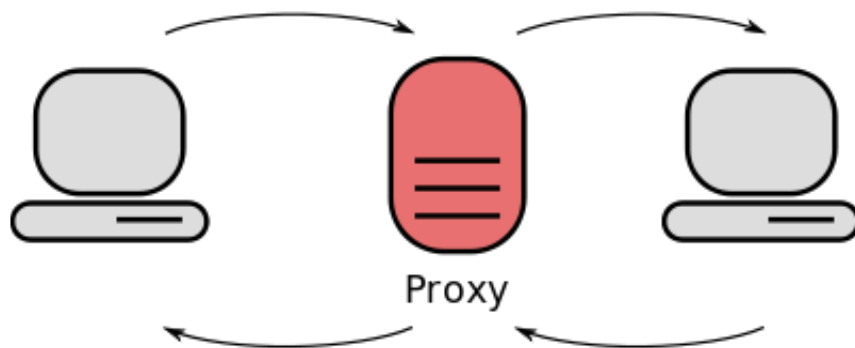


HTTP Proxy

With management capability



מגיש - עופר בר, 207943366

מקצוע - הגנת סייבר

שמות המנחים - אלון בר לב ושרית לולב

מרכז חינוך ליאו-באק

יוני 2017

תוכן עניינים

| | |
|----|---------------------|
| 1 | תוכן עניינים |
| 2 | מבוא |
| 3 | ארכיטקטורה |
| 3 | בקשת GET |
| 4 | בקשת CONNECT |
| 5 | גישה לממשק המשתמש |
| 6 | דיאגרמת רצף |
| 8 | רקע תיאורטי |
| 11 | מימוש |
| 11 | דיאגרמת בלוקים |
| 12 | Poller |
| 12 | HttpServer |
| 16 | ProxySocket |
| 16 | DirectSocketUp |
| 17 | Cache |
| 18 | מבני נתונים נוספים |
| 20 | בעיות ידועות |
| 21 | התקנה ותפעול |
| 21 | הכנות מקדימות |
| 22 | הרצת התכנית |
| 22 | ארגומנטים |
| 23 | ממשק גרפי |
| 24 | מסמך Log |
| 24 | קובץ README.md |
| 27 | תוכניות לעתיד |
| 28 | פרק אישי |
| 29 | קוד הפרויקט ותיעודו |
| 30 | נספחים |

מבוא

המוצר אותו פיתחתי במסגרת פרויקט סיום זה הוא HTTP proxy, עם יכולות ניהול.

שרת פרוקסי הוא שרת המשמש להעברת תקשורת בין שני רכיבים, לרוב לקוח ושרת. שרת הפרוקסי מחובר אל שני הרכיבים, ודרכו עוברות כל הבקשות לדפים באינטרנט. באופן זה, הפרוקסי יכול לנתח את הבקשות והתשובות, ולבצע מגוון פעולות, לדוגמא, שמירת קבצי מטמון וחסימת אתרים מסוימים.

שרת הפרוקסי שפיתחתי בפרויקט זה, תומך בבקשות GET ו-CONNECT, בפרוטוקול HTTP. השרת מעביר כל בקשה ותשובה שעוברות דרכו, ומאפשר שמירת קבצי מטמון (cache), כלומר שמירת דפים שבוקשו בעבר, ובכך חוסך בתעבורת הרשת ומאפשר גישה מהירה לדפים שמאפשרים זאת.

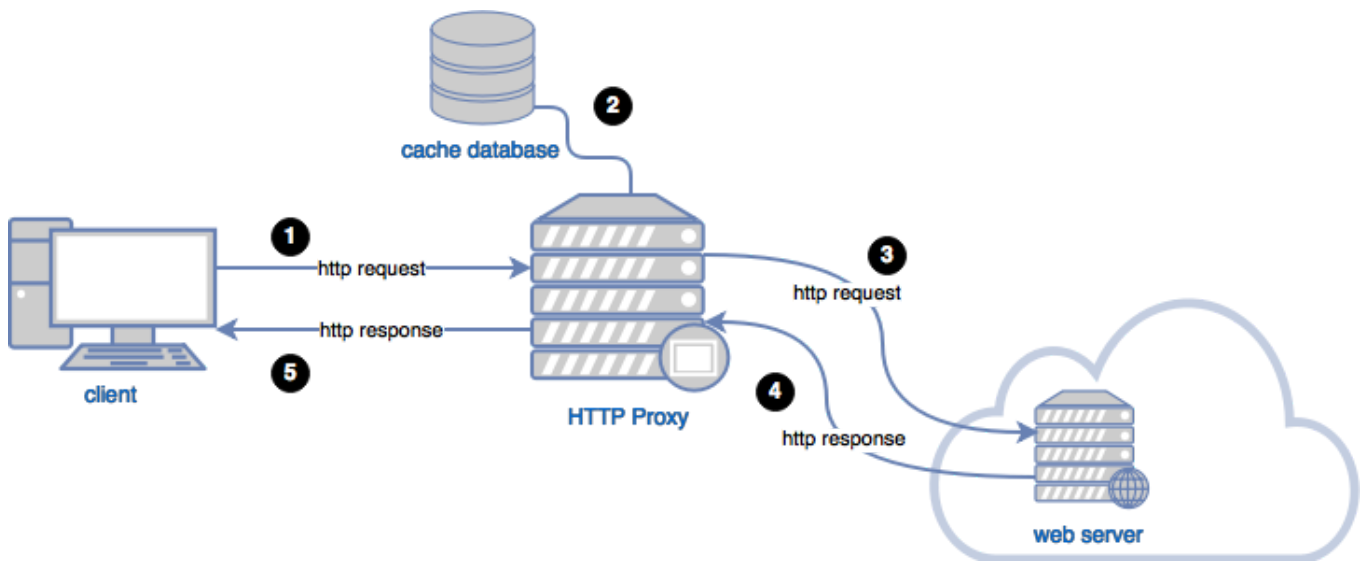
בנוסף, ניתן לגשת אל ממשק ניהול, בו ניתן לראות את קבצי המטמון השמורים, לראות את מספר ה-cache hits, כלומר כמה פעמים נעשה שימוש בקובץ הcache, ולמחוק אותם. בנוסף ישנו נתון המציג באופן קבוע את קצב שליחת וקבלת הבייטים בעשר השניות האחרונות (Throughput). במסמך זה ניתן לראות את ארכיטקטורת הפתרון, למצוא הסברים על אופן פעולת התוכנית, והוראות להתקנתה ולהפעלתה.

ארכיטקטורה

בפרויקט קיימים שלושה מקרים אפשריים, ולכל אחד מהם דרך פעולה מעט שונה:

בקשת GET

לאחר הגדרת שרת proxy בצד הclient, כל בקשות ה-GET בפרוטוקול HTTP המגיעות ממנו נשלחות תחילה אל הפרוקסי. הפרוקסי מנתח את הבקשה, ובודק האם הדף המבוקש שמור אצלו ב-cache, ובתוקף. במידה והדף זמין, הוא מחזיר אותו אל ה-client, ללא התחברות לשרת היעד. במידה והדף לא זמין ב-cache, שרת הפרוקסי מתחבר את שרת היעד, ושולח לו את הבקשה. עם קבלת התשובה מהשרת, הפרוקסי מנתח אותה, ובודק האם ניתן לשמור אותה ב-cache. במידה והתשובה מאפשרת את שמירתה ב-cache היא נשמרת, והתשובה נשלחת חזרה אל ה-client.

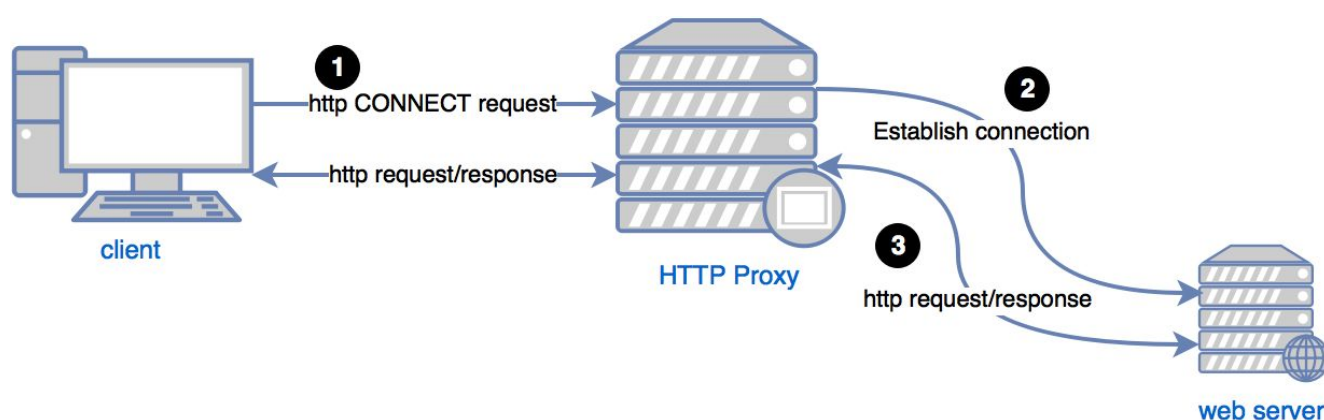


| הסבר | |
|------|--|
| 1 | הלקוח שולח בקשה אשר נשלחת לשרת הפרוקסי, שם היא מנותחת. |
| 2 | שרת הפרוקסי בודק האם התשובה מאוחסנת ב-cache ובתוקף, במידה וכן מדלג על שלבים 3 ו-4, ועובר לשלב 5. |
| 3 | שרת הפרוקסי שולח את הבקשה אל שרת היעד. |

| | |
|---|--|
| 4 | שרת הפרוקסי מקבל את התשובה של שרת היעד ומנתח אותה. במידה ואפשר לשמור את התשובה ב-cache, התשובה תשמר. |
| 5 | שרת הפרוקסי מחזיר את התשובה אל הלקוח. |

בקשת CONNECT

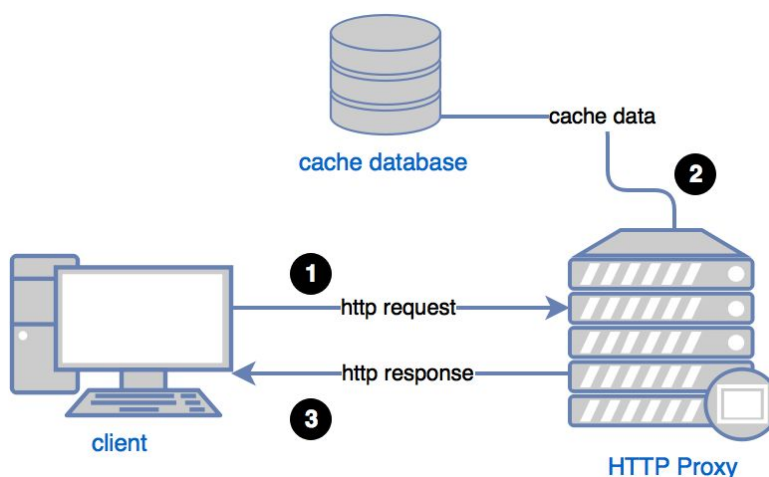
במידה ומתקבלת בקשת CONNECT בפרוטוקול HTTP מה-client, שרת הפרוקסי מתחבר אל שרת היעד ומעביר את הבקשה אליו ללא ניתוחה, וללא שימוש ב-cache. עם קבלת התשובה, הוא מחזיר אותה ישירות אל ה-client, באופן "שקוף".



| הסבר | |
|------|---|
| 1 | הלקוח שולח בקשה אשר נשלחת לשרת הפרוקסי, שם היא מנותחת, ומתגלה כבקשת CONNECT. |
| 2 | שרת הפרוקסי יוצר חיבור עם הכתובת המצוינת בבקשה שהתקבלה. |
| 3 | ה-client ושרת היעד מעבירים בקשות ותשובות, אשר עוברות דרך שרת הפרוקסי אך לא מנותחות או נשמרות. |

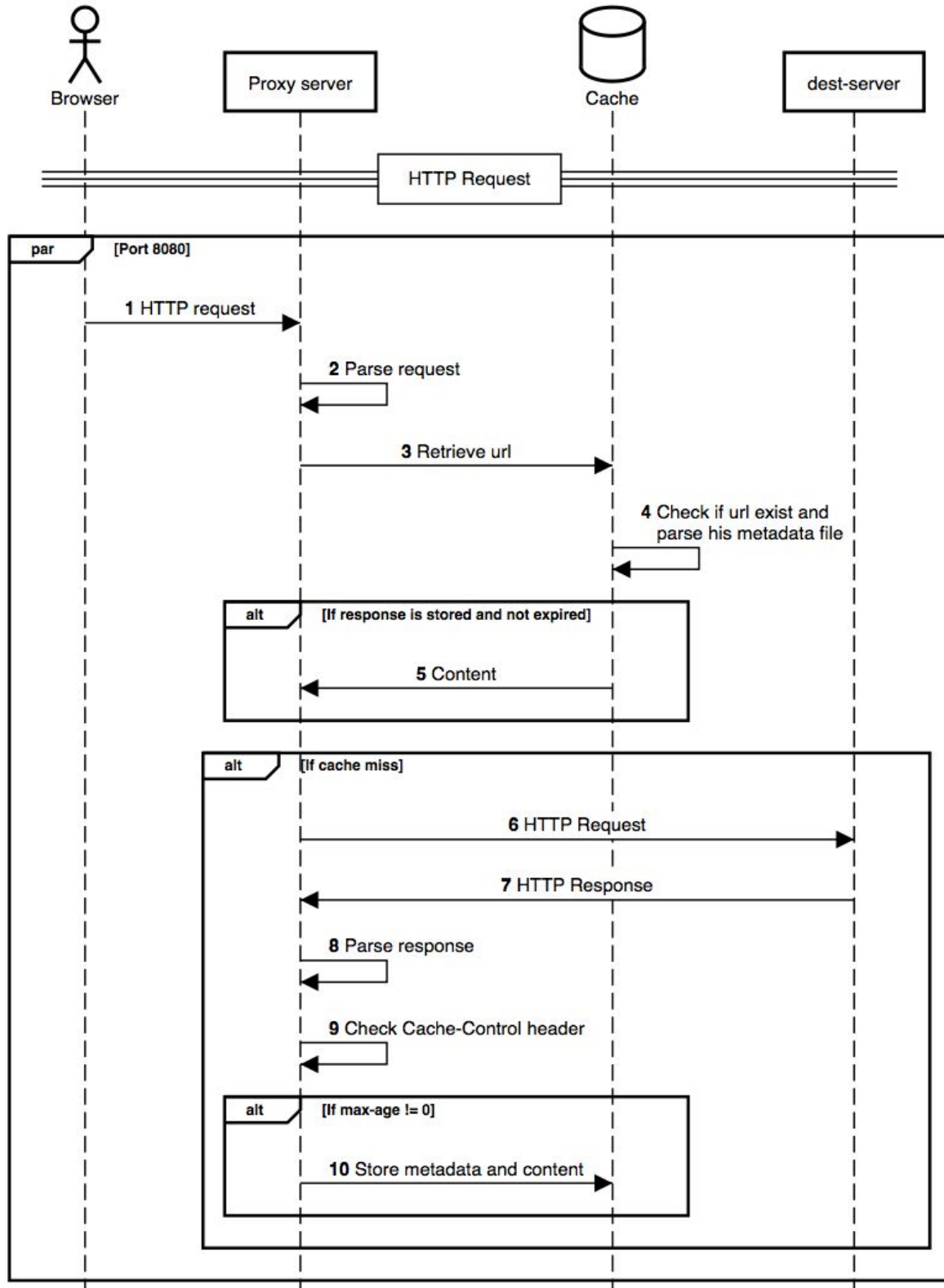
גישה לממשק המשתמש

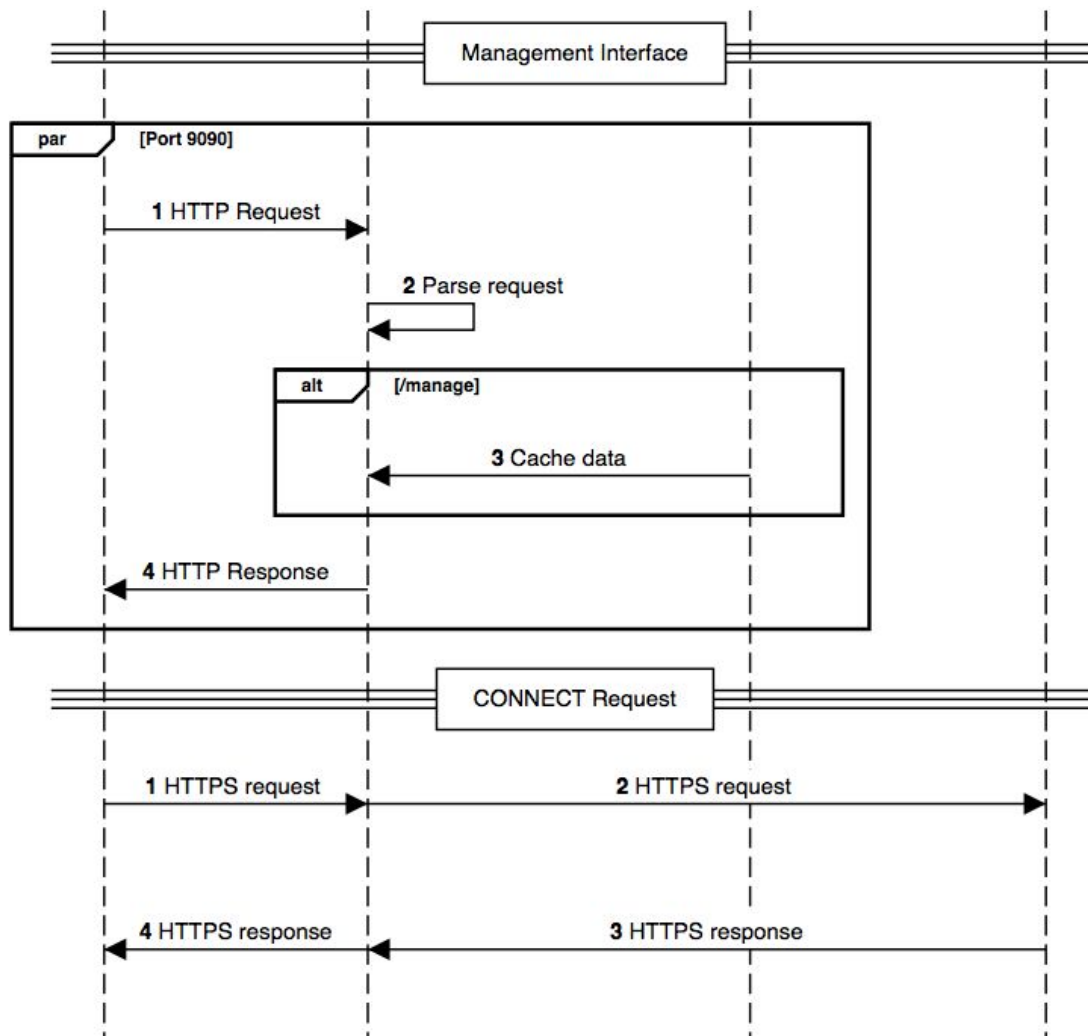
כאשר מתקבלת בקשה המבקשת להתחבר לפורט עליו רץ הסרבר, האחראי על שליחת ממשק הניהול, ולקבל ממנו את ממשק הניהול, שרת הפרוקסי מקבל את הנתונים העדכניים לגבי קבצי ה-cache השמורים, ושולח את דף ממשק הניהול ללקוח.



| הסבר | |
|------|---|
| 1 | הלקוח שולח בקשה לשרת הפרוקסי, לפורט האחראי על שליחת קבצים מקומיים, ומבקש את ממשק הניהול. שרת הפרוקסי מנתח את הבקשה. |
| 2 | שרת הפרוקסי מקבל את הנתונים העדכניים לגבי השימוש בקבצי ה-cache. |
| 3 | שרת הפרוקסי מחזיר ללקוח את דף ניהול המשתמש המבוקש. |

HTTP Proxy: Sequences





רקע תיאורטי

תיאור והסבר מושגים וטכנולוגיות שנעשה בהן שימוש בפרויקט:

[HTTP Proxy](#) - שרת שתפקידו לחבר בין לקוח לשרת יעד מסוים, כאשר הוא משמש כמתווך ביניהם. כלומר, הלקוח מתחבר לשרת הפרוקסי ושולח אליו את המידע והיעד מקבל את המידע הזה משרת הפרוקסי ולהיפך.

תפקידו העיקרי של שרת הפרוקסי הוא לספק גישה מהירה למשאבים חיצוניים ברשת מחשבים (באמצעות שימוש בקבצי מטמון). על ידי חיבור לשרת Proxy, ניתן להסוות את כתובת ה-IP בה משתמשים, מכיון שכל הבקשות נשלחות דרך שרת הפרוקסי.

מטרת הפרויקט היא מימוש HTTP Proxy.

[HTTP - \(Hypertext Transfer Protocol\)](#) - פרוטוקול תקשורת שנועד להעברת דפי HTML ואובייקטים שהם מכילים (כמו תמונות, קובצי קול, סרטוני פלאש וכו') ברשת האינטרנט. הפרוטוקול פועל בשכבת היישום של מודל ה-OSI ובשכבת היישום של מודל TCP/IP. שרתי HTTP הם שרתי התוכן המרכזיים ברשת האינטרנט ודפדפנים הם תוכנות הלקוח הנפוצות ביותר לפרוטוקול HTTP.

שני סוגי בקשות HTTP הנתמכות בפרויקט:

- GET - מיועדת לקבלת אובייקט שנמצא על השרת, בכתובת שניתנת בתחילת ההודעה. בקשות GET הן הנפוצות ביותר ברשת האינטרנט.
- CONNECT - בקשה זו יוצרת תקשורת דו כיוונית עם הכתובת המבוקשת, והיא יכולה לשמש ליצירת tunnel. לדוגמא, בקשות CONNECT יכולות לשמש לגישה לאתרים המשתמשים ב-SSL (HTTPS).

כל התקשורת בפרויקט מבוססת על פרוטוקול זה.

[תקשורת אסינכרונית - Async I/O](#) - גישה לטיפול ב-IO המאפשרת לשלוח מסרים בתוך ערוץ תקשורת מבלי לחכות שהצד המקבל מוכן, ומבלי לתקוע את התכנית. באמצעות השימוש במערכת אסינכרונית קיימת אפשרות לעבודה מקבילית הן של שליחה והן של קבלת הודעות ממספר חיבורים, ובכך ליצור תקשורת רבת משתמשים תוך שימוש בתהליך יחיד.

בפרויקט זה כל התקשורת היא אסינכרונית, ובכך היא מאפשרת עבודה במקביל עם חיבורים רבים, ללא תקיעת התוכנית.

Cache - מטמון - אוסף נתונים על בסיס ערכים מקוריים אשר מאוחסנים במיקום אחר, או שהופקו קודם לכן באמצעות חישוב כלשהו. השימוש במטמון מאפשר שליפה מחודשת של המידע במהירות במקום לחזור אל המאגר המקורי שהוא יחסית איטי או מרוחק.

שרת הפרוקסי בפרויקט שומר קבצי מטמון.

תכנות מונחה אירועים - Event driven development - תפיסה בתכנות, אשר על פיה בתוך תוכנית המחשב קיימים חלקים, הממתינים לקבלת אות. האות נקרא "אירוע" (event) והוא מתקבל כאשר מתרחש אירוע מסוים במערכת, אליו קשוב היישום.

POSIX - ("ממשק מערכת הפעלה תואם", "Portable Operating System Interface") אוסף תקנים של IEEE, המיועדים לשמירה על תאימות בין מערכות הפעלה ובמיוחד מערכות דמויות יוניקס.

HTTPS - (Hypertext Transfer Protocol Secure) - פרוטוקול תקשורת נפוץ במיוחד לאבטחת מידע ברשתות מחשבים, ומיושם במיוחד באינטרנט. באופן רשמי, הוא אינו פרוטוקול תקשורת כשלעצמו, אלא שימוש בפרוטוקול HTTP על שכבת SSL/TLS ובכך מקנה יכולות אבטחה של סטנדרט SSL/TLS לתקשורת HTTP רגילה.

Socket - נקודת קצה (endpoint) עבור זרם נתונים בתקשורת בין תהליכים על גבי רשת מחשבים.

Port - הוא תהליך ספציפי שדרכו יכולות תוכנות להעביר נתונים באופן ישיר. השימוש הנפוץ ביותר בפורט הוא בתקשורת מחשבים במסגרת הפרוטוקולים הנפוצים בשכבת התעבורה: TCP ו-UDP. פורט מזוהה לכל כתובת או פרוטוקול מסוים על ידי מספר באורך 16 ביטים היוצר 65536 כתובות אפשריות ל-UDP ו-65535 כתובות אפשריות ל-TCP. כתובת זו נקראת "מספר הפורט".

IP address - שדה מספרי באורך קבוע המשמש לזיהוי יחיד של נקודת קצה כלשהי המתקשרת בפרוטוקול התקשורת IP.

TCP - (Transmission Control Protocol) - פרוטוקול בתקשורת נתונים הפועל בשכבות התעבורה של מודל ה-OSI ובמודל ה-TCP/IP, ומבטיח העברה אמינה של נתונים בין שתי תחנות ברשת מחשבים באמצעות יצירת חיבור מקושר (Connection Oriented).

Python - שפת תכנות דינמית נפוצה. תוכננה לצורך שימוש בפשטות במבני נתונים מסובכים.

HTML - שפת תגיות לתצוגה ועיצוב של דפי אינטרנט בדפדפן.

Git - מערכת ניהול גרסאות מבוזרת.

[Doxygen](#) - מחולל תיעוד, כלי ליצירת תיעוד מקושר ודינמי של פרויקטים.

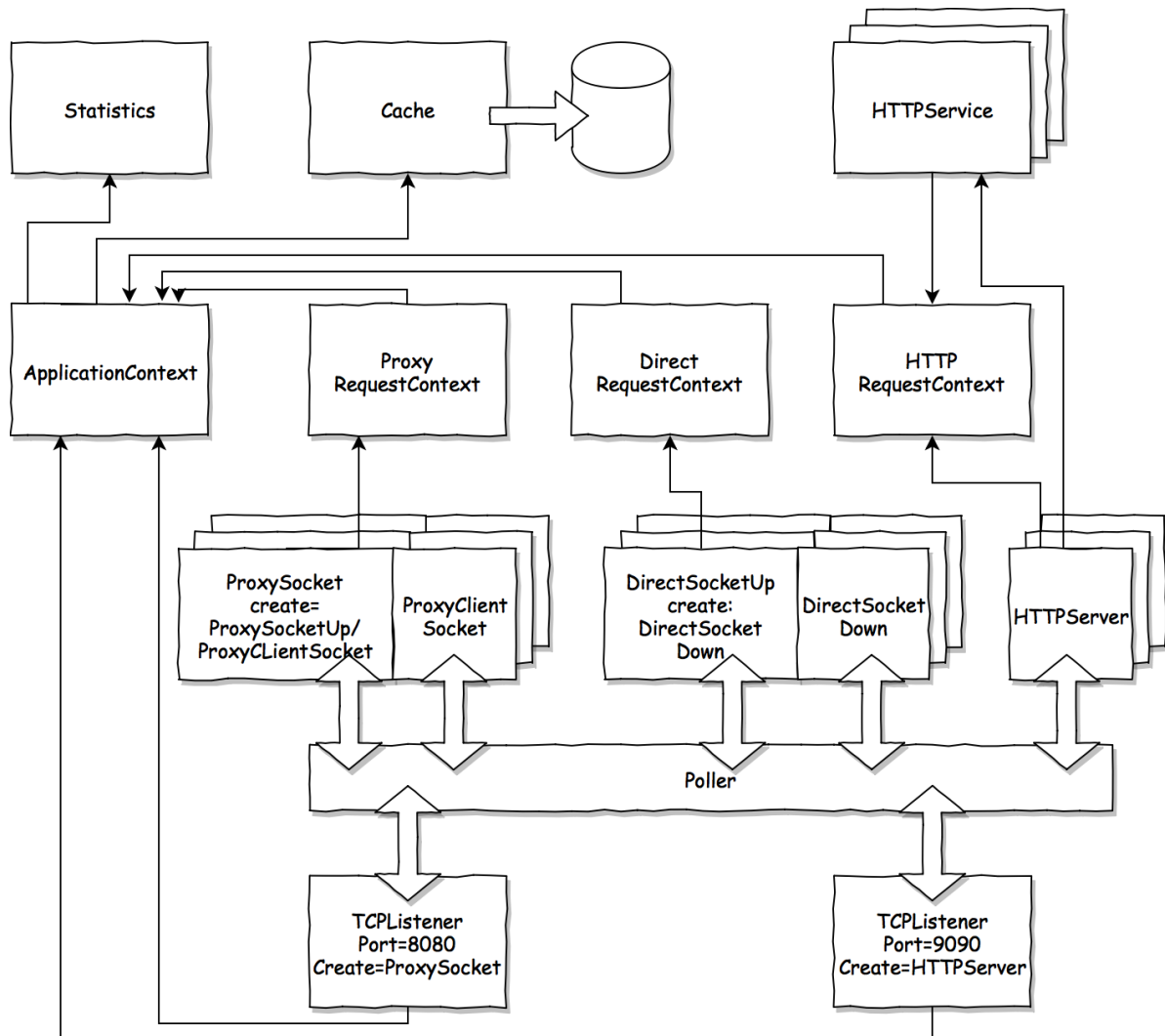
[Sequence diagrams](#) - **תרשים רצף** - תרשים רצף הנועד לתאר תהליך ולהסביר כיצד המערכת ורכיביה מבצעים תהליך זה.

[log](#) - **קובץ log** - קובץ שמטרתו לתעד את האירועים ואת השגיאות שמתרחשים במהלך ריצה של תכנית. פרויקט זה תומך בהגדרת מסמך log, אליו נכתבים פרטים של חיבורי רשת שנפתחים ונסגרים במהלך הריצה, כמו גם תיעוד של שגיאות אפשריות.

[CSS](#) - שפת תכנות לעיצוב דפי אינטרנט. הגיליונות קובעים את עיצובם של תגים ב-HTML, XHTML וכל שפה דומה ל-XML לבניית אתרי אינטרנט.

מימוש

דיאגרמת בלוקים



בדיאגרמה זו ניתן לראות את היחסים בין הישויות השונות בקוד, ואת האופן בו המערכת בנויה. בבסיס המערכת נמצא אובייקט הpoller, המאפשר את הפעולה האסינכרונית.

Poller

רוב מרכיבי המערכת מתבססים על תקשורת אסינכרונית, המאפשרת לשלוח מסרים בתוך ערוץ תקשורת, מבלי לחכות שהצד השני יהיה מוכן, ובכך להמנע מתקיעות ולאפשר תמיכה במספר מרובה של משתמשים.

מימוש שיטה זו מתבצע באמצעות אובייקט `polling`. בשיטה זו "רושמים" סוקטים שונים אל ה`poll`, וכאשר אחד מהם פנוי לקריאה או כתיבה מוחזר אירוע מתאים.

- `POLLIN` - אירוע קריאה.

- `POLLOUT` - אירוע כתיבה

- `POLLERR` - שגיאה

כל `socket` אשר נרשם אל הפול עטוף במחלקה מסוג `Pollable`, אשר בה ישנן פונקציות מתאימות הנקראות בכל אירוע: `on read`, `on write`, `on error`.

עם הרצת התכנית, נוצר אובייקט `poller`, והוא מלווה את התכנית עד לסגירתה.

בנוסף, עם הרצת התוכנית נוצרים שני אובייקטים, מהסוגים `ProxyListen` ו-`ServerListen`, האחראיים לקבלת חיבורים נוספים ובקשות מהשרת. שני אובייקטים אלה "נרשמים" אל אובייקט ה-`poll`.

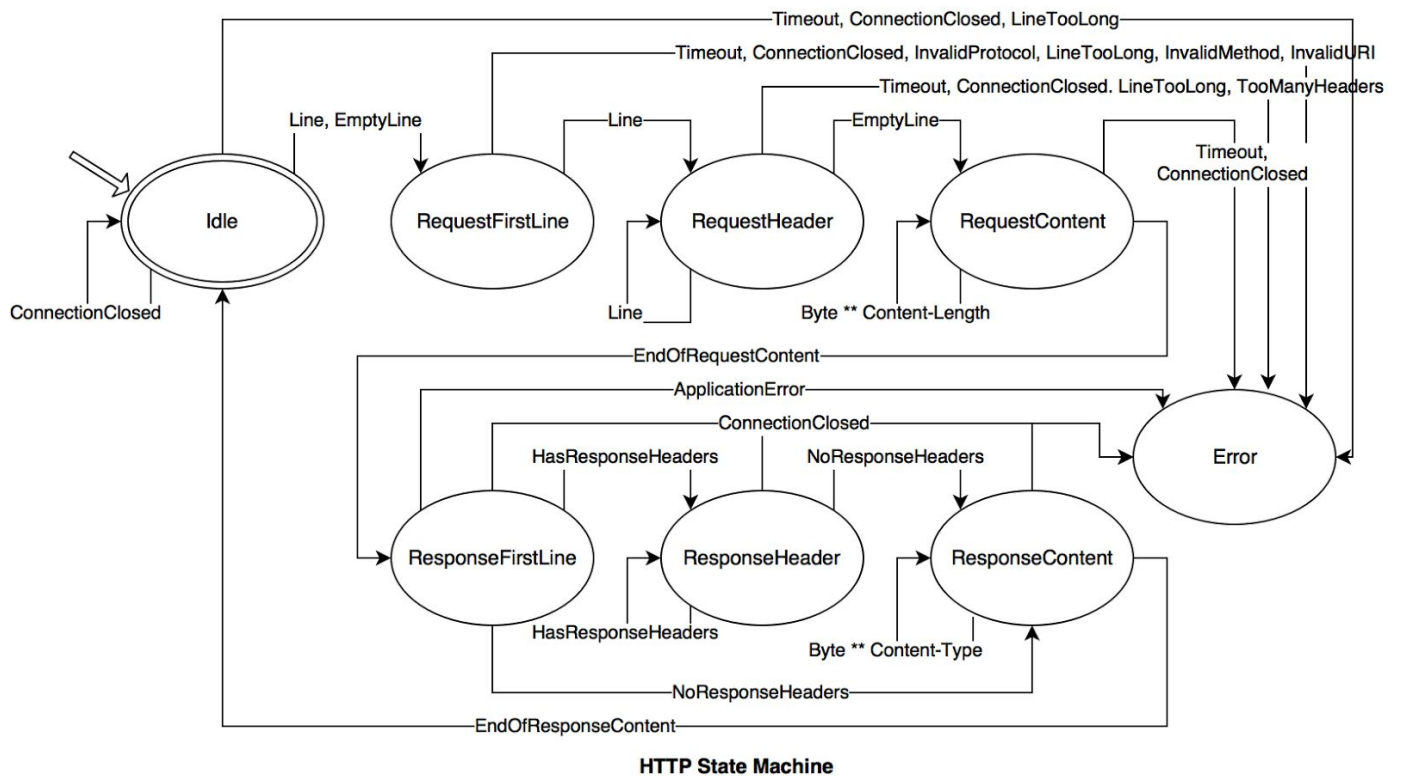
- `ProxyListen` - מאזין לפורט 8080 (כאשר לא צוין פורט אחר), ומוכן לקבלת חיבורים מה-`client`. בקשות המגיעות בפורט זה ינותחו ובקשתן תועבר הלאה לבדיקת זמינות ב-`cache` ולבקשה משרת היעד. האובייקט הנוצר עם קריאה לפונקציית `on read` של מחלקה זו, הוא `ProxySocket`.

- `ServerListen` - מאזין לפורט 9090 (כאשר לא צוין פורט אחר), ומוכן לקבלת חיבורים מה-`client`. בקשות המגיעות בפורט זה ינותחו על ידי סרבר שיווצר, והתשובה ללקוח תהיה קובץ המאוחסן מקומית או ממשק הניהול. האובייקט הוצא עם קריאה לפונקציית `on read` של מחלקה זו, הוא `HttpServer`.

HttpServer

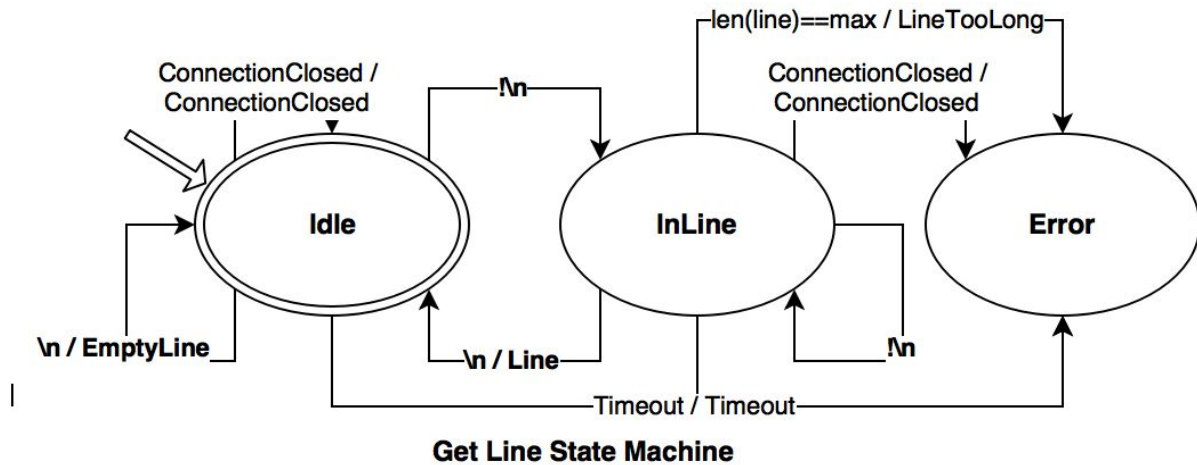
מחלקה זו היא למעשה שרת המאזין לפורט 9090.

כל בקשת HTTP המתקבלת במחלקה זו מנותחת ומקבלת תשובה על פי מכונת המצבים הבאה:



| מצב | הסבר |
|---------------------|---|
| Idle | שלב המתנה לבקשה. |
| Request First Line | שלב קבלת שורת הסטטוס. השרת מקבל את שם השירות המבוקש. |
| Request Header | שלב קריאת ה-Headers. מכילים מידע אודות סוג ואורך ההודעה המתקבלת ועליהם מבוסס הפרוטוקול. |
| Request Content | שלב קריאת תוכן ההודעה. |
| Response First Line | שלב שליחת שורת הסטטוס של התגובה. השרת מודיע ללקוח על כישלון או הצלחה בביצוע השירות. |
| Response Headers | שלב שליחת ה-Headers. השרת שולח ללקוח מידע אודות התגובה. |
| Response Content | שלב שליחת תוכן התגובה (עבור ממשק המשתמש, תוכן התגובה יהיה דף HTML המכיל את הדף הרצוי). |
| Error | במידה באחד מהשלבים שתוארו התקבלה שגיאה בשרת, השרת מנתק את |

כל שורה בפרוטוקול HTTP נגמרת בשורה חדשה ('r/n'). לכן קבלת התוכן בפרוטוקול התבצע על ידי קריאת שורות מלאות, על מנת לטפל בהן לאחר מכן. להלן מכונת מצבים אשר מתארת את תהליך קריאת השורה:



Input events

| Input event | Description |
|------------------|--------------------------------------|
| \n | New line |
| !\n | Any byte other than new line |
| Byte | Any byte |
| Timeout | A timeout while waiting for an input |
| ConnectionClosed | Connection closed by peer |

Output events

| Output event | Description |
|------------------|--------------------------------------|
| Line | A non empty line |
| EmptyLine | An empty line |
| Timeout | A timeout while waiting for an input |
| ConnectionClosed | Connection closed by peer |

States

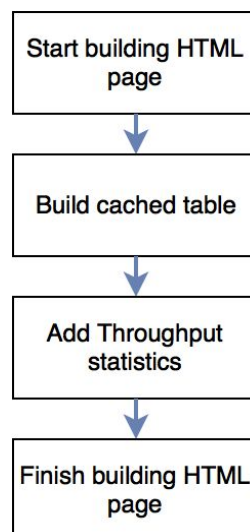
| Output event | Description | Transition in init | Per invocation |
|--------------|-----------------|--------------------|----------------|
| Idle | Between lines | | |
| InLine | Inside line | Set line = " | line += byte |
| Error | Error condition | | |

בשלב קבלת שורת הסטטוס, השרת בודק האם בוקש שירות מיוחד, ובמידה וכן יוצר אובייקט מתאים. במידה ולא התבקש שירות מיוחד קיים, השרת יחפש קובץ בשם שהתקבל, בתיקיית הבסיס של הפרויקט, וישלח אותו חזרה ללקוח. לדוגמא, באופן זה פועל שליחת קובץ CSS, המקנה לדף הניהול את העיצוב שלו.

כרגע בפרויקט ישנו שירות אחד פעיל, והוא שירות הניהול. שירות הניהול, `ManageInterface`, נקרא כאשר הוזן נראה כך (בהתאם לכתובת ולפורט שהוגדר):

`localhost:9090/manage`

דף הניהול נבנה בשפת HTML על פי המבנה הבא:



ProxySocket

מחלקה זו היא המחלקה האחראית על התקשורת בין הלקוח לבין שרת הפרוקסי. כאשר מתקבלת בקשה מצד הלקוח, היא מנותחת במחלקה זו, והיא זו שמחליטה איזו מחלקה לפתוח על מנת להמשיך לטפל בבקשה.

1. ראשית, מיד לאחר פענוח שורת הבקשה הראשונה, מתבצעת פעולת בדיקת סוג הבקשה. במידה והתקבלה בקשת CONNECT, מחלקה מסוג DirectSocketUp נוצרת ונרשמת ל poll, והמחלקה הנוכחית נסגרת ומוציאה עצמה מה poll.

2. במידה והתקבלה בקשת GET, מתבצעת בדיקה האם התשובה ל url שבוקש שמורה ב-cache, על ידי קריאה לפעולה מתוך המחלקה Cache. במידה והתשובה שמורה ובתוקף, התשובה תשלח על ידי המחלקה ישירות מתוך הזיכרון, ולאחר מכן ה socket יסגר ויצא מה-poll.

3. במידה והתקבלה בקשת GET, אשר אינה שמורה בזיכרון ה-cache, תיווצר המחלקה ProxyClientSocket. מחלקה זו מקושרת למחלקה ProxySocket והן עובדות בצמוד זו לזו, כאשר ProxyClientSocket שולחת את הבקשות שנותרו על ידי ה-ProxySocket לשרת היעד, מקבלת ממנו את התשובה ומנתחת אותה, ומחזירה את התשובה ל ProxySocket, משם היא נשלחת חזרה ללקוח. המחלקה משתמשת בפונקציה מהמחלקה Cache (תתואר בהמשך), על מנת לבדוק האם ניתן לאחסן את התשובה. במידה וניתן, התשובה תשמר ב-cache. לאחר סיום שליחת התשובה שני ה-sockets יסגרו ויצאו מאובייקט ה-poll.

DirectSocketUp

מחלקה זו נוצרת על ידי ה-ProxySocket כאשר מתקבלות בקשות מסוג CONNECT. ה-socket הפעיל במחלקה זו מועבר מהמחלקה ProxySocket, כך שעדיין קיים חיבור בין מחלקה זו לבין הלקוח השולח את הבקשה.

בתחילת ריצתה, המחלקה יוצרת את המחלקה DirectSocketDown, אשר מתחברת אל שרת היעד, בהתאם לכתובת שצוינה בבקשה שהתקבלה. שתי מחלקות אלה עובדות בצמוד זו לזו,

ויוצרות מעין tunnel. כל הבקשות/תשובות המתקבלות מאחד הצדדים, נשלחות לצד השני, ללא ניתוח ההודעות או שמירתן. זהו חיבור מאובטח המאפשר בקשות בפרוטוקול HTTPS.

Cache

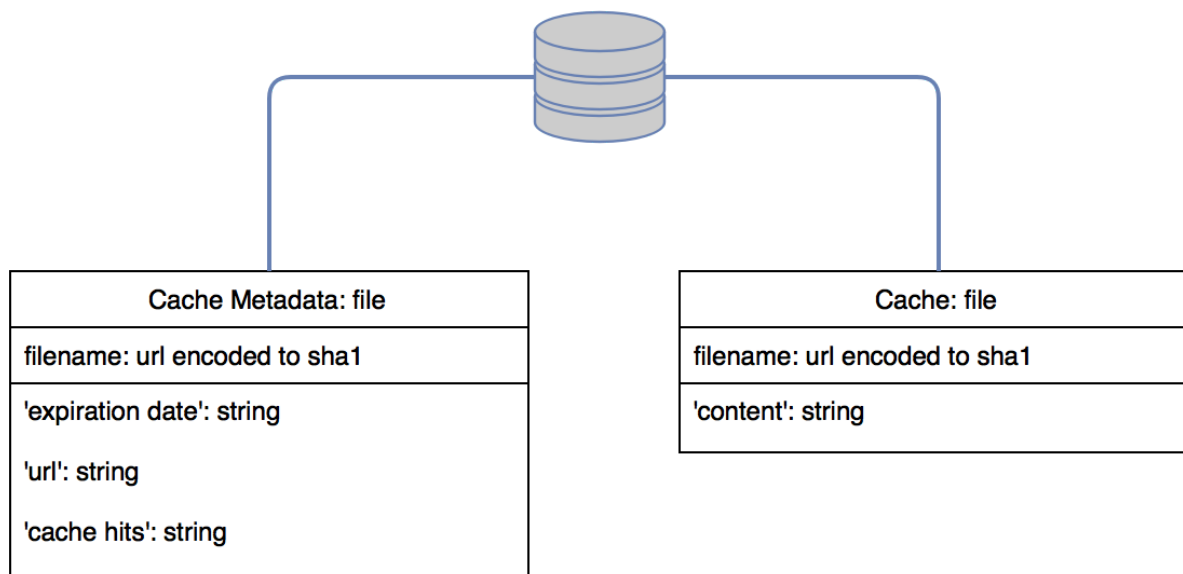
מחלקה זו היא המחלקה האחראית לטיפול בכל הפעולות הקשורות ב-cache.

המחלקה נוצרת עם הרצת התוכנית, ומועברת ליתר המחלקות על מנת שיכולו לבצע שימוש בפעולותיה.

למחלקה מספר שימושים עיקריים:

- שמירת קבצי cache חדשים - כאשר מתקבלות תשובות שרת היעד במחלקה ProxyClientSocket, מתבצעת קריאה לפונקציה הבודקת האם ניתן לשמור את התשובה. במידה וכן, התשובה המלאה תשמר.
- הבדיקה נעשית על פי ה-headers שמתקבלים בתשובה. Cache-Control הוא ה-header הרלוונטי לטיפול ב-cache. במידה ונרשם בשורה זו- max-age=0, no-cache או שהשורה אינה קיימת, התשובה לא תישמר. אחרת, התשובה תישמר למשך הזמן שניתן ב-max-age, בשניות.
- קבצי ה-cache ישמרו בתיקייה ייעודית בתוך תיקיית הפרויקט. לכל תשובה ישמרו שני מסמכים - אחד, המכיל את תוכן התשובה עצמה, והשני, קובץ metadata, המכיל נתונים הקשורים ב-cache ספציפי זה. שמות הקבצים יהיו זהים, אך הם ימצאו בתיקיות שונות. שמות הקבצים הם למעשה התוצר של פונקציית גיבוב, sha1, על כתובת ה-url המבוקשת. ישנו צורך בשימוש בפונקציית הגיבוב, מכיון שלעיתים כתובות ה-url ארוכות מדי מכדי לשמש כשם לקובץ.

ניתן לראות את אופן שמירת הקבצים בתרשים הבא:



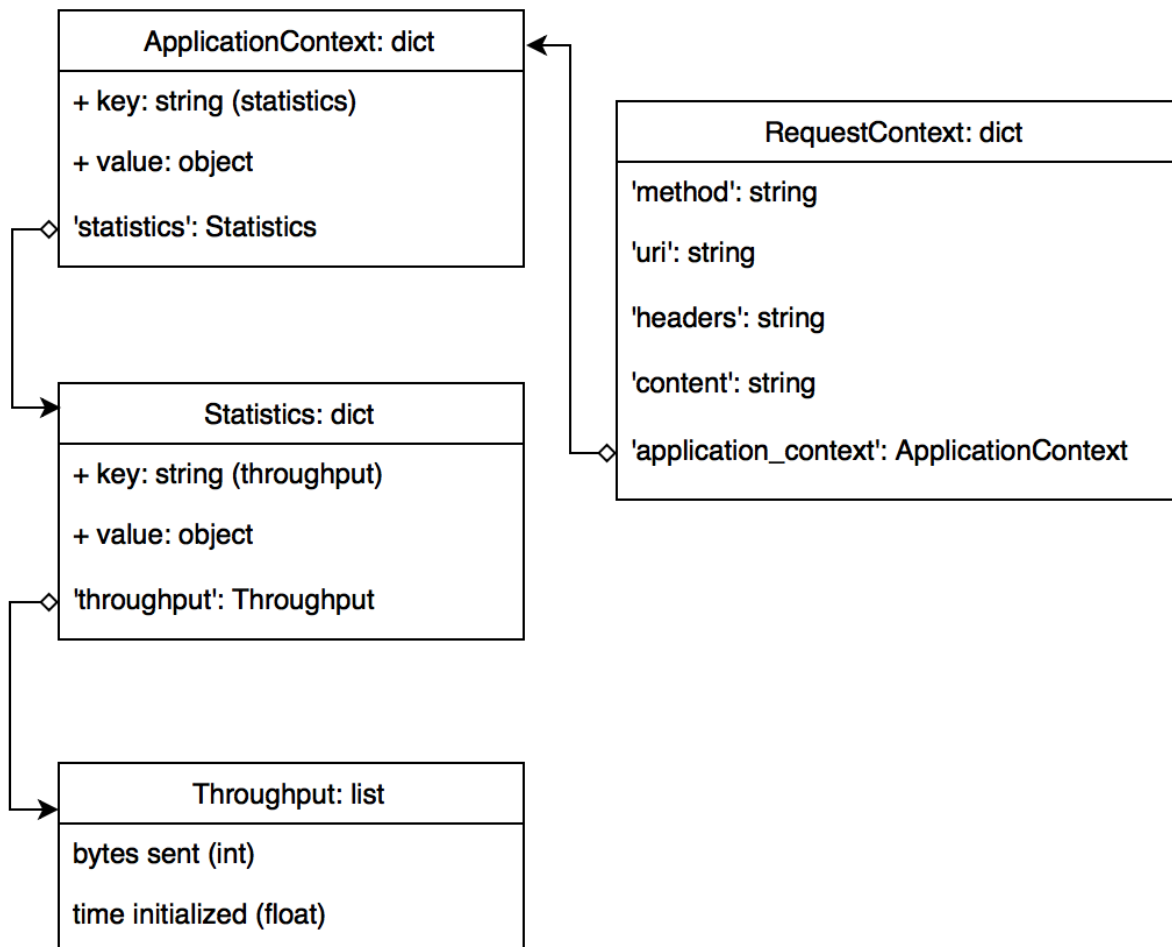
- שימוש בקבצי cache - כאשר מתקבלת בקשה מהלקוח, היא מנותחת במחלקה ProxySocket, ומתבצעת בדיקה האם התשובה לבקשה מאוחסנת ב-cache. לצורך הבדיקה, נקראת הפונקציה check_if_cache. ראשית, ישנה בדיקה האם קיים קובץ המאחסן את התשובה. במידה וקיים קובץ כזה, נבדק קובץ ה-metadata המתאים, על מנת לבדוק האם התגובה בתוקף. אם התגובה אינה בתוקף, הקובץ נמחק. במידה וה-cache מתאים לשימוש, הקובץ נפתח והתשובה נקראת ממנו בחלקים, בהתאם לקצב השליחה ולגודל buffer השליחה המקסימלי.
- הצגת הנתונים הקשורים ב-cache - כאשר מתקבלת בקשה להצגת ממשק המשתמש, נוצר הדף, המכיל בעיקר נתונים על השימוש בפרוקסי. כאשר דף זה נבנה, המחלקה הבונה אותו, Managelinterface, קוראת לפונקציה get_cached_data מתוך המחלקה. הפונקציה עוברת על הקבצים השמורים, וקוראת את קבצי ה-metadata שלהם. מוחזר dict המכיל את כתובת ה-url, תאריך התפוגה של ה-cache, ומספר ה-cache hits, כלומר כמה פעמים נעשה שימוש בקובץ.
- מחיקת קבצי cache - קבצי cache אשר בוקשו אך פג תוקפם ימחקו אוטומטית. בנוסף לכך, יש אפשרות למחוק קבצי cache דרך ממשק הניהול, קבצים בודדים או את כולם. כאשר מתקבלת בקשה כזו בשרת, הוא קורא לפונקציה delete_cache/delete_all_cache שבמחלקה Cache. הקובץ המאחסן את התשובה וקובץ ה-metadata המתאים לו ימחקו.

מבני נתונים נוספים

מבני נתונים נוספים אשר נעשה בהם שימוש בפרויקט הם הם המילונים, dictionaries.

- RequestContext - מילון זה נוצר במחלקות HttpProxy, HttpClientProxy ו-HttpServer. מילונים אלו נועדו לאחסון נתונים המתקבלים מבקשות / תשובות HTTP, על מנת לאפשר העברה נוחה של הנתונים ואחסונם במקום אחיד אחד. אחד הדברים המאוחסנים במילונים אלה, הוא המילון ApplicationContext.
- ApplicationContext - מילון זה נוצר בתחילת הרצת התוכנית, והוא מלווה אותה לכל אורכה, כאשר הוא מועבר למחלקות השונות. במילון זה מאוחסנים נתונים הקשורים לאפליקציות ולשירותים השונים בתכנית. כרגע, האיבר היחיד הנמצא בו הוא המילון statistics, אך התשתית להוספת אלמנטים ושירותים נוספים קיימת.
- Statistics - מילון זה נועד לשמירת נתונים שונים וסטטיסטיקות בנוגע לשימוש בתכנית. מטרת מילון זה ושמירת הסטטיסטיקות היא הצגתם בממשק הניהול. במילון זה קיים המפתח Throughput, אשר מחזיק בו רשימה המכילה נתונים על תעבורת האינטרנט העוברת דרך הפרוקסי.
- Throughput - רשימה (list) המכיל נתונים בנוגע ל-Throughput. באיבר הראשון מאוחסן מספר הבייטים שעברו דרך הפרוקסי, ובאיבר השני מאוחסן הזמן בו התחילה הספירה. מבנה נתונים זה מאפשר את הצגת קצב שליחת הנתונים בממשק המשתמש.

ניתן לראות ייצוג למבני הנתונים הנ"ל בתרשים הבא:



בעיות ידועות

המגבלה העיקרית של התוכנית היא אי התאמתה למערכת ההפעלה Windows, אלא רק למערכות Unix.

בעיה נוספת היא מגבלתו של האובייקט select לעבודה עם עד 256 sockets. בשיטה עליה מבוסס פרויקט זה, שימוש בספריית select של פייטון, אין פתרון לבעיה כרגע. פתרון אפשרי הוא הרצת התכנית עם select. עם מעבר הכמות המקסימלית של החיבורים יצירת תהליך אשר יוצר אובייקט select נוסף. כך ניתן יהיה לטפל בחיבורים נוספים האובייקט הנוסף מבלי לעלות על רף החיבורים באף אחד מהם. עם זאת, לא נעשה מחקר וניסיון לתקן את הבעיה הזו. (הבעיה הזו משפיעה לרוב במקרים בהם מנסים להתחבר מהדפדפן למספר אתרים כבדים בו זמנית, לדוגמה CNN ו-Ynet).

התקנה ותפעול

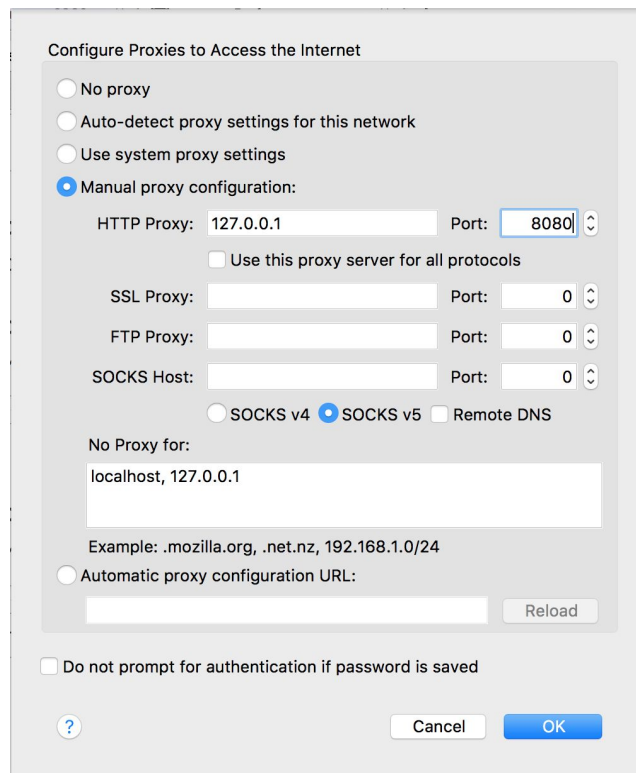
ההוראות הבאות יאפשרו לכם להריץ את הפרויקט על מחשבכם, לשימוש ופיתוח.

הכנות מקדימות

יש לבצע את השלבים הבאים טרם הפעלת התכנית:

1. הורידו והתקינו python 2.7.
[/https://www.python.org/download/releases/2.7](https://www.python.org/download/releases/2.7)
2. הורידו את הפצת התוכנית מתוך github. (לינק בהמשך המסמך)
3. במידה ואתם משתמשים במערכת ההפעלה ווינדוס, הורידו והתקינו Cygwin. (אם אתם משתמשים במערכת הפעלה תומכת POSIX, אין צורך בשלב זה)
4. הורידו והתקינו כל דפדפן אינטרנט מודרני. (Firefox מומלץ, מכיון שניתן לקבוע בו את הגדרות הפרוקסי מבלי לשנות את הגדרות הפרוקסי של המחשב)
5. הגדירו את הדפדפן כך שישתמש בפרוקסי, בהתאם לכתובת הIP של המחשב עליו תרוץ התכנית ובהתאם לפורט אותו תבחרו (ברירת מחדל: 8080).

On Firefox: Open Preferences → Advanced → Network → Connection



settings

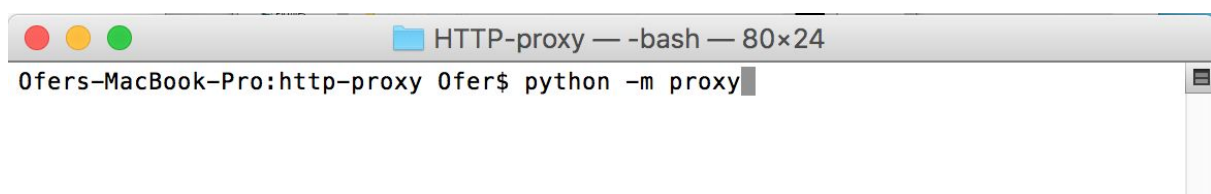
הרצת התכנית

הגיעו אל תיקיית האב: (HTTP-proxy)

`cd [location of HTTP-proxy]`

הריצו את התכנית:

`python -m proxy [args]`



ארגומנטים

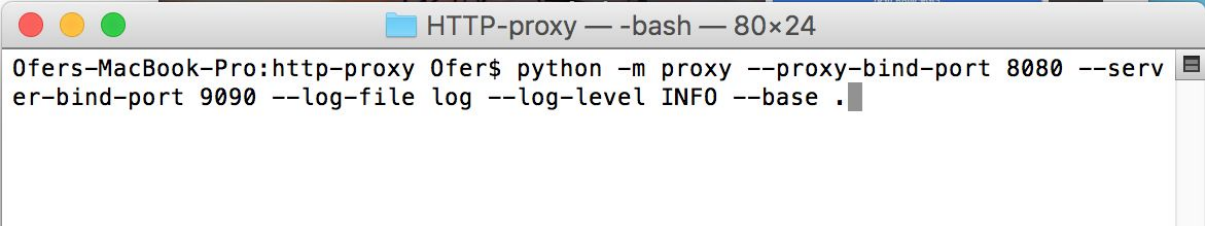
כל הארגומנטים בתכנית הם אופציונאליים. על מנת לקבל עזרה ולראות את ברירות המחדל, יש להכנס ל- --help

--bind-address - Bind address, default: 0.0.0.0
--proxy-bind-port - Proxy bind port, default: 8080
--server-bind-port - Server bind port, default: 9090
--base - Base directory to search files in, default: .
--log-level - Log details level, default: INFO
--log-file - Logfile to write to, default: os.devnull

על מנת ליצור קובץ לוג, יש לתת כארגומנט את שם קובץ הלוג. במידה והשם לא ינתן, לא ישמר קובץ לוג.

הרצת התכנית עם ארגומנטים תראה כך:

```
python -m proxy --proxy-bind-port 8080 --server-bind-port 9090 --log-file log  
--log-level INFO
```



```
HTTP-proxy — -bash — 80x24  
Ofers-MacBook-Pro:http-proxy Ofer$ python -m proxy --proxy-bind-port 8080 --server-bind-port 9090 --log-file log --log-level INFO --base .
```

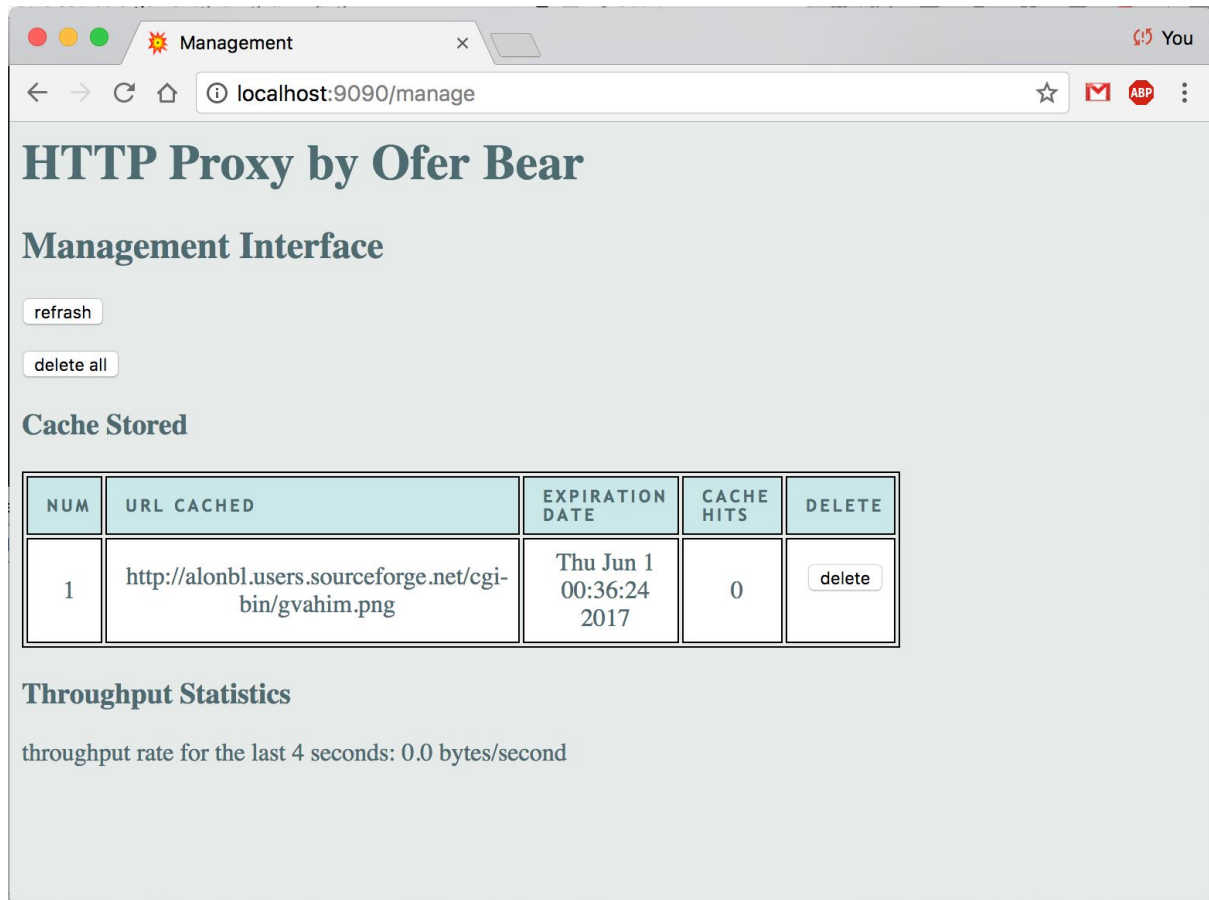
ממשק גרפי

אין ממשק גרפי כחלק מהתכנית הראשית. על מנת לגשת לממשק המשתמש בדפדפן יש לכתוב ככתובת:

(ip):(server_port)/manage

כאשר IP הוא כתובת הIP של המחשב עליו רצה התכנית, וserver_port הוא הפורט שנקבע כארגומנט עם הרצת התכנית, כאשר ברירת המחדל הוא 9090.

יפתח דף הניהול המרכזי, המכיל את טבלת cache ונתונים אחרים.



Management Interface

refresh

delete all

Cache Stored

| NUM | URL CACHED | EXPIRATION DATE | CACHE HITS | DELETE |
|-----|---|-------------------------|------------|-------------------------|
| 1 | http://alonbl.users.sourceforge.net/cgi-bin/gvahim.png | Thu Jun 1 00:36:24 2017 | 0 | <button>delete</button> |

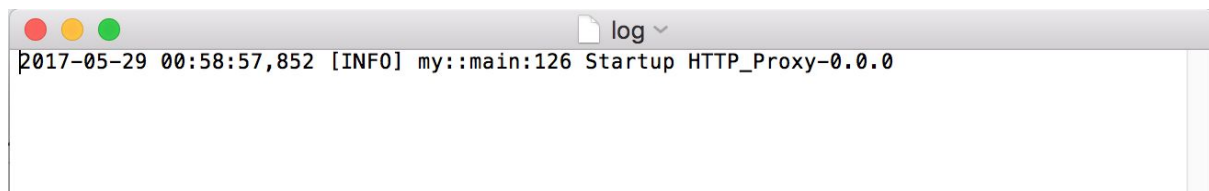
Throughput Statistics

throughput rate for the last 4 seconds: 0.0 bytes/second

מסמך Log

את מסמך הלוג ניתן למצוא בתיקיה הראשית של הפרויקט, HTTP-proxy, תחת השם אותו העברנו כארגומנט, או "log" כברירת מחדל.

כאמור, ניתן לבחור את סוג ההערות שירשמו בלוג ואת מידת הפירוט, על ידי העברת הארגומנט "--log-level". האופציות הן: DEBUG, INFO, WARNING, ERROR, CRITICAL. בהרצת התוכנית עם רמת INFO, קובץ הלוג יראה כך:



```
log
2017-05-29 00:58:57,852 [INFO] my::main:126 Startup HTTP_Proxy-0.0.0
```

לאחר בקשה (לכתובת <http://alonbl.users.sourceforge.net/cgi-bin/gvahim.png>), ולאחר יציאה מהתכנית, קובץ הלוג יראה כך:



```
2017-05-29 01:05:24,333 [INFO] my::main:126 Startup HTTP_Proxy-0.0.0
2017-05-29 01:05:52,336 [INFO] my::on_read:528 proxy request (HttpSocket object created - 6)
2017-05-29 01:05:52,336 [INFO] my::request_state:136 HttpSocket 6: GET request, http://
alonbl.users.sourceforge.net/cgi-bin/gvahim.png, HTTP/1.1
2017-05-29 01:06:15,539 [INFO] my::main:160 All sockets closed, shutting down log
```

קובץ README.md

HTTP Proxy

HTTP proxy with management capability

A final project for the Gvahim program. This project provides proxy server, with management capabilities for HTTP requests. When the server is running, all network between the browser and the internet goes through the proxy, allowing it to store and use cache, and display statistics.

Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

Prerequisites

Here are the things you will need to download in order to get this system up and running:

- 1) Download and install Python2.7
(<https://www.python.org/download/releases/2.7/>)
- 2) Download this repository on your machine (Posix machine only)
- 3) On Windows, download and install Cygwin
- 4) Download and install any modern browser
- 5) Set up your browser to use proxy, according to the ip where the program will run and the port you'll choose (on Firefox, use "Foxyproxy")

Execution

Reach parent directory (HTTP-proxy)

```
cd [location of HTTP-proxy]
```

Running the Proxy:

```
python -m proxy [args]
```

Arguments

All arguments are optional. To view defaults, see `--help`. Execution with arguments would look like:

```
python -m proxy --proxy-bind-port 8080 --server-bind-port 9090 --log-file log  
--log-level DEBUG
```

Graphical Interface

There is no graphical interface as part of the main program. In order to enter the GUI in your browser type:

```
(your_ip):9090/manage
```

This will open the main page where statistics about the program and cache details may be seen.

Authors

- Ofer Bear - *Initial work* - [My Profile](#)

Acknowledgments

- Thanks to Alon Bar-Lev and Sarit Lulav for teaching, helping and commenting.

תוכניות לעתיד

כיום, ניתן למצוא עשרות שרתי פרוקסי המופצים וניתנים לשימוש בחינם. לשרתי הפרוקסי ישנם שימושים רבים: הסוואת כתובת הIP, ניהול רשתות מחשבים, אחסון קבצי cache לרשת מהירה יותר, חסימת אתרים, עקיפת חסימות אינטרנט ועוד.

שרת הפרוקסי שממשיך אינו משמש לכל מטרות אלה, וניתן היה להמשיך ולפתח אותו. מפאת קוצר הזמן, לא הוכנסו אליו פיצ'רים נוספים.

לדוגמא, ניתן היה להוסיף:

- הוספת תצורה וממשק ניהול לסינון כתובות URL מסוימות. אם משתמש יבקש כתובת URL מהרשימה הוא יחסם על ידי הפרוקסי.
- הוספת אלגוריתם סינון המבוסס על חתימות התוכן. כל דף מאושר יקבל header עם חתימה מוצפנת. מפתחות ההצפנה יתווספו אל הפרוקסי, והפרוקסי יאפשר גישה רק לדפים מאושרים.
- הוספת נתונים וסטטיסטיקות נוספות לממשק הניהול.
- התאמת התכנית לעבודה במערכת ההפעלה ווינדוס.
- שיפור ועיצוב ממשק המשתמש. כעת, ממשק המשתמש מאוד בסיסי, וניתן לשפר ולעצב אותו בעזרת קובץ CSS.

פרק אישי

פרויקט זה היה פרויקט התוכנה הגדול הראשון שיצא לי לבצע, ולמדתי ממנו הרבה על אופן העבודה ועל איך לבצע פרויקטים גדולים בהנדסת תוכנה.

עם תחילת העבודה על הפרויקטים, כאשר ניתנה לנו האופציה לבחור פרויקט לפיתוח, בחרתי בפרויקט זה. בחרתי נושא זה משום שהוא נשמע לי מעניין ורלוונטי גם לחיי היום יום - יצא לי להשתמש לא מעט פעמים בשרתי פרוקסי, והמחשבה על פיתוח אחד משלי נשמעה לי מעניינת ומיוחדת.

טרם תחילת הפיתוח ובמהלכו, למדתי הרבה חומר תיאורטי הקשור לנושא, אשר היה נחוץ על מנת להבין כיצד לממש את המערכת. בנוסף, למדתי להשתמש בכלים חשובים רבים וחשובים, אשר לא הכרתי טרם למימוש פרויקט זה. ידע זה שרכשתי בזכות הפרויקט, ימשיך כנראה ללוות אותי בהמשך דרכי.

במהלך הפרויקט נתקלתי באתגרים רבים אשר הקשו עלי, החל מקושי בחשיבה על דרך פתרון לבעיה מסוימת, דרך באג קטן שהופיע והיה קשה לדבאג, ועד כתיבת תיק פרויקט זה. הצלחתי להתמודד בזכות חברים טובים, מנחים טובים, חיפושים וחקירה בגוגל והשקעה רבה. אני יודע כי למדתי מקשיים אלה הרבה, ורכשתי נסיון רב בהתמודדות איתם.

אחד הדברים החשובים שהבנתי בנוגע לעבודה על פרויקט בסדר גדול כזה, הוא חשיבות התכנון מראש של המערכת, ובניית לוחות זמנים ועמידה מהם. במידה והייתי מיישם תובנות אלו בפרויקט, הייתי נמנע מלחץ ברגע האחרון, ויתכן כי מבנה קוד הפרויקט היה מסודר יותר.

הפרויקט בעיני הוא רק ההתחלה. בעתיד, כנראה שארצה לעסוק בתחום הנדסת התוכנה והמחשבים, וכנראה שאבצע פרויקטים נוספים בחיי. פרויקט זה נתן לי מוטיבציה להמשיך ולפתח, והעמיק את סקרנותי לעולם הסייבר.

לסיכום, אני רוצה להודות לשרית לולב ולאלון בר לב, על ההנחיה ועל התמיכה במהלך הפרויקט, ועל ההוראה והלימוד במשך שלושת שנות הלימוד במגמה.

קוד הפרויקט ותיעודו

ניתן למצוא את כל הקוד בgithub, בלינק הבא:

<https://github.com/oferbear/HTTP-proxy/releases>

יש להוריד את ההפצה האחרונה של הפרויקט, ולחלץ אותה מקובץ ה-zip.

התיעוד מצורף, ונמצא בתיקייה documentation.zip. על מנת לצפות בתיעוד, יש לפתוח את

התיקייה, לחלץ ממנה את הקבצים, ולפתוח את הקובץ index.html.

נספחים

HTTP Proxy Sequence Diagram:

(<http://sequencediagram.org>)

title HTTP Proxy: Sequences

actor Browser

participant Proxy server

database Cache

participant dest-server

==HTTP Request==

autonumber 1

par Port 8080

Browser->Proxy server: HTTP request

Proxy server->Proxy server: Parse request

Proxy server->Cache: Retrieve url

Cache->Cache: Check if url exist and \nparse his metadata file

alt If response is stored and not expired

Cache->Proxy server: Content

end

alt If cache miss

Proxy server->dest-server: HTTP Request

dest-server->Proxy server: HTTP Response

Proxy server->Proxy server: Parse response

Proxy server->Proxy server: Check Cache-Control header

alt If max-age != 0

Proxy server->Cache: Store metadata and content

end

end

end

==Management Interface==

autonumber 1

par Port 9090

Browser->Proxy server: HTTP Request


```
Proxy server->Proxy server: Parse request
alt /manage
    Cache->Proxy server: Cache data
end
Proxy server->Browser: HTTP Response
end
```

==CONNECT Request==

```
autonumber 1
parallel on
Browser->Proxy server: HTTPS request
Proxy server->dest-server: HTTPS request
par
dest-server->Proxy server: HTTPS response
Proxy server->Browser: HTTPS response
parallel off
```