שלום לכולם.  
  
הקדמה קצרה.

למי שלא מכיר אותי שמי ניר וכיום אני עובד בקונדואיט על פיתוח מוצר חדש שאמור לצאת לשוק בחודשים הקרובים.

כשדיברנו על ההרצאה אני ורוני (שירצה אחרי הפסקת הצהרים), רצינו לבנות אפליקציה לדוגמא שתכיל את הנושאים שתכנו לדבר עליהם ולהציג את האפליקציה. הרעיון היה לא להציג מצגת של רשימת מכולת אלא דוגמאות עובדות של קוד.

בגלל מגבלות בעיקר של חוסר זמן (כי עדיין המחויבות הראשית היא למקום עבודה) ומגבלות טכניות שאת חלקם אני אעביר בהרצאה לא הספקנו להרים אפליקציה אבל מרות זאת ההרצאה שלי מלווה בלא מעט קוד והמטרה שלי היא שמי שירצה יקח את השקפים לאחר ההרצאה ויוכל ישר להתחיל לשחק עם הקוד ולא רק להיות עם רקע תאורטי.  
  
למי שיש שאלות אני מעדיף שתרשמו אותם ואני אנסה לענות בסוף ההרצאה או בהפסקה. תרגישו חופשי לגשת לשאול ואם לא הספקתם קחו כרטיס ביקור ותשלחו לי שאלות במייל.

התכנים שאני עומד לדבר עליהם היום:

Web workers  
כאשר בWEBWORKERS נכסה את הסוגים הבאים

1. **Dedicated Workers**
2. **Inline Workers**
3. **Shared Workers**

אני מניח שרובכם נתקל בWEB WORKERS אבל יחד עם זאת אני מניח שרובכם לא שמע על INLINE וSHARED WORKERS.

fileAPI  
הנושא הזה מכיל בתוכו 2 חלקים: עבודה מול קבצים מקומיים במחשב וAPI מקביל שמאפשר עבודה מלאה מול קבצים בSANDBOX של הדפדפן ומכיל בתוכו API שלם לעבודה עם FILES וDIRECTORIES.   
מקוצר זמן אני לא אספיק לעבור על הנושא האחרון של ההרצאה אבל גם עבורו הכנתי מצגת נפרדת עם דוגמאות של קוד.  
  
2 הנושאים הללו חולקים מספר דברים משותפים ובמהלך ההרצאה נראה שיש לא מעט קוד משותף ל2 השנואים הללו.

1. הרעיון המרכזי של WEBWORKER הוא לפתור את הבעיה שאנחנו המפתחים עומדים מולה היום והיא שדפדפנים לא תומכים בMULTI THREADS.  
   כיום על לדמותTHREADS היינו משתמשים באחת מהצורות הבאות על מנת להריץ קוד בצורה אסינכרונית.  
   AJAX  
   2XHR – רכיב נוסף ב 5HTML שלא יכוסה בהרצאה הזו.  
   EVENTS  
   SETINTERVAL שאני באופן אישי לא ממליץ להשתמש בו בכלל היות ואף פעם לא נדע בוודאות כמה זמן לקח לקוד לרוץ. לדוגמא אם אנחנו מעדכנים חלק בדף וטוענים תוכן ממקור חיצוני כל 0.5 שניות ובפועל כמה בקשות לא נסתיימו בזמן, התשובה טרם היגיע חזרה לדפדן אבל בגלל שהשתמשנו בSETINTERVAL ולא SETIMEOUT הפונקציה נקראת שוב ונשלחת עוד בקשת עדכון.  
   הבעיה היא: איזה מידע יוצג אחרון.  
   הפתרון – שימוש רק בSETTIMEOUT במקום כאשר את הSETTIMEOUT נפעיל בסוף הפונקציה כאשר היא סיימה את פעולתה. ככה נוכל להבטיח שלא יווצר מצב שהפונקציה תקרא שוב לפני שהיא נסתיימה.  
     
   שימוש בWEB WORKERS מייתר את הצורך למצוא פתרונות יצירתיים שכאלו.
2. WORKERS
   1. הרעיון המרכזי הוא שהקוד שWORKER מאפשרים שימוש בTHREADS להרצת קוד JS והם רצים במקביל לTHREAD הראשי שלרוב יטפל בכל נושאי ה UI בדף.
   2. הקוד של הWORKER נמצא בקובץ נפרד (חוץ ממקרה יוצא מהכלל שנראה מאוחר יותר)
   3. יצירה של WORKER מורידה את הקוד בצורה אסינכרונית ואם הקובץ לא נמצא נקבל שגיאה 404 והקוד מן הסתם לא ירוץ.
   4. WORKERS יכולים לתקשר עם העולם החיצון בעזרת POST MESSAGE ו ONMESSAGE.  
      שזו גם הצורה היחידה שבה ניתן לבצע DEBUG על WEBWORKERS.
3. לWORKERS יש גישה לאובייקטים הבאים:  
   IMPORT SCRIPT זוהי הפונקציה שמאפשרת טעינה של קבצים חיצוניים לWORKER
4. היות שWORKERS רצים בסביבת הרצה נפרדת משלהם אין להם גישה לאובייקטים הבאים:
   1. DOM – not a thread safe is there was an aces
   2. WINDOW - כנגזרת מזה אין גישה לפונקציות ולמשתנים בדף הראשי.
   3. Document - לא ניתן לשנות או להוסיף EVENTS
   4. ושוב העברת מידע בין הWORKER לחלון הראשי או לחלונות- טאבים אחרים מתבצעת באמצעות MESSAGES
5. WORKER CODE MAIN WINDOW
   1. זוהי הצורה הבסיסית ליצירת WORKER. יוצרים NEW WORKER וכפרמטר מעבירים את שם הקובץ שיש לטעון
   2. העברת מידע לWORKER יכולה להתבצע ב1 מ2 דרכים:
      1. דרך ONMESSAGE
      2. דרף LISTENER
      3. 2 הדרכים הללו זהות לחלוטין
   3. על מנת להתחיל את הWORKER יש לשלוח לו הודעה בעזרת POST MESSAGE רוב הדפדפנים תומכים גם בהעברת הודעות בפורמט JSON.  
      כרום מגרסה 14 תומך גם העברת מידע המכיל ARRAYBUFFER (נגיע לזה יותר מאוחר)
6. WORKER FILE
   1. בWORKER CONTEXT אנחנו מאזינים לMESSAGE כמו הדף הראשי.
   2. כמו כן SELF THIS או כלום כולם מתייחסים לWORKER CONTEXT
7. DEDICATED WORKER DEMO  
   ניתן לראות ששימוש בWEBWORKERS לא חוסם את הדפדפן מלבצע פעולות אחרות ובמקביל פעולות שצורכות זמן וכח חישוב מתבצעות ברקע.
8. INLINE WORKER  
   בניגוד לDEDICATED WORKER שבו כאמור הקוד נמתא בקובץ נפרדINLINE WORKER הוא WORKER שנוצא בצורה דינמית כאשר. זה מחזיר אותנו לשקף השני שבו ציינו שהקוד של הWORKER מוגדר בקובץ נפרד ונטען בצורה אסינכרונית. כאן כאמור הקוד נכתב ON THE FLY ויכול אפילו לשבת בדף הראשי או בכל מקום אחר.  
     
   INLINE WORKER כפי שתכף נראה בדוגמת קוד נוצר באמצעות BLBOBUILDER. אובייקט זה יחזור אלינו מאוחר יותר כשנעבור לFILEAPI.  
   השימוש בBLOBLBUILDER יותר לנו אובייקט מסוגBLOBURL שמאוחסן בזכרון של הדפדפן. משך החיים שלו הוא כל עוד הדף חי.  
   רישמה של כל הBLOBS ניתן לראות בקישור הבא.
9. דוגמת קוד לINLINE WORKER
   1. ה"טריק" שאנו מציגים בדוגמא הנל הוא לרשום את הקוד של הWORKER בתוף תגית סקריפט מסוג לא מוכר על מנת שהדפדפן "יתעלם" ולא יעבד את התוכן כסקריפט. זו טכניקה נפוצה כיום ומי שמכיר JQ TEMPLATE מתבסס על אותו הרעיון.  
      בתוך תגיד הסקריפט נכתוב את הקוד כאילו הוא כתוב בקובץ חיצוני.
   2. יצירת הWORKER מתבצעת על ידי טעינת הקוד של הWORKERלתוך BLOB BUILDER ולאחר מכן יצירת OBJECT URL שזה עוד משהו שנפגש איתו מאוחר יותר כשנגיע לFILE API  
      הקוד הנ"ל יוצר לנו WORKER שיושב בזכרון ושוב ניתן לראותו תחת הקישור לדפי הBLOB כפי שנראה בדמו בעוד 2 שקפים.
   3. שאר הקוד זה מה שאנחנו כבר מכירים.
   4. היות והקוד יושב בזכרון חשוב לשחרר את הזכרון במקרה ואין צורך יותר בWORKER שלנו.
10. עוד דוגמת קוד זהה לקודמת רק שהפעם במקום לקרוא את הקוד מתוך תגית סקריפט הקוד מוזרק בתור טקסט לBLOB BUILDER
11. דמו של INLINE WORKER  
      
    בדמו הזה הWORKER שלנו שולח כל שניה הודעה לדף הראשי.  
    ניתן לראות שהID של הWORKER זהה כל עוד לא פתחנו אחד חדש. ברגע שנפתח אחד חדש תתוסף לנו רשומה בדף הרלוונטי
12. SHARED WEB WORKER
    1. המשותף לשתי התצורות הקודמות של WORKERS שראינו קודם היא שכל אחד מהם חי בעולם סגור משל עצמו ויודע רק לתקשר עם הסביבה על ידי האזנה ושליחה של הודעות.  
         
       הסוג הזה של WORKERS חי בסביבה "חופשית" שמאפשרת גישה לקוד שלו כמעט לכל אחד. כל קוד שיושב תחת אותו דומיין יכול לגשת לWORKER , לקרוא ולעדכן בו כראות עיניו.
    2. היות וציינו כבר שהקוד של הWORKER הזה הוא נגיש לכולם נגזר מזה שהוא סוג של SINGLETONE . בשונה מהתצורות הקודמות שראינו הדיבור עם הסוג השה של הWORKER נעשה באמצעות PORT. כל מי שרוצה לתקשר עם הWORKER חייב דבר ראשון לקבל ערוץ תקשורת משלו ובמילים פשוטות לפתוח פורט ורק לאחר מכן יוכל לדבר עם הWORKER שלנו.
    3. נקודה חשובה:  
       בניגוד לסוגים הקודמים של WORKERS הסוג הזה חי עד אשר "נהרוג" אותו. לכן מאוד חשוב ברגע שסיימנו את העבודה עם הWORKER לסגור את הPORT שלנו איתו על מנת לשחרר אותו מהזכרון.
13. דוגמת קוד לSHARED WORKER
    1. יצירת הWORKER כמו במקרים הקודמים כששוב אני מזכיר שהשוני פה הוא שורות 10 ו13 כשהתקשורת עם הWORKER מתבצעת דרך PORT ולא ישירות דרף MESSAGES
14. SHARED WORKER הקובץ עצמו.
    1. הEVENT שאנחנו מאזינים לו במקרה הזה הוא CONNECT ולא MESSAGE
    2. בבקשת ההתחברות אל הWORKER נשלח פרמטר PORT שהוא תמיד יהיה באנדקס 0
    3. לאחר מכן הפורט מאזין להודעות שנשלחות אליו ושולח בחזרה MESSAGE כתשובה.
    4. לאחר שרמשנו את כל הEVENTS הרלוונטים נקרא לSTART ושוב הכל נעזה דרך פורט.
    5. SHARED WORKER יכולים לדבר עם כל "העולם" בעזרת HTML5 MESSAGE API
15. בדמו הנל ניתן לראות שכל פעם שאנחנו יוצרים WORKER חדש למעשה אנחנו מקבלים קישור לWORKER הראשון שנוצר. אנחנו מעבירים הודעות ומיד מקבלים תשובה מאותו אחד.
    1. ברגע שנהרוג את הWORKER אין לנו יותר דרך לדבר איתו שוב.
16. FILE API
17. DEMOS
18. מה ניתן לעשות בעזרת ה FILE API  
    התכונות והאובייקטים המוצגים פה משותפים גם לFILESYSTEM API כאשר FSAPI מרחיב אותם ומספק API מלא לFILE SYSTEM כמו צד שרת, עבודה עם ספריות, יצירת ושינוי קבצים מקומיים וכו.
    1. לייבא תוכן של קובץ מהרשת או מהמחשב המקומי לדוגמא הדמו של COPY PASTE
    2. ליצור קבצים וספריות
    3. לשנות קבצים קיימים וכו
19. FILE API
    1. הAPI מספק לנו את האובייקטים הבאים לעבודה עם קבצים:
       1. FILE – מעצם שמו זהו REFRENCE לקובץ
       2. FILE LIST – רשימה של קבציםץ למשל כאשר בוחרים ספריה נקבל רשימה של כל הקבצים בתוכה
       3. BLOB – כמו שכבר ראינו לפני כן בWORKER אפשרות ליצור קבצים ON THE FLY
       4. DRAG AND DROP – ניתן לגרור קבצים לשדה FILE, להדביק על הדף וכו.
20. כל הפעולות שתארנו עד כה מתבצעות דרך אובייקט יחיד FILE READER.
    1. אחרי שיש בידינו REFRENCE לFILE על מנת לעשות פעולה כלשהיא איתו אנחנו צריכים להשתמש בFILE REDER.
    2. אובייקט זה מכיל בתוכו את כל הפעולות שנצטרך לבצע והוא המוח מאחורי כל הAPI
    3. קריאה של קבצים מתבצעת בצורה אסינכרונית
    4. בעזרת FILE READER ניתן לנטר את התקמות הקריאה, הודעות שגיאה במקרה ויש כאלו, לדעת מתי הקובץ נטען ועוד.
21. FILE READER מספק 4 דרכים שונות לקרוא את התוכן של כל קובץ:
    1. BINARY – רצף של בתים בינריים
    2. TEXT – קריאה של טקסט כאשר ישנה אפשרות להעביר כפרמטר את ENCODING שנרצה לקודד בו את התשובה
    3. DATA URL – נושא בפני עצמו ולא נרחיב על זה אבל מצורפת דוגמת קוד.  
       בעיקרון DATA URL מאפשרים לנו לשלוח DATA בפורמט דומה לBASE64
    4. ARRAY BUFFER – הצורה הכי נפוצה לקריאה של קבצים. כזכור לכם הזכרנו שבכרום WORKER יכולים להעביר מידע שהוא ARRAY BUFFER. צורה זו מאפשר להעביר כל תוכן בינארי שנרצה כגון תמונות, מסמכים, קבצים וכו.
22. FILE READER מאפשר לנו להרשם לEVENTS הבאים.
23. דוגמאות:
    1. רישום של EVENT CHANGE לשדה מסוג FILE
    2. ברגע שהמשתמש יבחר או יזרוק קבצים על השדה הנ"ל אנחנו נקבל רשימה של הקבצים שנבחרו.
    3. בדוגמא הנ"ל אנחנו נקרא את הMETADATA של הקבצים
24. קריאה של חלק מתוכן הקובץ:
    1. אם זכור לכם הדמו שהציג את המידע 3IDV מתוך ה3MP ניתן לקרוא את המידע תוך שימוש בSLICE. בדמו ההוא השתמשתי באובייקט אחר שנקרא DATAVIEW שגם זה אובייקט שמאפשר קריאה של בתים מתוך קובץ.
25. הדמו של העתקה והדבקה של תמונות.  
    זה כל הקוד שנמצא בדף. כמו שאתם רואים אני הגבלתי את ההדבק לסוג מסוים של תמונות וכמובן שאפשר להגביל מה שרק רוצים ולבצע VALIDATION לקבצים בצד הלקוח לפני תחילת העבודה או לפני שהם נשלחים לשרת.  
    למשל ניתן להגביל העלאת סוגים מסוימים של קבצים או להגביל את גודל הקובץ שניתן להעלות וכו.  
    כמו שאתם רואים בדוגמא הזו נעשה שימוש בBLOB שבו נתקלנו בעבר בWEB WORKERS.
26. אחד הדברים היפים שכבר מימשו עבורנו הוא היכולת לנטר בזמן אמת את תהליך ההתקדמות של העבודה על קובץ. למשל מי שמכיר מGMAIL את הATTACHMENTS שמראה את ההתקדמות של הקובץ שעשינו עבור ATTACH.