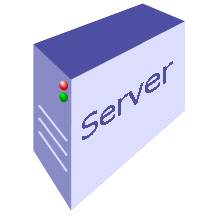
**מבוא ורקע כללי**

הרעיון העיקרי של הפרוייקט הוא מתן גישה נוחה מרחוק לשאר המחשבים ברשת. לכן, הפרויקט צריך גישה למחשבים דרך הרשת ולהצליח להשפיע על פעולותיו. כדי לעשות זו יש לפרויקט כמה מרכיבים עיקיים:

* מרכיב עיקרי בפרויקט שלי הוא פרוטוקול Wake on LAN. Wake on LAN, אחד מהתכונות העתיקות ביותר במערכות ההפעלה, המכונה לעיתים כ"השכמה מרחוק", היא טכנולוגיה המאפשרת להפעיל מחשב ברשת מרחוק על-ידי שליחת מנת נתונים מיוחדת (בנקראת מנת Magic). בעזרת התאמה בהגדרות, גם אם המחשב כבוי, מתאם הרשת עדיין "יאזין" לרשת, כך שכאשר המנה המיוחדת מגיעה, מתאם הרשת יכול להפעיל את המחשב.

מבנה של Magic Packet

* מרכיב נוסף חשוב בפרויקט שלי הוא תקשורת. תקשורת בעולם המחשבים מבוססת על Socket, שהם נקודות קצה עבור זרם נתונים בתקשורת בין תהליכים על גבי רשת מחשבים. רוב הסוקטים כיום מבוססים על ה – Internet Protocol (IP). בפרויקט שלי התקשורת בין השרת ללקוחות מתבצעת באמצעות Internet Socket, כאשר בשכבת התעבורה יש שימוש בפרוטוקול ה-TCP, שייחודו בכך שהוא מוודא שכל הנתונים שנשלחו התקבלו ובסדר הנכון, ואחרת שולח אותם שוב. בפרויקט שלי יש חשיבות לכך שלא יהיה איבוד מידע ופרוטוקול ה-TCP/IP מתאים בדיוק למטרה זו.
* מרכיב שלישי בעל חשיבות לפרוייקט הוא היכולת להתעסק עם מערכת ההפעלה. כך ניתנת לתוכנה גישה לקבצים, לתהליכים וכדומה. לשם כך אני משתמש ב – Windows API. ה-Windows API הוא סט כלים תכנותיים אשר נותנים למתכנתים גישה למערכת ההפעלה. כך, הפרויקט יכול לגשת דרך מערכת ההפעלה לקבצים ולתוכניות.
* מרכיב נוסף בפרויקט הוא שימוש ב-GUI שאפשר להתחבר אליו דרך הדפדפן. כך, אפשר לגשת לשרת מכל מחשב ברשת על ידי הדפדן. דבר זה ניתן על ידי מודול Flask אשר פותח שרת HTTP במחשב ומטפל בתקשורת בין השרת ללקוח (הדפדפן). השרת פונה לקוד שנכתב על ידי המשתמש בעת בקשת דף ומחזיר את מה שמחזיר הקוד.
* בנוסף, הפרויקט משלב עבודה עם מסד נתונים. הנתונים שהתוכנה אוספת לגבי כל אחד ממחשבי הרשת נשמרים במסד הנתונים, שהעבודה איתו היא בטכנולוגיית ה – SQLite, שמממשת את רוב הסטנדרט של SQL (שהיא השפה המקובלת לטיפול ועיבוד מידע בבסיסי נתונים). להבדיל מרוב המערכות לניהול בסיסי נתונים, SQLite אינה תהליך עצמאי נפרד, אלא ספריה בתוך התהליך באופן אינטגרלי. זוהי בחירה פופולרית כאשר מפתחים בסיס נתונים מוטמע עבור צד לקוח כשירות ניהול מידע מקומי. מסד הנתונים שאותו מנהלת התוכנה שלי שומר מידע לשימוש מקומי בלבד, ולכן בחרתי להשתמש ב-SQLite שלא דורש הרצה של SQL Server.



דפדפן

שרת HTTP

שרת ראשי

עם מסד נתונים

לקוחות

לקוחות

לקוחות

תאור הרשת בפרויקט ותקשורת בין חלקיה

# תיאור המוצר המוגמר

שם המוצר הוא Network Computers Manager.

ביכולתו להדליק ולכבות מחשבים מרחוק (בתוך הרשת) לפי זמן שנקבע מראש. על מחשבים בהם פועלת תוכנת הלקוח אפשר לבצע פעולות נוספות: גישה לקבצים, מחיקתם או יצירת חדשים, גישה לתהליכים עצירתם או הפעלת תהליכים אחרים. כל זאת כאשר ממשק המשתמש של התוכנה נפתח בדפדפן וכך, יכול להפתח מכל מחשב ברשת בעזרת הדפדפן, כל עוד השרת עובד באחד המחשבים ברשת.

המטרות המרכזיות של ה-Network Manager Tool הן הקלת העבודה על מנהל הרשת. הוא יכול לפקח על המחשבים ופעולותיהם מרחוק ואין לא צורך לגשת אליהם פיזית. בנוסף, הממשק מאפשר גישה לשרת מכל מחשב ברשת דרך הדפדפן כך שגם אם מנהל הרשת לא ליד העמדה שמחוברת למחשב השרת, כל עוד עמדתו מחוברת לרשת הוא יכול לגשת אליה. כל זה קורה תוך תקשורת מאובטחת בין השרת והלקוחות.

שאלות שהמערכת עונה עליהן:

**כיצד ימצאו תוכנות הלקוח את השרת?** כל כמה שניות, השרת שולח הודעת Broadcast שכוללת את המפתח הפומבי שלו ו-Hash של הודעה שנקבע מראש. תוכנת הלקוח שהופעלה זה עתה תאזין לקשורת ותאתר את ההודע הזו, לאחר אימות של המסר שנקבע מראש, יתחיל הלקוח בתהליך החיבור לשרת.

**כיצד להפעיל מרחוק מחשב?** קיימת אפשרות להפעיל מחשב מרחוק דרך כרטיס הרשת. יש לסדר בהגדרות המחשב שגם כאשר הוא כבוי, כרטיס הרשת ישתמש במעט חשמל כדי להאזין לתקשורת . כאשר כרטיס הרשת מוצא Magic Packet שמיועדת לו, הוא יפעיל את המחשב.

**כיצד לגשת לקבצים ולתהליכים של מחשב מרוחק?** דבר זה נעשה בעזרת תקשורת שרת – לקוח. השרת שולח ללקוח שלו את הבקשה שלו (לדוגמה: מחיקת קובץ ספציפי) והלקוח בעזרת הגישה ש-Windows API מאפשר לו, מבצע אצ הפעולה ומחזיר תשובה לשרת.

השוואת העבודה עם פתרונות ויישומים קיימים:

ישנן תוכנות המבצעות Wake on LAN וגם תוכנות דרכן אפשר לפקח מרחוק על המחשבים ברשת. לא נמצאה תוכנה שלממשקה אפשר לגשת דרך הדפדפן, כאמור דבר המאפשר נוחות וגישה לשרת גם מהמחשבים שלא מריצים אותו עליהם. בנוסף, התקשורת בין השרת והלקוח מאובטחת ולכן אין חשש שגורם זר ישפיע על מחשבים ברשת. לעומת זאת בתוכנה הנוכחית, אין פיקוח על דברים הקשורים בחומרה כמו פיקוח על טמפרטורת מעבד וצריכת משאבי מערכת. בנוסף אין תמיכה באותיות עבריות במערכת הקבצים והתהליכים.

# מבט אישי על העבודה ותהליך הפיתוח

במהלך העבודה עמדתי לפני כמה אתגרים והיו לי כמה מחקרים לעשות:

* תחילה, להבין כיצד להפעיל מחשב מרחוק. למדתי שקיים פרוטוקול Wake on LAN שאני יכול להיעזר בו. למדתי שיש צורך לאפשר הגדרות מסוימות ב-BIOS של המחשב ובהגדרות מערכת ההפעלה כדי לאפשר לפרוטוקול לעבוד ואכן להפעיל את המחשב.
* שנית היה עלי להבין כיצד אני יכול לדעת שמחשב היעד אכן קיים וכיצד למצוא אותו. לאחר חיפוש באינטרנט וקריאה של חומר, הבנתי שכל עוד המחשב כבוי, אי אפשר לדעת על קיומו. כדי לפתור את הבעיה היה עליי למצוא את המחשב כאשר הוא פועל ולשמור במאגר נתונים את קיומו. כך אדע שהמחשב קיים ואוכל ליידע את המשתמש על כך גם כאשר הוא לא פועל ולא נמצא ברשת.
* לאחר מכן היה עליי להבין כיצד למצוא מחשבים ברשת. על כך היה עליי להשתמש בפרוטוקול ARP – Address Resolution Protocol. התוכנה עוברת על כל הכתובות ה-IP האפשריות ברשת ושלחת ב-broadcast שאלה למי יש את הכתובת. מחשב בעל הכתובת מחזיר פקטת תשובה עם כתובת ה-MAC שלו. כך המחשב מקבל כתובות IP ו-MAC של כל ישויות הקיימות ברשת.

ארוע מעניין במהלך העבודה שהוסיף לאתגר היה ההצעה של גד, עוזר ההוראה, לפתח GUI בממשק web שיפתח בדפדפן. כך שמעתי על הספריה Flask ולמדתי להשתמש בה לצורך פיתוח ממשק משתמש. בנוסף גם למדתי מעט HTML ו-CSS לשם שיפור עיצוב הדף.

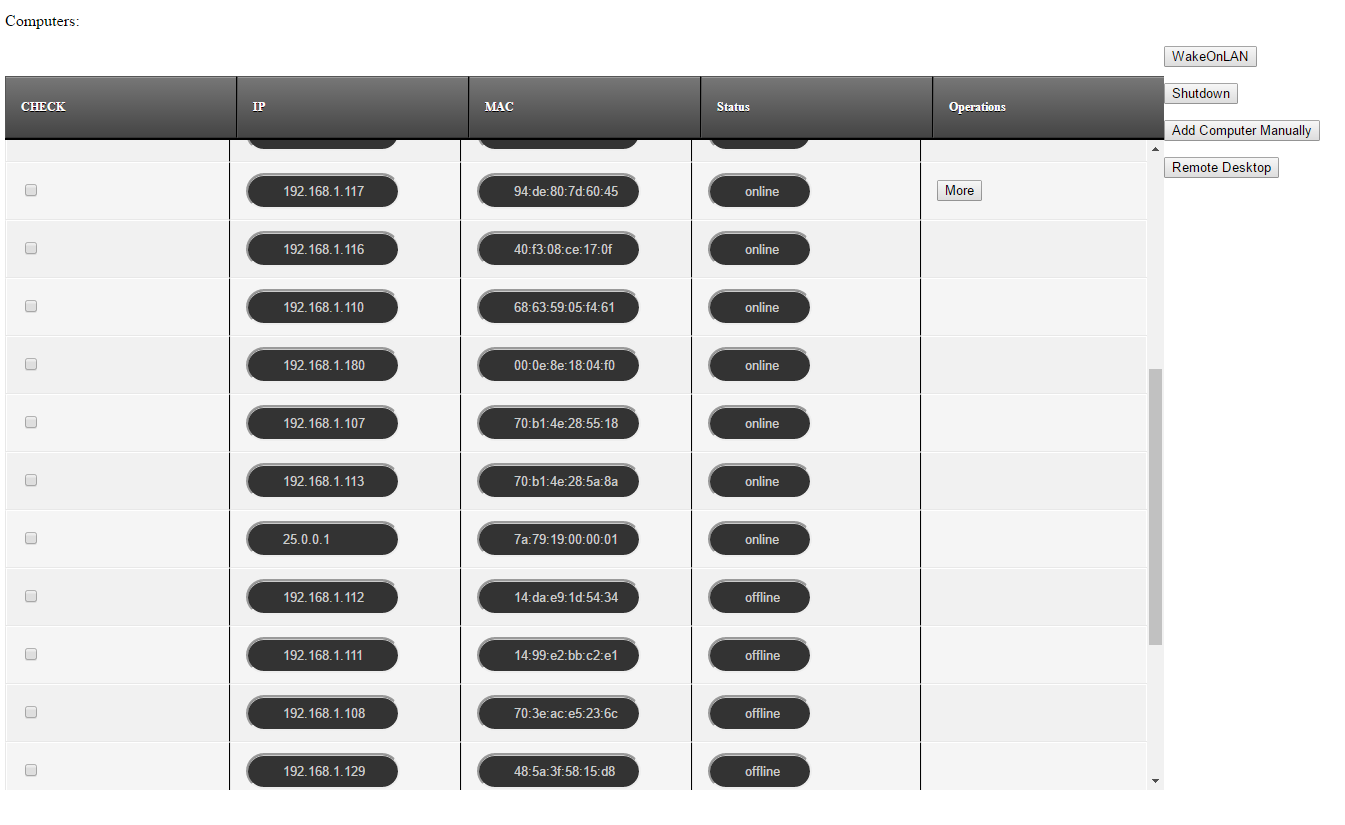
דבר החסר בפרוייקט והייתי רוצה להוסיף לו היה לי עוד זמן הוא התקנה יותר קלה של הפרויקט. התחלתי לחקור בנושא אך לא נותר לי די זמן. רציתי להקל לרמה שכל הצורך יהיה בהתקנת תוכנת השרת בלבד והוא ידאג לשאר – התקנת הלקוחות, הלקוחות יכניסו עצמם ל-Registry כדי שיופעלו עם הפעלת המחשב ואף בעצמם יטפלו ב-BIOS כדי לסדר את ההגדרות לתמיכה ב-Wake on LAN.

אני רוצה להודות למיכאל צ'רנובילסקי ולגד רוזנטל על העזרה במהלך הפרויקט, והזמן שהשיקעו בי ומתן עזרה.

# תיאור הממשק למשתמש

תחילה, יש להתקין את תוכנת השרת על מחשב אחד ברשת (לפחות) ואת תוכנת הלקוח על כל המחשבים.

כדי לאפשר הדלקת מחשבים מרחוק, בכל מחשב יש לאפשר ב-BIOS את האפשרות ל-Wake on LAN או Wake on magic packet תחת הקטגוריה Power management.

לאחר שכל אלו נעשו, יש להפעיל את השרת. מייד יפתח חלון חדש או לשונית חדשה בדפדפן שתציג את ההודעה: Server is starting. It might take a while…. יש לחקות עד לעליית השרת.

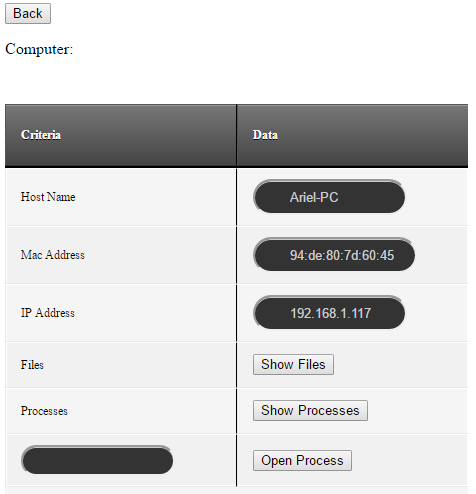
החלון הראשי:

מציג טבלה של כל המחשבים ברשת שהשרת יודע על קיומם. אפשר לבחור מהם מחשבים ולהפעיל אותם על ידי לחיצה על כפתוח Wake on LAN שיביא למסך הזה:



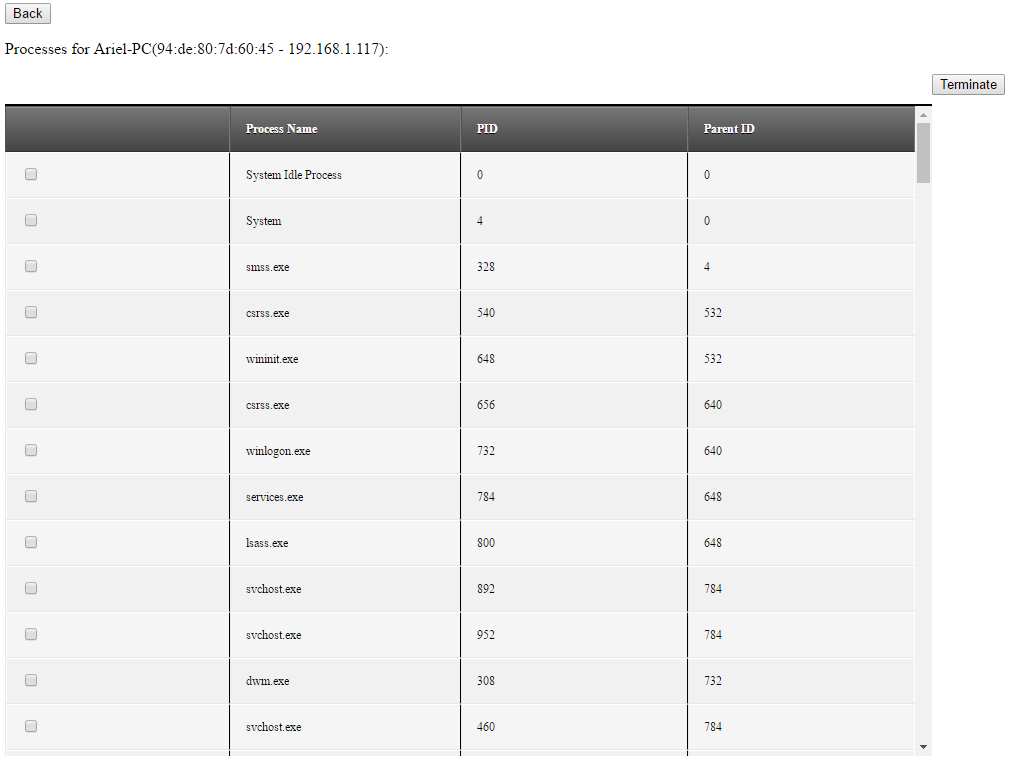
כאן אפשר לבחור אם להפעיל את המחשב בזמן ספציפי או מייד ולבחור את הזמן. אפשר גם לעשות שפעולה זו תחזור על עצמה על יום. לחיצה על Shutdown במסך הראשי תביא לחלון זהה.

ליד מחשבים שעליהם מופעלת תוכנת הלקוח יופיע כפתוח More, לחיצה עליו תביא למסך:



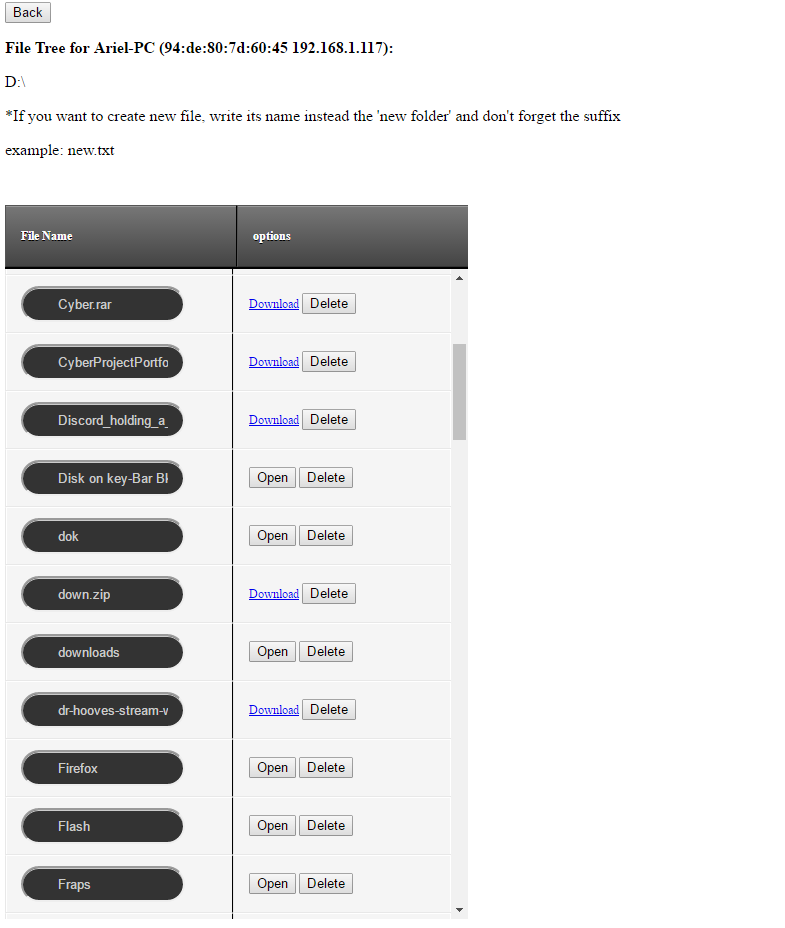
מסך זה מציג מידע על המחשב הספציפי הזה. כאן אפשר לכתוב פקודה שתרוץ בshell של המחשב ולשלוח אותה עם לחיצה על Open Process, אפשר לגשת לחלון התהליכים וחלון הקבצים על ידי לחיצה על Show Processes ו-Show Files בהתאם.

חלון תהליכים:

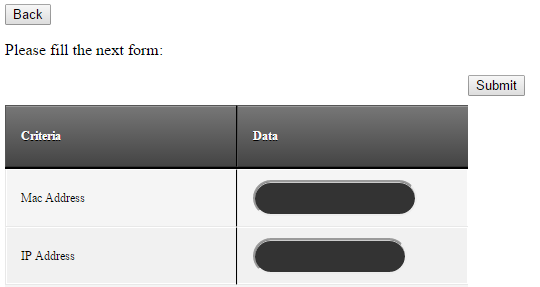


מציג את כל התהליכים הרצים על המחשב. אפשר לבחור מתוכם ואז ללחוץ על Terminate לסגירתם המיידית.

חלון הקבצים:

מציג את התוכלה של תיקיות. לחיצה על הכפתוח Open תציג את תכולת התיקייה. לחיצה על Delete תמחק את הקובץ או התיקייה לאלתר.

אם רוצים להוסיף לרשימה מחשב באופן ידני יש ללחוץ על הכפתור “Add Computer Manually” שיובל למסך הבא:



אשר בו יש להזין את פרטי המחשב וללחוץ "Submit".

כדי לגשת לממשק המשתמש דרך הדפדפן יש לכתוב את כתובת ה-IP של המחשב עליו רץ השרת, נקודותיים ואז-5000. לדוגמה:



# סביבת העבודה

הפרויקט נכתב ב-Python, ב- HTML וב-CSS. ליבת הפרויקט, ועיקרו נכתבו בשפה Python בשל הנוחות של השפה. ה-GUI נכתב ב-Python, ב-HTML וב-CSS. HTML ו-CSS שימשו לעיצוב ה-GUI עצמו כדי שיהיה אפשר להציג אותו בקלות ונוחות בדפדפן. הקוד שמחבר בין ה-GUI לתוכנה נכתב ב-Python. כדי לחבר בין הקוד שנכתב ב-Python לדפדפן שמשתמש בפרוטוקול HTTP, השתמשתי בספריה שהוכנה עבור Python בשם Flask. זו יוצרת שרת HTTP, מכסה את התקשורת איתו ומעניקה למתכנת כלים כדי להציג בקלות דפי HTML שהוא עיצב.

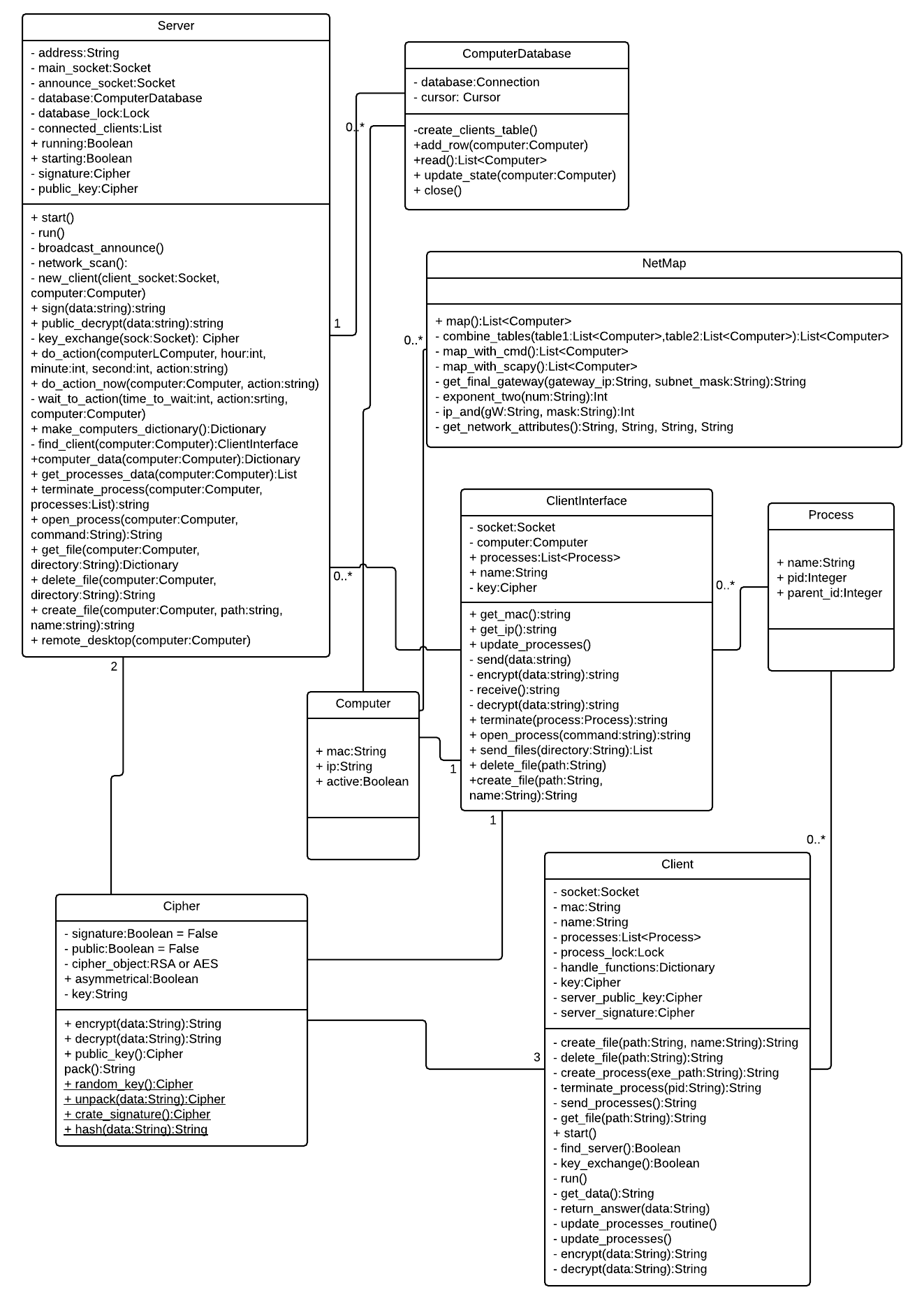
סביבת העבודה בה השתמשתי עבור שלוש השפות היא PyCharm. היא תוכנה מקצועית ונוחה לעבודה על פרויקטים מרובי קבצים ומאפשרת גם הרצה ו-debugging של הסקריפטים.



# תיאור המודולים

המודולים המרכיבים את הפרוייקט והקשרים ביניהם:

* מודול GuiEngine. מודול זה מפעיל את ממשק המשתמש ואת מנוע השרת ומהווה את החוליה המקשרת ביניהם.
* מודול Server מהווה את תוכנת השרת העיקרית. הוא מופעל על ידי מודול GuiEngine ומתופעל חלקית על ידיו. מודול זה משתמש במודולים הבאים:
  + מודול ComputerDatabase שמהווה את הקישור של השרת למאגר הנתונים שלו.
  + מודול NetMap שתפקידו מיפוי הרשת ומציאת כל היישויות המחוברות בה.
  + מודול ClientInterface שמכסה את הממשק של השרת לקשר עם הלקוחות.
  + מודל Cipher שאחראי על ההצפנה ואבטחת המידע.
* מודול ClientInterface משתמש במודולים: WakeOnLan שמכסה את הפעלת המחשבים וכיבויים מרחוק, ובמודול Process שמכיל אובייקט לייצוג תהליך.
* מודול ComputerDatabase משתמש באובייקט Computer לייצוג מחשב ממודול ClientInterface.
* מודול Client מהוות את תוכנת הלקוח. הוא משתמש במודולים:
  + Process שמכיל את האובייקט Process.
  + Cipher
* כל המודולים משתמשים במודול Constants שמכיל את הקבועים של התוכנה.



תרשים UML של המחלקות המרכיבות את הפרויקט

# האלגוריתמים המרכזיים בפרוייקט

בין הבעיות בהן התקלתי במהלך הפרויקט:

* מימוש שרת מולטי-קליינט:

הבעיה הראשון שעמדה לפניי בפיתוח המוצר היא יצירת שרת מולטי קליינט. כלומר, הייתי צריך לממש שרת שמתחברים אליו מספר לקוחות בו זמנית והוא מסוגל לדבר עם כולם. יש צורך ביצירת פרוטוקול תקשורת שיגדיר את אופי ההתנהלות של התקשורת. עליי גם לוודא שהמידע יהיה מאובטח ולהצפין אותו וכך לוודא שהמידע יעבור במלואו בבירור למרות ההצפנה.

פתרון: פיתחתי פרוטוקול פשוט משלי בו קיימות כמה הודעות אשר השרת והלקוח מכירים ואיתן הם מתקשרים ומעבירים ביניהם את המידע. כדי להצפין באופן יעיל את המידע, תחילה יש העברה של מפתחות אסימטריים של RSA ובעזרתם מועבר מפתח סימטרי של AES. שאר התקשורת מתבצעת בהצפנת AES כיוון שהצפנה סימטרית דורשת פחות משאבי מערכת מאשר הצפנה אסימטרית. כדי שלקוחות ימצאו את השרת, כל פרק זמן של 10 שניות, השרת מודיע ב-broadcast לכל הרשת על קיומו. כך לקוח שמופעל מאזין לרשת ויודע היכן נמצא השרת ומחבר אליו. דוגמת שלבי התקשורת בין השרת והלקוח:

I am Server

Connection Request

Public RSA Key

RSA( AES Key ) + Hash( AES Key )

Terminate Process

OK

שלב Broadcast

שלב Key Exchange

תקשורת בין שרת ולקוח



לקוח

שרת

* כיבוי והדלקה מרחוק של מחשבים

עליי לדעת כיצד להפעיל ולכבות מחשבים דרך הרשת בעזרת Python ללא בעיות.

פתרון: כדי לשלוח Magic Packet יש להשתמש ב-Socket אשר שולח הודעה ב-Broadcast וללא אף פרוטוקול שיכסה את הפקטה. לאחר התאמת ההגדרות לאובייקט ה-Socket אפשר לכתוב לתוכו את תוכן ה-Magic Packet ולשלוח.  
כדי להפעיל מחשב מרחוק אני משתמש ב-API של Windows. המחשב יכול לקבל הרשאות ובעזרתן לכבות דרך הרשת מחשבי Windows אחרים.

* קבלת גישה לתהליכים ולקבצים של המחשב

כיצד התוכנה שלי, שנכתבה ב-Python ומופעלת על ידי מערכת ההפעלה יכולה לקחת תפקידים של מערכת ההפעלה? כלומר, על התוכנה לסייר בין הקבצים, ליצור חדשים ולמחוק קיימים. התוכנה צריכה גם להיות מסוגלת להפעיל ולסגור תהליכים הרצים על המחשב.

פתרון: התוכנה משתמשת ב-API של Windows שהוא אוסף כלים תכנותיים שאפשר לתכנת בעזרתם גם עם שימוש במערכת ההפעה. ל-Python נכתבה ספרייה Win32API אשר מנגישה את ה-Windows API גם ל-Python. בעזרת אלו אני יכול לגשת למערכת ההפעלה.

* שמירת מאגר מחשבים של הרשת

כיצד התוכנה תוכל לזכור את קיומם של מחשבים שלא מחוברים לרשת. אם לדוגמה, יפול החמשל והשרת יאלץ לרוץ מחדש, כיצד הוא יוכל לזכור את כל המחשבים שהכיר בריצה הקודמת ללא סריקה מחודשת? וכיצד ידע על קיומם של מחשבים שכבר לא מחוברים לרשת?

פתרון: אני משתמש במאגר נתונים מקומי בעזרת SQLite. SQLite למעשה לא דורשת הפעלה של שרת Database כיוון שאין צורך בגישה ל-Database דרך הרשת. ה-Database נכתב בקובץ מקומי והתוכנה ניגשת אליו בעת הצורך.

תדפיס של הקוד

Cipher.py

*# region ---------------------------- ABOUT ----------------------------  
"""  
##################################################################  
# Created By: Ariel Westfried #  
# Date: 01/01/2016 #  
# Name: Cipher #  
# Version: 1.0 #  
# Windows Tested Versions: Win 7 32-bit #  
# Python Tested Versions: 2.6 32-bit #  
# Python Environment : PyCharm #  
##################################################################  
"""  
# endregion  
  
# region ---------------------------- IMPORTS ----------------------------***from** Crypto.PublicKey **import** RSA  
**from** Crypto.Cipher **import** AES  
**from** Crypto.Hash **import** MD5  
**from** Crypto.Random **import** get\_random\_bytes  
**import** pickle  
*# endregion  
  
# region ---------------------------- CONSTANTS ----------------------------*RSA\_KEY\_LENGTH = 1024  
AES\_KEY\_LENGTHS = [16, 24, 32]  
RSA\_BLOCK\_SIZE = 128  
BLOCK\_SEPARATOR = **'~'***# endregion  
  
# region ---------------------------- Cipher CLASS ----------------------------***class** Cipher(object):  
 \_\_signature = False  
 \_\_public = False  
  
 *# constructor* **def** \_\_init\_\_(self, key=None):  
 **if not** key: *# Creates random asymmetrical key* self.\_\_cipher\_object = RSA.generate(RSA\_KEY\_LENGTH)  
 self.asymmetrical = True  
 **elif** key == **'signature'**: *# creates a random asymmetrical key that functions as a signature* self.\_\_cipher\_object = RSA.generate(RSA\_KEY\_LENGTH)  
 self.\_\_signature = True  
 self.asymmetrical = True  
 **elif** isinstance(key, str) **and** len(key) **in** AES\_KEY\_LENGTHS: *# creates symmetrical key with given string* self.\_\_cipher\_object = AES.new(key)  
 self.asymmetrical = False  
 self.\_\_key = key  
 **elif** isinstance(key, Cipher): *# extracts public key from an existing key* self.\_\_cipher\_object = key.\_\_cipher\_object.publickey()  
 **if** key.\_\_signature:  
 self.\_\_signature = True  
 self.asymmetrical = True  
 self.\_\_public = True  
 **else**:  
 **raise** NameError(**"'key' must be 16, 24 or 32 byte long (or a pubkey object)."**)  
  
 *# the encrypting method* **def** encrypt(self, data):  
 *"""  
 Input:  
 data - the plaintext  
 Output:  
 Encrypted message  
 """* **if** self.\_\_signature:  
 cipher\_data = self.\_\_cipher\_object.decrypt(data)  
 **elif** self.asymmetrical:  
 cipher\_data = **""** data\_parts = [data[i: i + RSA\_BLOCK\_SIZE] **for** i **in** xrange(0, len(data), RSA\_BLOCK\_SIZE)]  
 **for** part **in** data\_parts:  
 cipher\_data += self.\_\_cipher\_object.encrypt(part, 1)[0].encode(**'base64'**) + BLOCK\_SEPARATOR  
 **return** cipher\_data  
 **else**:  
 to\_add = 16 - (len(data) % 16)  
 data += **' '** \* to\_add  
 cipher\_data = self.\_\_cipher\_object.encrypt(data)  
 **return** cipher\_data.encode(**'base64'**)  
  
 *# the decrypting method* **def** decrypt(self, data):  
 *"""  
 Input:  
 data - the encrypted data  
 Output: plaintext  
 """* **if** self.\_\_public **and not** self.\_\_signature:  
 **raise** NameError(**"Public key cannot decrypt!"**)  
 **if** self.\_\_signature:  
 decrypted = self.\_\_cipher\_object.encrypt(data.decode(**'base64'**), 1)[0]  
 **elif** self.asymmetrical:  
 decrypted = **""  
 for** part **in** data.split(BLOCK\_SEPARATOR)[:-1]:  
 to\_add = self.\_\_cipher\_object.decrypt(part.decode(**'base64'**))  
 decrypted += to\_add  
 **else**:  
 data = data.decode(**'base64'**)  
 decrypted = self.\_\_cipher\_object.decrypt(data)  
 **if ' ' in** decrypted:  
 decrypted = decrypted.rstrip(**' '**)  
 **return** decrypted  
  
 *# public key generator* **def** public\_key(self):  
 **if** self.asymmetrical:  
 **return** Cipher(key=self)  
  
 *# serialization* **def** pack(self):  
 **if** self.asymmetrical:  
 **return '1'** + pickle.dumps(self).encode(**'base64'**)  
 **return '0'** + self.\_\_key.encode(**'base64'**)  
  
 *# alternative constructor for random symmetric key* @staticmethod  
 **def** random\_key():  
 **return** Cipher(get\_random\_bytes(32))  
  
 *# deserialization* @staticmethod  
 **def** unpack(data):  
 asymmetrical = data[0] == **'1'  
 if not** asymmetrical:  
 **return** Cipher(key=data[1:].decode(**'base64'**))  
 **return** pickle.loads(data[1:].decode(**'base64'**))  
  
 *# alternative constructor for signature object* @staticmethod  
 **def** crate\_signature():  
 **return** Cipher(key=**'signature'**)  
  
 *# hashing method* @staticmethod  
 **def** hash(data):  
 md5 = MD5.new(data)  
 **return** md5.hexdigest()  
  
*# endregion*

Client.py

**try**:  
 **import** socket  
 **from** uuid **import** getnode **as** get\_mac  
 **from** Constants **import** \*  
 **from** threading **import** Thread, Lock  
 **from** wmi **import** WMI  
 **from** time **import** sleep  
 **from** win32file **import** CreateDirectory, DeleteFile, RemoveDirectory  
 **from** win32process **import** CreateProcess, STARTUPINFO, TerminateProcess, STARTF\_USESHOWWINDOW  
 **from** win32api **import** OpenProcess, GetLogicalDriveStrings, CloseHandle  
 **from** win32con **import** PROCESS\_TERMINATE, NORMAL\_PRIORITY\_CLASS, SW\_NORMAL  
 **import** pythoncom  
 **import** os  
 **from** os.path **import** exists  
 **from** ctypes **import** windll  
 **from** select **import** select  
 **from** Process **import** Process  
 **import** pickle  
 **import** subprocess  
 **import** sys  
 **from** Cipher **import** Cipher  
**except**:  
 pipe = subprocess.Popen([sys.executable, **'install.py'**], stdin=subprocess.PIPE)  
 pipe.stdin.write(**'Client.py'** + END\_LINE)  
 exit()  
  
  
**class** Client(object):  
 **def** \_\_init\_\_(self):  
 self.\_\_socket = socket.socket()  
 self.\_\_mac = get\_mac()  
 self.\_\_name = socket.gethostname()  
 self.\_\_processes = []  
 self.\_\_key = Cipher.random\_key()  
 self.handle\_functions = {  
 **"CreateFile"**: self.\_\_create\_file,  
 **"DeleteFile"**: self.\_\_delete\_file,  
 **"CreateProcess"**: self.\_\_create\_process,  
 **"TerminateProcess"**: self.\_\_terminate\_process,  
 **"UpdateProcesses"**: self.\_\_send\_processes,  
 **"GetFile"**: self.\_\_get\_file,  
 **"Upload"**: self.\_\_send\_file  
 }  
  
 **def** \_\_send\_file(self, directory):  
 **try**:  
 the\_file = open(directory, **'rb'**)  
 file\_data = the\_file.read()  
 the\_file.close()  
 **return** file\_data  
 **except**:  
 **return "ERROR: File doesn't exists!"  
  
 def** \_\_create\_file(self, path, name):  
 *"""  
 Creates a new file in directory  
 """* **if** exists(path):  
 directory = path  
 **if not** directory.endswith(**'\\'**):  
 directory += **'\\'** directory += name  
 **try**:  
 **if** len(directory.split(**'.'**)) >= 2:  
 open(directory, **'w'**)  
 **else**:  
 CreateDirectory(directory, None)  
 result = **"Created "** + directory  
 **except**:  
 result = **"ERROR: internal error"  
 else**:  
 result = **"ERROR: no directory "** + path  
 **return** result  
  
 **def** \_\_delete\_file(self, path):  
 *"""  
 Deltes the file in path  
 """* **if** exists(path):  
 **try**:  
 **if** len(path.split(**'.'**)) >= 2:  
 DeleteFile(path)  
 **else**:  
 RemoveDirectory(path)  
 windll.shell32.SHEmptyRecycleBinA(None, None, 1 **or** 2 **or** 4)  
 result = **"Deleted "** + path  
 **except**:  
 result = **"ERROR: internal error"  
 else**:  
 result = **"ERROR: no directory "** + path  
 **return** result  
  
 **def** \_\_create\_process(self, command):  
 *"""  
 Opens a new process  
 """* **try**:  
 startup\_info = STARTUPINFO()  
 startup\_info.dwFlags = STARTF\_USESHOWWINDOW  
 startup\_info.wShowWindow = SW\_NORMAL  
 CreateProcess(None, command, None, None, 0, NORMAL\_PRIORITY\_CLASS, None, None, startup\_info)  
 result = **"Opened "** + command  
 **except**:  
 result = **"ERROR: internal error"  
 return** result  
  
 **def** \_\_terminate\_process(self, pid, name):  
 *"""  
 kill a process  
 """* process\_exists = False  
 **for** process **in** self.\_\_processes:  
 **if** process.pid == pid:  
 process\_exists = True  
 **break  
 if** process\_exists:  
 **try**:  
 handle = OpenProcess(PROCESS\_TERMINATE, False, int(pid))  
 TerminateProcess(handle, -1)  
 CloseHandle(handle)  
 result = name+ **'-'** + pid + **" terminated"  
 except**:  
 result = **"ERROR: internal error"  
 else**:  
 result = **"ERROR: no process "** + name + **'-'** + pid  
 **return** result  
  
 **def** \_\_send\_processes(self):  
 *"""  
 Creates a list of all processes  
 """* result = []  
 self.\_\_update\_processes()  
 **for** process **in** self.\_\_processes:  
 process\_parts = str(process.name) + PROCESS\_PARTS\_SEPARATOR + str(process.pid) + PROCESS\_PARTS\_SEPARATOR + str(process.parent\_id)  
 result.append(str(process\_parts))  
 **return** str(result)  
  
 **def** \_\_print(self, data):  
 **print** data  
  
 **def** start(self):  
 *"""  
 starts the client program  
 """* self.\_\_print(**"connecting"**)  
 is\_server = self.\_\_find\_server()  
 **if** is\_server:  
 **print "connected to server."** self.\_\_run()  
 **else**:  
 **print "server not found."  
  
 def** \_\_find\_server(self):  
 *"""  
 Looking for server in the network. if found returns True else False  
 """* search\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 search\_socket.settimeout(ANNOUNCE\_SLEEP\_TIME)  
 search\_socket.bind((**"0.0.0.0"**, BROADCAST\_PORT))  
 **try**:  
 message, address = search\_socket.recvfrom(BUFFER\_SIZE)  
 **except**:  
 **return** False  
 **else**:  
 server\_address = address[0]  
 **if** message == SERVER\_ANNOUNCE\_MESSAGE:  
 status = self.\_\_socket.connect\_ex((server\_address, SERVER\_PORT))  
 **if** status == 0:  
 **return** self.\_\_key\_exchange()  
 **return** False  
  
 **def** \_\_key\_exchange(self):  
 server\_public\_key\_data = self.\_\_socket.recv(BUFFER\_SIZE)  
 server\_public\_key = Cipher.unpack(server\_public\_key\_data)  
 to\_send = self.\_\_key.pack() + IN\_PACK\_SEPARATOR + Cipher.hash(self.\_\_key.pack())  
 to\_send = server\_public\_key.encrypt(to\_send)  
 self.\_\_socket.send(to\_send)  
 **return** True  
  
 **def** \_\_run(self):  
 *"""  
 The actual main code of the client  
 """* **while** True:  
 data = self.\_\_get\_data()  
 **if** data != **""**:  
 data = data.split()  
 case = len(data)  
 **try**:  
 **if** case == 3:  
 result = self.handle\_functions[data[0]](data[1], data[2])  
 **elif** case == 2:  
 result = self.handle\_functions[data[0]](data[1])  
 **elif** case == 1:  
 result = self.handle\_functions[data[0]]()  
 **else**:  
 result = **"ERROR"  
 except**:  
 result = **"ERROR"** self.\_\_return\_answer(result)  
  
 **def** \_\_get\_data(self):  
 *"""  
 Relieves the requests from the server  
 """* to\_read, to\_write, error = select([self.\_\_socket], [self.\_\_socket], [])  
 **if** self.\_\_socket **in** to\_read:  
 **try**:  
 data = self.\_\_socket.recv(BUFFER\_SIZE)  
 **except** socket.error:  
 data = **""  
 if** data == **""**:  
 self.\_\_print(**"Lost connection with server."**)  
 exit()  
 **return** self.\_\_decrypt(data)  
 **return ""  
  
 def** \_\_encrypt(self, data):  
 result = self.\_\_key.encrypt(data) + IN\_PACK\_SEPARATOR + Cipher.hash(data)  
 **return** result  
  
 **def** \_\_decrypt(self, data):  
 data, hashed = data.split(IN\_PACK\_SEPARATOR)  
 data = self.\_\_key.decrypt(data)  
 **if** Cipher.hash(data) == hashed:  
 **return** data  
 **else**:  
 **raise** EnvironmentError(**"Server Unauthorised"**)  
  
 **def** \_\_return\_answer(self, data):  
 *"""  
 sends back the result  
 """* to\_send = self.\_\_encrypt(data)  
 to\_send = [to\_send[i:i + BUFFER\_SIZE - 3] **for** i **in** xrange(0, len(to\_send), BUFFER\_SIZE - 3)]  
 self.\_\_socket.send(str(len(to\_send)))  
 sleep(1)  
 **for** i **in** xrange(len(to\_send)):  
 num = str(i)  
 num = **""**.join([str(j - j) **for** j **in** xrange(2 - len(num))]) + num  
 self.\_\_socket.send(num + **"@"** + to\_send[i])  
  
 **def** \_\_update\_processes(self):  
 *"""  
 Enumerates all running processes  
 """* wmi = WMI()  
 self.\_\_processes = []  
 **for** process **in** wmi.Win32\_Process():  
 process\_object = Process(process.Name, str(process.ProcessID), str(process.ParentProcessID))  
 self.\_\_processes.append(process\_object)  
  
 **def** \_\_get\_file(self, path):  
 *"""  
 Returns whats is inside that file  
 """* **if** path == EMPTY\_PATH:  
 files = FILE\_SEPARATOR.join(GetLogicalDriveStrings().split(**'\000'**)[:-1])  
 **elif** exists(path):  
 files = FILE\_SEPARATOR.join([str(f) **for** f **in** os.listdir(path)])  
 **else**:  
 files = **"ERROR: no directory "** + path  
 **return** files  
  
  
**def** main():  
 client = Client()  
 client.start()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

ClientInterfae.py

*# region ---------------------------- ABOUT ----------------------------  
"""  
##################################################################  
# Created By: Ariel Westfried #  
# Date: 01/01/2016 #  
# Name: Project - ClientInterface #  
# Version: 1.0 #  
# Windows Tested Versions: Win 7 32-bit #  
# Python Tested Versions: 2.6 32-bit #  
# Python Environment : PyCharm #  
##################################################################  
"""  
# endregion  
  
# region ---------------------------- IMPORTS ----------------------------***from** Constants **import** \*  
**from** socket **import** gethostbyaddr  
**from** WakeOnLan **import** wake\_on\_lan, shutdown  
**from** Process **import** Process  
**from** socket **import** socket  
**from** Cipher **import** Cipher  
*# endregion  
  
# region ---------------------------- CLASSES ----------------------------  
# region ---------------------------- ClientInterface CLASS ----------------------------***class** ClientInterface(object):  
 *# constructor* **def** \_\_init\_\_(self, sock, computer, key):  
 self.\_\_socket = sock  
 self.\_\_key = key  
 **if not** isinstance(computer, Computer):  
 **raise** ValueError  
 **else**:  
 self.\_\_computer = computer  
 self.name = gethostbyaddr(computer.ip)[0]  
 self.processes = []  
  
 *# return the client's mac* **def** get\_mac(self):  
 **return** self.\_\_computer.mac  
  
 *# returns the client's IP* **def** get\_ip(self):  
 **return** self.\_\_computer.ip  
  
 **def** \_\_eq\_\_(self, other):  
 **if** isinstance(other, ClientInterface):  
 **return** self.\_\_computer.mac == self.\_\_computer.mac  
 **elif** isinstance(other, Computer):  
 **return** self.\_\_computer.mac == other.mac  
 **else**:  
 **return** False  
  
 *# updates the process list* **def** update\_processes(self):  
 self.\_\_send(**"UpdateProcesses"**)  
 data = self.\_\_receive()  
 process\_list = [item[1:-1] **for** item **in** data[1:-1].split(**", "**)]  
 self.processes = []  
 **for** item **in** process\_list:  
 process\_parts = item.split(PROCESS\_PARTS\_SEPARATOR)  
 **try**:  
 process = Process(process\_parts[0], process\_parts[1], process\_parts[2])  
 **except**:  
 **pass  
 else**:  
 self.processes.append(process)  
  
 *# receives a file from the client* **def** download(self, file\_path):  
 self.\_\_send(**"Upload "** + file\_path)  
 data = self.\_\_receive()  
 **return** data  
  
 *# encrypts a message and sends it to the client* **def** \_\_send(self, data):  
 to\_send = self.\_\_encrypt(data)  
 self.\_\_socket.send(to\_send)  
  
 *# the encrypt method* **def** \_\_encrypt(self, data):  
 result = self.\_\_key.encrypt(data) + IN\_PACK\_SEPARATOR + Cipher.hash(data)  
 **return** result  
  
 *# receives an encrypted message from the client and returns it unencrypted* **def** \_\_receive(self):  
 parts = {}  
 length = self.\_\_socket.recv(BUFFER\_SIZE)  
 **for** i **in** xrange(int(length)):  
 data = self.\_\_socket.recv(BUFFER\_SIZE)  
 number, data = data.split(FRAGMENTS\_SEPARATOR)  
 parts[int(number)] = data  
 data = **""  
 for** i **in** xrange(int(length)):  
 data += parts[i]  
 **return** self.\_\_decrypt(data)  
  
 *# decrypts a message* **def** \_\_decrypt(self, data):  
 result, hashed = data.split(IN\_PACK\_SEPARATOR)  
 result = self.\_\_key.decrypt(result)  
 **if** Cipher.hash(result) == hashed:  
 **return** result  
 **else**:  
 **raise** EnvironmentError(**"CLIENT UNAUTHORISED"**)  
  
 *# Tells the client to terminate a process* **def** terminate(self, processes):  
 result = **""  
 for** process **in** processes:  
 **if** isinstance(process, Process):  
 self.\_\_send(**"TerminateProcess "** + process.pid + **" "** + process.name)  
 result += self.\_\_receive()  
 **return** result  
  
 *# Tells the client to open a process* **def** open\_process(self, command):  
 self.\_\_send(**"CreateProcess "** + command)  
 result = self.\_\_receive()  
 **return** result  
  
 *# Ask the client for the content of a folder* **def** send\_files(self, directory):  
 self.\_\_send(**"GetFile "** + directory)  
 result = self.\_\_receive()  
 **if "ERROR" not in** result:  
 result = result.split(FILE\_SEPARATOR)  
 **return** result  
 **return** result  
  
 *# Tells the client to delete a file or a folder* **def** delete\_file(self, path):  
 self.\_\_send(**"DeleteFile "** + path)  
 result = self.\_\_receive()  
 **return** result  
  
 *# tells the client to create a new file or a folder* **def** create\_file(self, path, name):  
 self.\_\_send(**"CreateFile "** + path + **" "** + name)  
 result = self.\_\_receive()  
 **return** result  
*# endregion  
  
# region ---------------------------- ClientList CLASS ----------------------------***class** ClientList(object):  
 *# constructor* **def** \_\_init\_\_(self):  
 self.\_\_items = []  
  
 **def** \_\_len\_\_(self):  
 **return** len(self.\_\_items)  
  
 **def** \_\_getitem\_\_(self, key):  
 **return** self.\_\_items[key]  
  
 **def** \_\_delitem\_\_(self, key):  
 **del** self.\_\_items[key]  
  
 **def** \_\_setitem\_\_(self, key, value):  
 **if** isinstance(value, ClientInterface):  
 self.\_\_items[key] = value  
  
 *# adds a client at the end of the list* **def** append(self, item):  
 **if** isinstance(item, ClientInterface):  
 self.\_\_items.append(item)  
  
 *# deletes a client from the list* **def** remove(self, item):  
 **if** isinstance(item, ClientInterface):  
 self.\_\_items.remove(item)  
 **elif** isinstance(item, socket):  
 **for** client **in** self.\_\_items:  
 **if** client.socket **is** item:  
 self.\_\_items.remove(client)  
 **break  
  
 def** \_\_iter\_\_(self):  
 **for** item **in** self.\_\_items:  
 **yield** item  
*# endregion  
  
# region ---------------------------- Computer CLASS ----------------------------***class** Computer(object):  
 *# constructor* **def** \_\_init\_\_(self, mac, ip, active=True):  
 self.mac = mac  
 self.ip = ip  
 self.active = active  
  
 **def** \_\_eq\_\_(self, other):  
 **if** isinstance(other, Computer):  
 **return** self.mac == other.mac  
 **if** isinstance(other, ClientInterface):  
 **return** self.mac == other.get\_mac()  
 **return** False  
  
 **def** \_\_repr\_\_(self):  
 **if** self.active:  
 status = **"online"  
 else**:  
 status = **"offline"  
 return** self.mac + **" "** \* 4 + self.ip + **" "** \* 4 + status  
  
 **def** \_\_str\_\_(self):  
 **if** self.active:  
 status = **"online"  
 else**:  
 status = **"offline"  
 return** self.mac + **" "** \* 4 + self.ip + **" "** \* 4 + status  
  
 *# Stars the computer with wake on lan* **def** wake\_up(self):  
 **if not** self.active:  
 wake\_on\_lan(self.mac)  
 self.active = True  
 **else**:  
 **raise** NameError  
  
 *# remotely shuts down the computer* **def** shutdown(self):  
 **if** self.active:  
 shutdown(self.ip, **"This computer will be shud down by the network manager."**, 5)  
 self.active = False  
 **else**:  
 **raise** NameError  
*# endregion  
# endregion*

ComputerDatabase.py

*# region ---------------------------- ABOUT ----------------------------  
"""  
##################################################################  
# Created By: Ariel Westfried #  
# Date: 01/01/2016 #  
# Name: Project - ComputerDatabase #  
# Version: 1.0 #  
# Windows Tested Versions: Win 7 32-bit #  
# Python Tested Versions: 2.6 32-bit #  
# Python Environment : PyCharm #  
##################################################################  
"""  
# endregion  
  
# region ---------------------------- IMPORTS ----------------------------***import** sqlite3  
**from** Constants **import** \*  
**from** ClientInterface **import** Computer  
**import** re  
*# endregion  
  
# region ---------------------------- ComputerDatabase CLASS ----------------------------***class** ComputerDatabase:  
 *# constructor* **def** \_\_init\_\_(self): *# Constructor* self.\_\_create\_computers\_table()  
  
 *# creates a new table in cas ethe table doesn't exist* **def** \_\_create\_computers\_table(self):  
 *"""  
 Creates the tables if it doesn't exist  
 """* database, cursor = self.\_\_open()  
 cursor.execute(**'''CREATE TABLE if not exists Computers  
 (MAC STRING,  
 IP STRING,  
 active STRING);'''**)  
 database.commit()  
 self.\_\_close(cursor, database)  
  
 *# adds a new computer to the database* **def** add\_row(self, computer):  
 *"""  
 Inserts a new row to the table  
 """* **if** isinstance(computer, Computer):  
 database, cursor = self.\_\_open()  
 cursor.execute(**"INSERT INTO Computers VALUES('%s','%s','%s')"** % (computer.mac, computer.ip, str(computer.active)))  
 database.commit()  
 self.\_\_close(cursor, database)  
 **else**:  
 **raise** ValueError  
  
 *# Return the database as a list of Computer objects* **def** read(self):  
 *"""  
 Reads all the database and returns it as a list  
 """* database, cursor = self.\_\_open()  
 computers = []  
 cursor.execute(**"SELECT \* FROM Computers ORDER BY active DESC"**)  
 database.commit()  
 rows = cursor.fetchall()  
 **for** row **in** rows:  
 computers.append(Computer(row[0], row[1], row[2] == **u"True" or** row[2] == **"True"**))  
 self.\_\_close(cursor, database)  
 **return** computers  
  
 *# changes the state (online or offline) of a computer in the database* **def** update\_state(self, computer):  
 *"""  
 Updates the state of a computer  
 """* database, cursor = self.\_\_open()  
 **if** isinstance(computer, Computer):  
 active = str(computer.active)  
 cursor.execute(**"UPDATE Computers SET active='%s' WHERE MAC='%s'"** % (active, computer.mac))  
 database.commit()  
 self.\_\_close(cursor, database)  
 **return  
 elif** (isinstance(computer, str) **or** isinstance(computer, unicode)) **and** re.match(IP\_REGULAR\_EXPRESSION, computer):  
 parameter = **"IP"  
 elif** (isinstance(computer, str) **or** isinstance(computer, unicode)) **and** re.match(MAC\_REGULAR\_EXPRESSION, computer):  
 parameter = **"MAC"  
 else**:  
 **raise** ValueError  
 cursor.execute(**"SELECT active FROM Computers WHERE %s='%s'"** % (parameter, computer))  
 database.commit()  
 active = cursor.fetchone()  
 active = active == **u"True" or "True"** active = **not** active  
 cursor.execute(**"UPDATE Computers SET active='%s' WHERE %s='%s'"** % (active, parameter, computer))  
 database.commit()  
 self.\_\_close(cursor, database)  
  
 *# returns the database as a dictionary* **def** make\_dictionary(self):  
 computers\_dict = {**"IP"**: [], **"MAC"**: [], **"STATUS"**: [], **"INDEX"**: []}  
 computers = self.read()  
 i = 1  
 **for** computer **in** computers:  
 computers\_dict[**"IP"**].append(computer.ip)  
 computers\_dict[**"MAC"**].append(computer.mac)  
 computers\_dict[**"STATUS"**].append( **"online" if** computer.active **else "offline"**)  
 computers\_dict[**"INDEX"**].append(i)  
 i += 1  
 **return** computers\_dict  
  
 *# closes the database objects* **def** \_\_close(self, cursor, database):  
 *"""  
 Closes itself  
 """* cursor.close()  
 database.close()  
  
 *# opens the database objecs* **def** \_\_open(self):  
 database = sqlite3.connect(DATABASE\_NAME)  
 cursor = database.cursor()  
 **return** database, cursor  
*# endregion  
  
# region ---------------------------- MAIN ----------------------------  
  
  
# prints the database***def** main():  
 db = ComputerDatabase()  
 data = db.read()  
 **for** computer **in** data:  
 **print** str(computer)  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()  
*# endregion*

Constants.py

*# region ---------------------------- ABOUT ----------------------------  
"""  
##################################################################  
# Created By: Ariel Westfried #  
# Date: 01/01/2016 #  
# Name: Project - Constants #  
# Version: 1.0 #  
# Windows Tested Versions: Win 7 32-bit #  
# Python Tested Versions: 2.6 32-bit #  
# Python Environment : PyCharm #  
##################################################################  
"""  
# endregion  
  
# region ---------------------------- Constants ----------------------------*MODULES\_TO\_INSTALL = [**"flask"**, **"wmi"**, **"pywin32"**, **"scapy"**, **"pywinrm"**] *# The modules needed for the program to run*SERVER\_PORT = 6070 *# The port where the server listens*BROADCAST\_PORT = 6071 *# The port used for the broadcast communication*SERVER\_ANNOUNCE\_MESSAGE = **"I'm the network manager server."** *# The message used to identify the server*BUFFER\_SIZE = 1024 *# The buffer size of the socket communication*IP\_REGULAR\_EXPRESSION = **r"^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$"** *# The regex for a valid ip*FLASK\_URL = **"http://127.0.0.1:5000/"** *# The url where the server sits*DATABASE\_NAME = **r"Database.db"** *# Name of the database file*MAC\_LENGTH = 12 *# Length of mac address*WAKE\_ON\_LAN\_PORT = 9 *# The port used for wake on lan protocol*NET\_SCAN\_WAIT = 60 *# Time to wait between network scans*ANNOUNCE\_SLEEP\_TIME = 10 *# Time to wait between broadcast announcements*PROCESS\_ENUMERATE\_SLEEP = 60 *# Time to wait between updating process list*EMPTY\_PATH = **'Drives'** *# First path when looking for folders*END\_LINE = **'\r\n'** *# End of line*BROKEN\_END\_LINE = **'\\r\\n'** *# End of line with an error*MAC\_REGULAR\_EXPRESSION = **b"^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$"** *# Regex for a valid mac address*PROCESS\_PARTS\_SEPARATOR = **"#"** *# Separator used between process when in string*FILE\_SEPARATOR = **'#'** *# Separator used between file names in string*IN\_PACK\_SEPARATOR = **'#'** *# Separator for key exchange communication*FRAGMENTS\_SEPARATOR = **'@'** *# Separates between a packet and its serial number*DOWNLOAD\_UPLOAD = **'C:\\ServerFiles\\'** *# Directory where to save downloaded files*

GuiEngine.py

*# region ---------------------------- ABOUT ----------------------------  
"""  
##################################################################  
# Created By: Ariel Westfried #  
# Date: 01/01/2016 #  
# Name: Project - GuiEngine #  
# Version: 1.0 #  
# Windows Tested Versions: Win 7 32-bit #  
# Python Tested Versions: 2.6 32-bit #  
# Python Environment : PyCharm #  
##################################################################  
"""  
# endregion  
  
# region ---------------------------- IMPORTS ----------------------------***from** Constants **import** \*  
**from** flask **import** Flask, render\_template, request, redirect, url\_for, safe\_join, send\_from\_directory  
**from** itertools **import** izip  
**import** webbrowser  
**from** Server **import** Server  
**from** threading **import** Thread  
**from** ClientInterface **import** Computer  
**from** Process **import** Process  
*# endregion  
  
# region ---------------------------- FLASK METHODS ----------------------------*app = Flask(\_\_name\_\_)  
server = Server()  
  
  
**def** unicode\_to\_list(unicode):  
 **return** [item[2:-1] **for** item **in** unicode[1:-1].split(**', '**)]  
  
  
*# handles MAinPage.html*@app.route(**'/'**, methods=[**'GET'**, **'POST'**])  
**def** show\_main\_form():  
 **if** (**not** server.running) **and** (**not** server.starting):  
 start\_thread = Thread(target=server.start)  
 start\_thread.setDaemon(True)  
 start\_thread.start()  
 **return** render\_template(**"Loading.html"**)  
 **elif** server.starting:  
 **return** render\_template(**"Loading.html"**)  
 **elif** server.running:  
 **if** request.method == **'POST'**:  
 **if** request.form[**'Action'**] == **"More"**:  
 ip = request.form[**'Ip'**]  
 mac = request.form[**'Mac'**]  
 url = url\_for(**"view\_computer"**, mac=mac, ip=ip)  
 **return** redirect(url)  
 **elif** request.form[**'Action'**] == **"Remote Desktop"**:  
 computers = request.form.getlist(**'check'**)  
 computer\_list = [Computer(comp.split(**'\_'**)[0], comp.split(**'\_'**)[1]) **for** comp **in** computers]  
 server.remote\_desktop(computer\_list)  
 **elif** request.form[**'Action'**] == **"Add Computer Manually"**:  
 **return** redirect(url\_for(**'add\_computer'**))  
 **else**:  
 action = request.form[**'Action'**]  
 **if** action == **'WakeOnLAN'**:  
 action = **'wake up'  
 else**:  
 action = **'shutdown'** computer\_list = request.form.getlist(**'check'**)  
 **return** redirect(url\_for(**'wake\_on\_lan'**, computer\_list=str(computer\_list), action=action))  
 computers\_dict = server.make\_computers\_dictionary()  
 computers = [dict(IP=ip,MAC=mac, STATUS=state, INDEX=index, CONNECTED=connected) **for** ip, mac, state, index, connected **in** izip(computers\_dict[**'IP'**], computers\_dict[**'MAC'**], computers\_dict[**'STATUS'**], computers\_dict[**'INDEX'**], computers\_dict[**'CONNECTED'**])]  
 **return** render\_template(**"MainPage.html"**, computers=computers)  
  
  
*# handles AddPage.html*@app.route(**'/add\_computer'**, methods=[**'GET'**, **'POST'**])  
**def** add\_computer():  
 message = **""  
 if** request.method == **'POST'**:  
 ip = request.form[**'Ip'**]  
 mac = request.form[**'Mac'**]  
 message = server.and\_computer(Computer(mac, ip, False))  
 **return** render\_template(**'AddPage.html'**, message=message)  
  
  
*# handles Downloading*@app.route(**'/download\_file?mac=<mac>&ip=<ip>&directory=<path:directory>&item=<item>'**, methods=[**'GET'**])  
**def** download\_file(mac, ip, directory, item):  
 file\_to\_send = server.download(Computer(mac, ip), safe\_join(directory, item))  
 **return** send\_from\_directory(DOWNLOAD\_UPLOAD, file\_to\_send)  
  
  
*# handles InfoPage.html*@app.route(**'/view\_computer?mac=<mac>&ip=<ip>'**, methods=[**'GET'**, **'POST'**])  
**def** view\_computer(mac, ip):  
 message = **""  
 if** request.method == **'POST'**:  
 ip = request.form[**'Ip'**]  
 mac = request.form[**'Mac'**]  
 name = request.form[**'Host'**]  
 function = request.form[**'Action'**].lower().replace(**' '**, **'\_'**)  
 **if** function == **"open\_process"**:  
 message = server.open\_process(Computer(mac, ip), request.form[**"ProcessName"**])  
 **elif** function == **'show\_files'**:  
 **return** redirect(url\_for(function, mac=mac, ip=ip, name=name, path=str(EMPTY\_PATH)))  
 **else**:  
 **return** redirect(url\_for(function, mac=mac, ip=ip, name=name))  
 computer\_data = server.computer\_data(Computer(mac, ip))  
 **return** render\_template(**"InfoPage.html"**, computer=computer\_data, message=message)  
  
  
*# handles FilesPage.html*@app.route(**'/view\_files?mac=<mac>&ip=<ip>&name=<name>&path=<path:path>'**, methods=[**'GET'**, **'POST'**])  
**def** show\_files(mac, ip, name, path):  
 message = **""  
 if** request.method == **'POST'**:  
 action = request.form[**'Action'**]  
 **if** action == **'Open'**:  
 **if** path == EMPTY\_PATH:  
 path = **""  
 else**:  
 path += **'\\'  
 return** redirect(url\_for(**'show\_files'**, mac=mac, ip=ip, name=name, path=path + request.form[**'FileName'**]))  
 **elif** action == **'Create'**:  
 message = server.create\_file(Computer(mac, ip), path, request.form[**'FileName'**])  
 **else**:  
 message = server.delete\_file(Computer(mac, ip), safe\_join(path, request.form[**'FileName'**]))  
 directory = server.get\_file(Computer(mac, ip), path)  
 **if "ERROR" in** directory[**'ITEMS'**]:  
 message = directory[**'ITEMS'**]  
 directory[**'ITEMS'**] = []  
 **return** render\_template(**"FilesPage.html"**, directory=directory, mac=mac, ip=ip, name=name, message=message)  
  
  
*# handles ProcessesPage.html*@app.route(**'/view\_processes?mac=<mac>&ip=<ip>&name=<name>'**, methods=[**'GET'**, **'POST'**])  
**def** show\_processes(mac, ip, name):  
 message = **""** computer = Computer(mac, ip)  
 **if** request.method == **'POST'**:  
 process\_list = request.form.getlist(**'check'**)  
 function = request.form[**'Action'**]  
 **if** function == **"Terminate"**:  
 new\_list = []  
 **for** process **in** process\_list:  
 process\_name, pid, parent\_pid = process.split(**u'\_'**)  
 new\_list.append(Process(process\_name, pid, parent\_pid))  
 process\_list = new\_list  
 message = server.terminate\_process(computer, process\_list)  
 **elif** function == **"Back"**:  
 **return** redirect(url\_for(**"view\_computer"**, mac=mac, ip=ip))  
 process\_list = server.get\_processes\_data(computer)  
 **return** render\_template(**"ProcessesPage.html"**, process\_list=process\_list, mac=mac, ip=ip, name=name, message=message)  
  
  
*# handles WakeOnLan.html*@app.route(**'/wake\_on\_lan?computer\_list=<computer\_list>&action=<action>'**, methods=[**'GET'**, **'POST'**])  
**def** wake\_on\_lan(computer\_list, action):  
 message = **""  
 if** request.method == **'POST'**:  
 active = action == **'shutdown'  
 if 'now' in** request.form[**'Go'**]:  
 **for** part **in** unicode\_to\_list(computer\_list):  
 mac, ip = part.split(**'\_'**)  
 server.do\_action\_now(Computer(mac, ip, active), action)  
 **return** redirect(url\_for(**'show\_main\_form'**))  
 **else**:  
 hour = request.form[**'hour'**]  
 minute = request.form[**'minute'**]  
 second = request.form[**'second'**]  
 **try**:  
 **for** part **in** [item[1:-1] **for** item **in** computer\_list[1:-1].split(**', '**)]:  
 mac, ip = part.split(**'\_'**)  
 server.do\_action(Computer(mac, ip, active), hour, minute, second, action)  
 **except**:  
 message = **"Error: Time wasn't inserted correctly."  
 else**:  
 **return** redirect(url\_for(**'show\_main\_form'**))  
 names = [str(item.split(**'\_'**)[1]) **for** item **in** unicode\_to\_list(computer\_list)]  
 **return** render\_template(**"WakeOnLan.html"**, computer\_list=computer\_list, action=action, message=message, names=names)  
  
  
*# Shows one folder up*@app.route(**'/go\_back?mac=<mac>&ip=<ip>&name=<name>&path=<path:path>'**)  
**def** back(mac, ip, name, path):  
 new\_path = **'\\'**.join(path.split(**'\\'**)[:-1])  
 **if not** new\_path:  
 new\_path = EMPTY\_PATH  
 **return** redirect(url\_for(**'show\_files'**, mac=mac, ip=ip, name=name, path=new\_path))  
*# endregion  
  
# region ---------------------------- MAIN ----------------------------***def** main():  
 webbrowser.open(FLASK\_URL)  
 app.run(host=**"0.0.0.0"**, debug=False)  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()  
*# endregion*

NetMap.py

*# region ---------------------------- ABOUT ----------------------------  
"""  
##################################################################  
# Created By: Ariel Westfried #  
# Date: 01/01/2016 #  
# Name: Project - NetMap #  
# Version: 1.0 #  
# Windows Tested Versions: Win 7 32-bit #  
# Python Tested Versions: 2.6 32-bit #  
# Python Environment : PyCharm #  
##################################################################  
"""  
# endregion  
  
# region ---------------------------- IMPORTS ----------------------------***from** Constants **import** \*  
**import** re  
**import** subprocess  
**import** wmi  
**from** scapy.all **import** \*  
**from** ClientInterface **import** Computer  
*# endregion  
  
# region ---------------------------- NemMap CLASS ----------------------------***class** NetMap(object):  
 *# main mapping method* @staticmethod  
 **def** map():  
 *"""  
 Main mapping function  
 """* list1 = NetMap.\_\_map\_with\_cmd()  
 list2 = NetMap.\_\_map\_with\_scapy()  
 result\_list = NetMap.\_\_combine\_tables(list1, list2)  
 **return** result\_list  
  
 *# combines the two arp tables generated* @staticmethod  
 **def** \_\_combine\_tables(list1, list2):  
 *"""  
 Combines two lists  
 """* result\_list = list(list1)  
 **for** computer **in** list2:  
 **if** computer **not in** result\_list:  
 result\_list.append(computer)  
 **return** result\_list  
  
 *# takes the computer's table* @staticmethod  
 **def** \_\_map\_with\_cmd():  
 *"""  
 Returns the computer's arp table  
 """* result = []  
 pipe = subprocess.Popen([**'arp'**, **'-a'**], stdout=subprocess.PIPE)  
 **for** line **in** pipe.stdout.readlines()[3:]:  
 **try**:  
 ip, mac, address\_type = line.split()  
 **except** ValueError:  
 ip, mac, address\_type = None, None, None  
 **if** address\_type == **'dynamic'**:  
 mac = mac.replace(**'-'**, **':'**)  
 computer = Computer(mac, ip)  
 result.append(computer)  
 **return** result  
  
 *# surveying the netwotk with scapy* @staticmethod  
 **def** \_\_map\_with\_scapy():  
 *"""  
 makes its own arp table  
 """* result = []  
 conf.verb = 0  
 my\_ip, gateway\_ip, my\_mac, subnet\_mask = NetMap.\_\_get\_network\_attributes()  
 *# print my\_ip, gateway\_ip, my\_mac, subnet\_mask* final\_gateway = NetMap.\_\_get\_final\_gateway(gateway\_ip, subnet\_mask)  
 pkt = Ether(dst=**"ff:ff:ff:ff:ff:ff"**)/ARP(pdst=final\_gateway)  
 answered, unanswered = srp(pkt, timeout=10)  
 **for** snd, rcv **in** answered:  
 mac = rcv.sprintf(**"%Ether.src%"**)  
 ip = rcv.sprintf(**"%ARP.psrc%"**)  
 computer = Computer(mac, ip)  
 result.append(computer)  
 **return** result  
  
 *# return the Broadcast address for the current network* @staticmethod  
 **def** \_\_get\_final\_gateway(gateway\_ip, subnet\_mask):  
 gate\_array = gateway\_ip.split(**"."**, 4)  
 mask\_array = subnet\_mask.split(**"."**, 4)  
 add\_bits = NetMap.\_\_exponent\_two(mask\_array[0]) + NetMap.\_\_exponent\_two(mask\_array[1])\  
 + NetMap.\_\_exponent\_two(mask\_array[2]) + NetMap.\_\_exponent\_two(mask\_array[3])  
 final\_gateway = str(NetMap.\_\_ip\_and(gate\_array[0], mask\_array[0])) + **"."** final\_gateway += str(NetMap.\_\_ip\_and(gate\_array[1], mask\_array[1])) + **"."** final\_gateway += str(NetMap.\_\_ip\_and(gate\_array[2], mask\_array[2])) + **"."** final\_gateway += str(NetMap.\_\_ip\_and(gate\_array[3], mask\_array[3]))  
 final\_gateway += **"/"** + str(add\_bits)  
 **return** final\_gateway  
  
 @staticmethod  
 **def** \_\_exponent\_two(num):  
 num = int(num)  
 times = 0  
 **while** num > 0:  
 **if** num % 2 != 0:  
 times += 1  
 num /= 2  
 **return** times  
  
 @staticmethod  
 **def** \_\_ip\_and(gW, mask):  
 **return** int(gW) & int(mask)  
  
 *# gets the network attributes* @staticmethod  
 **def** \_\_get\_network\_attributes():  
 *"""  
 returns the computer's network attributes  
 """* wmi\_obj = wmi.WMI()  
 sql = **"select MACAddress, IPAddress, DefaultIPGateway, IPSubnet from Win32\_NetworkAdapterConfiguration where IPEnabled=TRUE"** wmi\_out = wmi\_obj.query(sql)  
 index = 0  
 **for** result **in** wmi\_out:  
 **if** is\_ip(result.IPAddress[0]):  
 index = wmi\_out.index(result)  
 my\_ip, gateway\_ip, my\_mac = wmi\_out[index].IPAddress[0], wmi\_out[index].DefaultIPGateway[0], wmi\_out[index].MACAddress  
 subnet\_mask = wmi\_out[index].IPSubnet[0]  
 **return** my\_ip, gateway\_ip, my\_mac, subnet\_mask  
*# endregion  
  
  
# checks if ip is valid***def** is\_ip(ip):  
 **return** re.match(IP\_REGULAR\_EXPRESSION, ip)  
  
*# region ---------------------------- MAIN ----------------------------  
  
  
# Prints an up to date arp table***def** main():  
 **print "scanning...\r\nIt will take a while."** arp\_table = NetMap.map()  
 **print "scan complete!"  
 for** computer **in** arp\_table:  
 **print** computer.mac + **" "** \* 4 + computer.ip + **" "** \* 4 + computer.computer\_name  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()  
*# endregion*

Process.py

*# region ---------------------------- ABOUT ----------------------------  
"""  
##################################################################  
# Created By: Ariel Westfried #  
# Date: 01/01/2016 #  
# Name: Project - Process #  
# Version: 1.0 #  
# Windows Tested Versions: Win 7 32-bit #  
# Python Tested Versions: 2.6 32-bit #  
# Python Environment : PyCharm #  
##################################################################  
"""  
# endregion  
  
# region ---------------------------- IMPORTS ----------------------------***from** Constants **import** \*  
*# endregion  
  
# region ---------------------------- Process CLASS ----------------------------***class** Process(object):  
 *# constructor* **def** \_\_init\_\_(self, name, pid, parent\_id):  
 self.name = name  
 self.pid = pid  
 self.parent\_id = parent\_id  
  
 *# to string* **def** \_\_str\_\_(self):  
 **return "Name: "** + self.name + END\_LINE + **"PID: "** + self.pid + END\_LINE + **"Parent ID: "** + self.pid  
  
 *# to string* **def** \_\_repr\_\_(self):  
 **return "Name: "** + self.name + END\_LINE + **"PID: "** + self.pid + END\_LINE + **"Parent ID: "** + self.pid  
  
 *# for use boolean statements* **def** \_\_eq\_\_(self, other):  
 **if** isinstance(other, Process):  
 **return** self.pid == other.pid  
 **else**:  
 **raise** ValueError  
*# endregion*

Server.py

*# region ---------------------------- ABOUT ----------------------------  
"""  
##################################################################  
# Created By: Ariel Westfried #  
# Date: 01/01/2016 #  
# Name: Project - Server #  
# Version: 1.0 #  
# Windows Tested Versions: Win 7 32-bit #  
# Python Tested Versions: 2.6 32-bit #  
# Python Environment : PyCharm #  
##################################################################  
"""  
# endregion  
  
# region ---------------------------- IMPORTS ----------------------------***from** Constants **import** \*  
**from** socket **import** socket, gethostbyname, gethostname, AF\_INET, SOCK\_DGRAM, SOL\_SOCKET, SO\_BROADCAST  
**from** threading **import** Thread, Lock  
**from** ComputerDatabase **import** ComputerDatabase  
**from** NetMap **import** NetMap  
**from** time **import** sleep  
**import** pythoncom  
**from** select **import** select  
**from** ClientInterface **import** ClientInterface, ClientList, Computer  
**from** datetime **import** datetime  
**import** subprocess  
**from** Cipher **import** Cipher  
**from** os.path **import** exists  
**from** os **import** makedirs  
*# endregion  
  
# region ---------------------------- Server CLASS ----------------------------***class** Server(object):  
 *# constructor* **def** \_\_init\_\_(self):  
 self.\_\_address = gethostbyname(gethostname())  
 self.\_\_main\_socket = socket()  
 self.\_\_database = ComputerDatabase()  
 self.\_\_database\_lock = Lock()  
 self.\_\_announce\_socket = socket(AF\_INET, SOCK\_DGRAM)  
 self.running = False  
 self.starting = False  
 self.\_connected\_clients = ClientList()  
 self.\_\_signature = None  
 self.\_\_public\_key = None  
  
 *# starts server - updates database and starts threads* **def** start(self):  
 *"""  
 Starts the server  
 """* **if** self.starting **or** self.running:  
 **raise** NameError(**'Already running or starting...'**)  
 self.starting = True  
 self.\_\_signature = Cipher.crate\_signature()  
 self.\_\_public\_key = Cipher()  
 self.\_\_print(**"Updating database..."**)  
 pythoncom.CoInitialize()  
 current\_arp = NetMap.map()  
 pythoncom.CoUninitialize()  
 database = self.\_\_database.read()  
 *# Loop for updating the state of the computer* **for** computer **in** database:  
 **if** computer.ip == gethostbyname(gethostname()):  
 computer.active = True  
 self.\_\_database.update\_state(computer)  
 **elif** computer **not in** current\_arp:  
 computer.active = False  
 self.\_\_database.update\_state(computer)  
 *# Loop for updating the database* **for** computer **in** current\_arp:  
 **if** computer **not in** database:  
 self.\_\_database.add\_row(computer)  
 database = self.\_\_database.read()  
 self.\_\_print(**"Database updated."**)  
 **if not** exists(DOWNLOAD\_UPLOAD):  
 makedirs(DOWNLOAD\_UPLOAD)  
 network\_scan\_thread = Thread(target=self.\_\_network\_scan)  
 network\_scan\_thread.setDaemon(True)  
 network\_scan\_thread.start()  
 broadcast\_announce\_thread = Thread(target=self.\_\_broadcast\_announce)  
 broadcast\_announce\_thread.setDaemon(True)  
 broadcast\_announce\_thread.start()  
 self.\_\_main\_socket.bind((**"0.0.0.0"**, SERVER\_PORT))  
 self.\_\_main\_socket.listen(1)  
 self.starting = False  
 self.running = True  
 self.\_\_run()  
  
 *# the main running method of the server* **def** \_\_run(self):  
 *"""  
 Actual main code of the server  
 """* self.\_\_print(**"Server started!"**)  
 **while** True:  
 to\_read, to\_write, error = select([self.\_\_main\_socket], [], [])  
 **for** sock **in** to\_read:  
 **if** sock **is** self.\_\_main\_socket:  
 client\_socket, client\_address = self.\_\_main\_socket.accept()  
 **for** computer **in** self.\_\_database.read():  
 **if** computer.ip == client\_address[0]:  
 new\_client\_thread = Thread(target=self.\_\_new\_client, args=[client\_socket, computer])  
 new\_client\_thread.setDaemon(True)  
 new\_client\_thread.start()  
  
 *# handles a newly connected client* **def** \_\_new\_client(self, client\_socket, computer):  
 key = self.\_\_key\_exchange(client\_socket)  
 **if** isinstance(key, Cipher):  
 self.\_connected\_clients.append(ClientInterface(client\_socket, computer, key))  
  
 *# Operates the key exchange with client* **def** \_\_key\_exchange(self, sock):  
 sock.send(self.\_\_public\_key.public\_key().pack())  
 data = sock.recv(BUFFER\_SIZE)  
 key = self.\_\_public\_key.decrypt(data)  
 key, his\_hashed\_key = key.split(IN\_PACK\_SEPARATOR)  
 **if** Cipher.hash(key) == his\_hashed\_key:  
 key = Cipher.unpack(key)  
 **return** key  
  
 *# Has it's own thread - announces the server address to the network* **def** \_\_broadcast\_announce(self):  
 *"""  
 Runs in a thread. Announces the server's existence in the network  
 """* message = SERVER\_ANNOUNCE\_MESSAGE  
 self.\_\_announce\_socket.setsockopt(SOL\_SOCKET, SO\_BROADCAST, 1)  
 **while** True:  
 self.\_\_announce\_socket.sendto(message, (**"<broadcast>"**, BROADCAST\_PORT))  
 sleep(ANNOUNCE\_SLEEP\_TIME)  
  
   
 **def** \_\_network\_scan(self):  
 *"""  
 Scans the network  
 """* **while** True:  
 sleep(NET\_SCAN\_WAIT)  
 pythoncom.CoInitialize()  
 current\_arp = NetMap.map()  
 pythoncom.CoUninitialize()  
 database = self.\_\_database.read()  
 **for** computer **in** database:  
 **if** computer.ip == gethostbyname(gethostname()):  
 computer.active = True  
 self.\_\_database.update\_state(computer)  
 **elif** computer **not in** current\_arp:  
 computer.active = False  
 self.\_\_database.update\_state(computer)  
 **for** computer **in** current\_arp:  
 **if** computer **not in** database:  
 self.\_\_database.add\_row(computer)  
 database = self.\_\_database.read()  
  
 **def** \_\_print(self, data):  
 **print** data  
  
 **def** do\_action(self, computer, hour, minute, second, action):  
 **if** isinstance(computer, Computer):  
 **for** other\_computer **in** self.\_\_database.read():  
 **if** other\_computer == computer:  
 now = datetime.now()  
 my\_time = datetime(now.year, now.month, now.day, int(hour), int(minute), int(second))  
 to\_wait = (my\_time - now).seconds  
 wait\_thread = Thread(target=self.\_\_wait\_to\_action, args=(to\_wait, action, computer))  
 wait\_thread.start()  
 **return  
  
 def** do\_action\_now(self, computer, action):  
 **if** isinstance(computer, Computer):  
 **for** other\_computer **in** self.\_\_database.read():  
 **if** other\_computer == computer:  
 self.\_\_wait\_to\_action(0, action, computer)  
 **return  
  
 def** \_\_wait\_to\_action(self, time\_to\_wait, action, computer):  
 sleep(time\_to\_wait)  
 **if** action == **"shutdown"**:  
 client = self.\_\_find\_client(computer)  
 **if** isinstance(client, ClientInterface):  
 self.\_connected\_clients.remove(client)  
 computer.shutdown()  
 **else**:  
 computer.wake\_up()  
 self.\_\_database.update\_state(computer)  
  
 **def** make\_computers\_dictionary(self):  
 dictionary = self.\_\_database.make\_dictionary()  
 dictionary[**"CONNECTED"**] = []  
 **for** i **in** xrange(len(dictionary[**"MAC"**])):  
 dictionary[**"CONNECTED"**].append(str(Computer(dictionary[**"MAC"**][i], dictionary[**"IP"**][i]) **in** self.\_connected\_clients))  
 **return** dictionary  
  
 **def** \_\_find\_client(self, computer):  
 **for** client **in** self.\_connected\_clients:  
 **if** computer == client:  
 **return** client  
  
 **def** computer\_data(self, computer):  
 client = self.\_\_find\_client(computer)  
 **return** {**"MAC"**: client.get\_mac(),  
 **"IP"**: client.get\_ip(),  
 **"HOST"**: client.name}  
  
 **def** get\_processes\_data(self, computer):  
 client = self.\_\_find\_client(computer)  
 **if** isinstance(client, ClientInterface):  
 **try**:  
 client.update\_processes()  
 **except**:  
 self.\_connected\_clients.remove(client)  
 **else**:  
 processes\_list = [dict(NAME=process.name, PID=process.pid, PARENT\_ID=process.parent\_id) **for** process **in** client.processes]  
 **return** processes\_list  
  
 **def** terminate\_process(self, computer, processes):  
 client = self.\_\_find\_client(computer)  
 **if** isinstance(client, ClientInterface):  
 **try**:  
 result = client.terminate(processes)  
 **except**:  
 self.\_connected\_clients.remove(client)  
 **else**:  
 **return** result  
  
 **def** open\_process(self, computer, command):  
 client = self.\_\_find\_client(computer)  
 **if** isinstance(client, ClientInterface):  
 **try**:  
 result = client.open\_process(command)  
 **except**:  
 self.\_connected\_clients.remove(client)  
 **else**:  
 **return** result  
  
 **def** get\_file(self, computer, directory):  
 client = self.\_\_find\_client(computer)  
 **if** isinstance(client, ClientInterface):  
 **try**:  
 result = client.send\_files(directory)  
 **except**:  
 self.\_connected\_clients.remove(client)  
 **else**:  
 **return** {  
 **'NAME'**: directory,  
 **'ITEMS'**: result  
 }  
  
 **def** delete\_file(self, computer, directory):  
 client = self.\_\_find\_client(computer)  
 **if** isinstance(client, ClientInterface):  
 **try**:  
 result = client.delete\_file(directory)  
 **except**:  
 self.\_connected\_clients.remove(client)  
 **else**:  
 **return** result  
  
 **def** create\_file(self, computer, path, name):  
 client = self.\_\_find\_client(computer)  
 **if** isinstance(client, ClientInterface):  
 **try**:  
 result = client.create\_file(path, name)  
 **except**:  
 self.\_connected\_clients.remove(client)  
 **else**:  
 **return** result  
  
 **def** and\_computer(self, computer):  
 **if** isinstance(computer, Computer):  
 computer.active = False  
 self.\_\_database.add\_row(computer)  
 **return "Computer successfully added."  
  
 def** download(self, computer, directory):  
 client = self.\_\_find\_client(computer)  
 **if** isinstance(client, ClientInterface):  
 file\_data = client.download(directory)  
 file\_name = directory.split(**'\\'**)[-1]  
 file\_dump = open(DOWNLOAD\_UPLOAD + **'\\'** + file\_name, **'wb+'**)  
 file\_dump.write(file\_data)  
 file\_dump.close()  
 **return** file\_name  
  
 **def** remote\_desktop(self, computer\_list):  
 **for** computer **in** computer\_list:  
 subprocess.Popen([**'mstsc'**, **'/v:'** + computer.ip])  
*# endregion  
  
# region ---------------------------- MAIN ----------------------------  
  
  
# runs server without GUI***def** main():  
 server = Server()  
 server.start()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()  
*# endregion*

WakeOnLan.py

**from** Constants **import** \*  
**from** socket **import** \*  
**from** struct **import** pack  
**from** win32api **import** GetCurrentProcess, InitiateSystemShutdown  
**from** win32con **import** SE\_SHUTDOWN\_NAME, SE\_REMOTE\_SHUTDOWN\_NAME, SE\_PRIVILEGE\_ENABLED, TOKEN\_ALL\_ACCESS  
**from** win32security **import** LookupPrivilegeValue, OpenProcessToken, AdjustTokenPrivileges  
  
  
**def** wake\_on\_lan(mac\_address):  
 *"""  
 Wakes a computer  
 """* **if** len(mac\_address) == MAC\_LENGTH + 5:  
 separator = mac\_address[2]  
 mac\_address = mac\_address.replace(separator, **''**)  
 **elif** len(mac\_address) != MAC\_LENGTH:  
 **raise** ValueError(**"Incorrect MAC address format."**)  
  
 data = **''**.join([**'FFFFFFFFFFFF'**, mac\_address \* 16])  
 send\_data = **''  
 for** i **in** range(0, len(data), 2):  
 send\_data = **''**.join([send\_data, pack(**'B'**, int(data[i: i + 2], 16))])  
  
 sock = socket(AF\_INET, SOCK\_DGRAM)  
 sock.setsockopt(SOL\_SOCKET, SO\_BROADCAST, 1)  
 sock.sendto(send\_data, (**'<broadcast>'**, WAKE\_ON\_LAN\_PORT))  
  
  
**def** shutdown(host=None, msg=None, timeout=0, force=1, reboot=0):  
 *"""  
 Shutdown a computer  
 """* host = gethostbyaddr(host)[0]  
 **try**:  
 privilege1 = LookupPrivilegeValue(host, SE\_SHUTDOWN\_NAME)  
 privilege2 = LookupPrivilegeValue(host, SE\_REMOTE\_SHUTDOWN\_NAME)  
 **except**:  
 **return "Error"  
 else**:  
 new\_state = [(privilege1, SE\_PRIVILEGE\_ENABLED), (privilege2, SE\_PRIVILEGE\_ENABLED)]  
 token = OpenProcessToken(GetCurrentProcess(), TOKEN\_ALL\_ACCESS)  
 AdjustTokenPrivileges(token, False, new\_state)  
 InitiateSystemShutdown(host, msg, timeout, force, reboot)  
 **return "Done"**

AddPage.html

<!DOCTYPE **html** PUBLIC **"-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"**>  
<**html xmlns="http://www.w3.org/1999/xhtml"**>  
<**head**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
 <**link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='stylesheets/MyStyle.css') }}"**>  
 <**title**>Network Manager</**title**>  
 <**script type="text/javascript"**>// <![CDATA[  
 function loading(){  
 $("#loading").show();  
 $("#content").hide();  
 }  
 // ]]></**script**>  
</**head**>  
<**body**>  
<**div id="loading"**></**div**>  
<**div id="content"**>  
 <**form method="get" action="{{ url\_for('show\_main\_form') }}"**><**input id="ButtonBack" type="submit" value="Back" onclick="loading();"**/></**form**>  
 <**p**><**strong**>{{ message }}</**strong**></**p**>  
 <**p**>Please fill the next form:</**p**>  
 <**table cellspacing='0' cellpadding='0'**> *<!-- cellspacing='0' is important, must stay -->* <**thead**><**tr**><**th**>Criteria</**th**><**th**>Data</**th**></**tr**></**thead**>  
 <**tbody**>  
 <**form method="post" action="{{ url\_for('add\_computer') }}"**>  
 <**tr class='even'**>  
 <**td**>Mac Address</**td**>  
 <**td**><**input class="MacText" type="text" value="" name="Mac"** /> </**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>IP Address</**td**>  
 <**td**><**input class="IpText" type="text" value="" name="Ip"** /> </**td**>  
 </**tr**>  
 </**form**>  
 </**tbody**>  
 <**input id="SubmitForm" type="submit" value="Submit" name="Action" onclick="loading();"**/>  
 </**table**>  
</**div**>  
</**body**>  
</**html**>

FilesPage.html

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='stylesheets/MyStyle.css') }}"** />  
 <**script type="text/javascript"**>// <![CDATA[  
 function loading(){  
 $("#loading").show();  
 $("#content").hide();  
 }  
 function alert(){  
 var r = confirm("Are you sure you want to delete the file?");  
 if (r == true) {  
 x = "You pressed OK!";  
 } else {  
 x = "You pressed Cancel!";  
 }  
 }  
  
 // ]]></**script**>  
</**head**>  
<**body**>  
<**div id="loading"**></**div**>  
<**div id="content"**>  
{% if 'Drives' in directory['NAME'] %}  
<**form method="get" action="{{ url\_for('view\_computer', mac=mac, ip=ip, name=name) }}"**><**input id="ButtonBack1" type="submit" value="Back" onclick="loading();"**/></**form**>  
{% else %}  
<**form method="get" action="{{ url\_for('back', mac=mac, ip=ip, name=name, path=directory['NAME']) }}"**><**input id="ButtonBack2" type="submit" value="Back" onclick="loading();"**/></**form**>  
{% endif %}  
<**p**><**strong**>{{ message }}</**strong**></**p**>  
<**p**><**strong**>File Tree for {{ name + " (" + mac + " " + ip + ")" }}:</**strong**></**p**>  
<**p**>{{ directory['NAME'] }}</**p**>  
{% if directory['ITEMS'] %}  
<**p**>\*If you want to create new file, write its name instead the 'new folder' and don't forget the suffix</**p**>  
<**p**>example: new.txt</**p**>  
<**table cellpadding="0" cellspacing="0"**>  
 <**thead**><**tr**><**th**>File Name</**th**><**th**>options</**th**></**tr**></**thead**>  
 <**tbody**>  
 {% for item in directory['ITEMS'] %}  
 <**form method="post" action="{{ url\_for('show\_files', mac=mac, ip=ip, name=name, path=directory['NAME']) }}"**>  
 <**tr class="even"**>  
 <**td**><**input class="ProcessText" type="text" value="{{ item }}" name="FileName"** /></**td**>  
 <**td**>  
 {% if item.startswith('.') or '.' not in item %}  
 <**input id="ButtonOpen" type="submit" value="Open" name="Action" onclick="loading();"**/>  
 {% else %}  
 <**a href="{{ url\_for('download\_file', mac=mac, ip=ip, directory=directory['NAME'], item=item) }}" target="\_blank"**>Download</**a**>  
 {% endif %}  
 <**input id="ButtonDelete" type="submit" value="Delete" name="Action" onclick="alert();"**/>  
 </**td**>  
 </**tr**>  
 </**form**>  
 {% endfor %}  
 {% if ':' not in directory['ITEMS'][0] %}  
 <**form method="post" action="{{ url\_for('show\_files', mac=mac, ip=ip, name=name, path=directory['NAME']) }}"**>  
 <**tr class="even"**>  
 <**td**><**input class="ProcessText" type="text" value="new folder" name="FileName"**/></**td**>  
 <**td**><**input id="NewButton" type="submit" value="Create" name="Action" onclick="loading();"**/></**td**>  
 </**tr**>  
 </**form**>  
 </**tbody**>  
 {% endif %}  
</**table**>  
{% endif %}  
 </**div**>  
</**body**>  
</**html**>

InfoPage.html

<!DOCTYPE **html** PUBLIC **"-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"**>  
<**html xmlns="http://www.w3.org/1999/xhtml"**>  
<**head**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
 <**link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='stylesheets/MyStyle.css') }}"**>  
 <**title**>Network Manager</**title**>  
 <**script type="text/javascript"**>// <![CDATA[  
 function loading(){  
 $("#loading").show();  
 $("#content").hide();  
 }  
 // ]]></**script**>  
</**head**>  
<**body**>  
<**div id="loading"**></**div**>  
<**div id="content"**>  
 <**form method="get" action="{{ url\_for('show\_main\_form') }}"**><**input id="ButtonBack" type="submit" value="Back" onclick="loading();"**/></**form**>  
 <**p**><**strong**>{{ message }}</**strong**></**p**>  
 <**p**>Computer:</**p**>  
 <**table cellspacing='0' cellpadding='0'**> *<!-- cellspacing='0' is important, must stay -->* <**thead**><**tr**><**th**>Criteria</**th**><**th**>Data</**th**></**tr**></**thead**>  
 <**tbody**>  
 <**form method="post" action="{{ url\_for('view\_computer', mac=computer['MAC'], ip=computer['IP']) }}"**>  
 <**tr class='even'**>  
 <**td**>Host Name</**td**>  
 <**td**><**input class="IpText" type="text" value="{{ computer['HOST'] }}" name="Host" readonly** /> </**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Mac Address</**td**>  
 <**td**><**input class="MacText" type="text" value="{{ computer['MAC'] }}" name="Mac" readonly** /> </**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>IP Address</**td**>  
 <**td**><**input class="IpText" type="text" value="{{ computer['IP'] }}" name="Ip" readonly** /> </**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Files</**td**>  
 <**td**><**input id="ButtonFiles" type="submit" value="Show Files" name="Action" onclick="loading();"**/></**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Processes</**td**>  
 <**td**><**input id="ButtonProcesses" type="submit" value="Show Processes" name="Action" onclick="loading();"**/></**td**>  
 </**tr**>  
 <**tr**>  
 <**td**><**input id="InputText" type="text" name="ProcessName"**/></**td**>  
 <**td**><**input id="ButtonOpen" type="submit" value="Open Process" name="Action" onclick="loading();"**/></**td**>  
 </**tr**>  
 </**form**>  
 </**tbody**>  
 </**table**>  
 </**div**>  
</**body**>  
</**html**>

Loading.html

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**meta http-equiv="refresh" content="1"** >  
 <**title**>Please Wait</**title**>  
</**head**>  
<**body**>  
Server is starting, this might take a while...  
</**body**>  
</**html**>

MainPage.html

<!DOCTYPE **html** PUBLIC **"-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"**>  
<**html xmlns="http://www.w3.org/1999/xhtml"**>  
<**head**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
 <**link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='stylesheets/MyStyle.css') }}"**>  
 <**title**>Network Manager</**title**>  
 <**script type="text/javascript"**>// <![CDATA[  
 function loading(){  
 $("#loading").show();  
 $("#content").hide();  
 }  
 // ]]></**script**>  
 <**meta http-equiv="refresh" content="60"** >  
</**head**>  
<**body**>  
<**div id="loading"**></**div**>  
<**div id="content"**>  
<**p**>Computers:</**p**>  
<**form method="post" action="{{ url\_for('show\_main\_form' ) }}"**>  
 <**div class="container"**><**table cellspacing='0' cellpadding='0'**> *<!-- cellspacing='0' is important, must stay -->* <**thead**>  
 <**tr**><**th**>CHECK</**th**><**th**>IP</**th**><**th**>MAC</**th**><**th**>Status</**th**><**th**>Operations</**th**></**tr**>*<!-- Table Header -->* </**thead**>  
 <**tbody**>  
 {% for row in computers %}  
 <**tr class='even'**>  
 <**td**><**input type="checkbox" name="check" value="{{ row['MAC'] }}\_{{ row['IP'] }}"**/></**td**>  
 <**form method="post" action="{{ url\_for('show\_main\_form') }}"**>  
 <**td**><**input class="IpText" type="text" value="{{ row['IP'] }}" name="Ip" readonly** /> </**td**>  
 <**td**><**input class="MacText" type="text" value="{{ row['MAC'] }}" name="Mac" readonly** /> </**td**>  
 <**td**><**input class="StatusText" type="text" value="{{ row['STATUS'] }}" name="Status" readonly** /> </**td**>  
 <**td**>  
 {% if row['CONNECTED'] == 'True'%}  
 <**input id="Information" type="submit" value="More" name="Action" onclick="loading();"**/>  
 {% endif %}  
 </**td**>  
 </**form**>  
 </**tr**>  
 {% endfor %}  
 </**tbody**>  
 </**table**>  
 </**div**>  
 <**p**> <**input id="ButtonWOL" type="submit" value="WakeOnLAN" name="Action" onclick="loading();"**/></**p**>  
 <**p**> <**input id="ButtonShutDown" type="submit" value="Shutdown" name="Action" onclick="loading();"**/></**p**>  
 <**p**> <**input id="ButtonManually" type="submit" value="Add Computer Manually" name="Action" onclick="loading();"**/></**p**>  
 <**p**> <**input id="RemoteDesktop" type="submit" value="Remote Desktop" name="Action" onclick="loading();"**/></**p**>  
</**form**>  
 </**div**>  
</**body**>  
</**html**>

ProcessesPage.html

<!DOCTYPE **html** PUBLIC **"-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"**>  
<**html xmlns="http://www.w3.org/1999/xhtml"**>  
<**head**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
 <**link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='stylesheets/MyStyle.css') }}"**>  
 <**title**>Network Manager</**title**>  
 <**script type="text/javascript"**>// <![CDATA[  
 function loading(){  
 $("#loading").show();  
 $("#content").hide();  
 }  
 // ]]></**script**>  
</**head**>  
<**body**>  
<**div id="loading"**></**div**>  
<**div id="content"**>  
<**form method="get" action="{{ url\_for('view\_computer', mac=mac, ip=ip, name=name) }}"**><**input id="ButtonBack" type="submit" value="Back" onclick="loading();"**/></**form**>  
<**p**>Processes for {{ name }}({{ mac }} - {{ip}}):</**p**>  
<**p**><**strong**>{{ message }}</**strong**></**p**>  
<**form method="post" action="{{ url\_for('show\_processes', mac=mac, ip=ip, name=name) }}"**>  
 <**table cellspacing='0' cellpadding='0'**> *<!-- cellspacing='0' is important, must stay -->* <**thead**><**tr**><**th**></**th**><**th**>Process Name</**th**><**th**>PID</**th**><**th**>Parent ID</**th**></**tr**></**thead**>  
 <**tbody**>  
 {% for process in process\_list %}  
 <**tr class='even'**>  
 <**td class="td\_check"**><**input type="checkbox" name="check" value="{{ process['NAME']}}\_{{ process['PID'] }}\_{{ process['PARENT\_ID'] }}"**/></**td**>  
 <**td**>{{ process['NAME'] }}</**td**>  
 <**td**>{{ process['PID'] }}</**td**>  
 <**td**>{{ process['PARENT\_ID'] }}</**td**>  
 </**tr**>  
 {% endfor %}  
 </**tbody**>  
 </**table**>  
 <**input id="ButtonTerminate" type="submit" value="Terminate" name="Action" onclick="loading();"**/>  
</**form**>  
 </**div**>  
</**body**>  
</**html**>

WakeOnLan.html

<!DOCTYPE **html** PUBLIC **"-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"**>  
<**html xmlns="http://www.w3.org/1999/xhtml"**>  
<**head**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
 <**link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='stylesheets/MyStyle.css') }}"**>  
 <**title**>Network Manager</**title**>  
 <**script type="text/javascript"**>// <![CDATA[  
 function loading(){  
 $("#loading").show();  
 $("#content").hide();  
 }  
 // ]]></**script**>  
</**head**>  
<**body**>  
<**div id="loading"**></**div**>  
<**div id="content"**>  
 <**form method="get" action="{{ url\_for('show\_main\_form') }}"**><**input id="ButtonBack" type="submit" value="Back" onclick="loading();"**/></**form**>  
 <**p**><**strong**>{{ message }}</**strong**></**p**>  
 <**p**>{{ action }} computers: {{ names }}</**p**>  
 <**form method="post" action="{{ url\_for('wake\_on\_lan', computer\_list=computer\_list, action=action) }}"**>  
 <**table cellspacing='0' cellpadding='0'**> *<!-- cellspacing='0' is important, must stay -->* <**thead**>  
 <**tr**><**th**>Criteria</**th**><**th**>Data</**th**></**tr**> *<!-- Table Header -->* </**thead**>  
 <**tbody**>  
 <**tr class='even'**>  
 <**td**>Time to {{ action }}:</**td**>  
 <**td**>  
 <**input class="TimeText" type="text" value="hh" name="hour"** />:  
 <**input class="TimeText" type="text" value="mm" name="minute"** />:  
 <**input class="TimeText" type="text" value="ss" name="second"** />  
 </**td**>  
 </**tr**>  
 </**tbody**>  
 </**table**>  
 <**p**><**input id="ButtonGo1" type="submit" name="Go" value="{{ action }} now" onclick="loading();"**/></**p**>  
 <**p**><**input id="ButtonGo2" type="submit" name="Go" value="{{ action }} in specific time" onclick="loading();"**/></**p**>  
 </**form**>  
</**div**>  
</**body**>  
</**html**>

MyStyle.css

@table\_width: 800px;  
@column\_text\_width: 300px;  
@column\_button\_width: 150px;  
@column\_check\_width: 50px;  
  
table {  
 background: #f5f5f5;  
 border-collapse: separate;  
 box-shadow: inset 0 1px 0 #fff;  
 font-size: 12px;  
 line-height: 24px;  
 margin: 30px auto;  
 text-align: left;  
 width: table\_width;  
 float:left;  
  
}  
  
th {  
 background: url(http://jackrugile.com/images/misc/noise-diagonal.png), linear-gradient(#777, #444);  
 border-left: 1px solid #555;  
 border-right: 1px solid #777;  
 border-top: 1px solid #555;  
 border-bottom: 1px solid #333;  
 box-shadow: inset 0 1px 0 #999;  
 color: #fff;  
 width: 100px;  
 font-weight: bold;  
 padding: 10px 15px;  
 position: relative;  
 text-shadow: 0 1px 0 #000;  
}  
  
td {  
 border-right: 1px solid #fff;  
 border-left: 1px solid #e8e8e8;  
 border-top: 1px solid #fff;  
 border-bottom: 1px solid #e8e8e8;  
 padding: 10px 15px;  
 position: relative;  
 transition: all 300ms;  
 width: 100px;  
}  
  
td\_check {  
 border-right: 1px solid #fff;  
 border-left: 1px solid #e8e8e8;  
 border-top: 1px solid #fff;  
 border-bottom: 1px solid #e8e8e8;  
 padding: 10px 15px;  
 position: relative;  
 transition: all 300ms;  
 width: 20px;  
}  
  
td:first-child {  
 box-shadow: inset 1px 0 0 #fff;  
}  
  
td:last-child {  
 border-right: 1px solid #e8e8e8;  
 box-shadow: inset -1px 0 0 #fff;  
}  
  
tr {  
 background: url(http://jackrugile.com/images/misc/noise-diagonal.png);  
}  
  
tr:nth-child(odd) td {  
 background: #f1f1f1 url(http://jackrugile.com/images/misc/noise-diagonal.png);  
}  
  
tr:last-of-type td {  
 box-shadow: inset 0 -1px 0 #fff;  
}  
  
tr:last-of-type td:first-child {  
 box-shadow: inset 1px -1px 0 #fff;  
}  
  
tr:last-of-type td:last-child {  
 box-shadow: inset -1px -1px 0 #fff;  
}  
  
  
table tbody,  
table thead { display: block; }  
  
thead tr th {  
 height: 40px;  
 line-height: 30px;  
}  
  
table tbody {  
 height: 650px;  
 overflow-y: auto;  
 overflow-x: hidden;  
}  
  
tbody { border-top: 2px solid black; }  
  
tbody td, thead th {  
 width: 200px; */\* Optional \*/* border-right: 1px solid black;  
 */\* white-space: nowrap; \*/*}  
  
tbody td:last-child, thead th:last-child {  
 border-right: none;  
}  
  
.IpText{  
 background: #333;  
 color: #ccc;  
 width: 100px;  
 height: 20px;  
 padding: 6px 15px 6px 35px;  
 border-radius: 20px;  
 box-shadow: 0 1px 0 #ccc inset;  
 transition: 500ms all ease;  
 outline: 0;  
}  
.IpText:hover {  
 color: #fff;  
}  
.MacText{  
 background: #333;  
 color: #ccc;  
 width: 110px;  
 height: 20px;  
 padding: 6px 15px 6px 35px;  
 border-radius: 20px;  
 box-shadow: 0 1px 0 #ccc inset;  
 transition: 500ms all ease;  
 outline: 0;  
}  
.MacText:hover {  
 color: #fff;  
}  
  
.StatusText{  
 background: #333;  
 color: #ccc;  
 width: 50px;  
 height: 20px;  
 padding: 6px 15px 6px 35px;  
 border-radius: 20px;  
 box-shadow: 0 1px 0 #ccc inset;  
 transition: 500ms all ease;  
 outline: 0;  
}  
.StatusText:hover {  
 color: #fff;  
}  
  
DIV.table  
{  
 display: table;  
}  
  
FORM.tr, DIV.tr  
{  
 display: table-row;  
}  
SPAN.td  
{  
 display: table-cell;  
}  
  
#InputText {  
 background: #333;  
 color: #ccc;  
 width: 100px;  
 padding: 6px 15px 6px 35px;  
 border-radius: 20px;  
 box-shadow: 0 1px 0 #ccc inset;  
 transition: 500ms all ease;  
 outline: 0;  
}  
  
#InputText:hover {  
 width: 250px;  
}   
  
.ProcessText{  
 background: #333;  
 color: #ccc;  
 width: 110px;  
 height: 20px;  
 padding: 6px 15px 6px 35px;  
 border-radius: 20px;  
 box-shadow: 0 1px 0 #ccc inset;  
 transition: 500ms all ease;  
 outline: 0;  
}  
.ProcessText:hover {  
 width: 250px;  
}  
  
.TimeText{  
 background: #333;  
 color: #ccc;  
 width: 20px;  
 height: 20px;  
 padding: 5px 10px 5px 15px;  
 border-radius: 20px;  
 box-shadow: 0 1px 0 #ccc inset;  
 transition: 500ms all ease;  
 outline: 0;  
}  
.TimeText:hover {  
 color: #fff;  
}  
  
div#loading {  
 width: 35px;  
 height: 35px;  
 display: none;  
 background: {{ url\_for(**'static'**, filename=**'loadingimage.gif'**) }} no-repeat;  
 cursor: wait;  
 }