**שקף 3**

אז בואו עכשיו נחזור קצת אחורה ל-event loop . בואו נדבר על ה-event loop של JavaScript באופן כללי. ה-event loop של javascript גם ב-client side עובד באופן די דומה ל-event loop של Node – Node בעצם התחיל מניסיון של Rayen Dahl להריץ את ה-event loop של javaScript ב-server . אז העיקרון של לא לחסום את ה-event loop בעצם התפתח לתכנות א-סינכרוני.

ב-high level , הפרדיגמה של תכנות א סינכרוני באמת בדרך כלל ממומשת על ידי הכנסת events ל-queue והוצאה שלהם אחד אחרי השני.

ב-JavaScript , כדי להשיג תכנות א-סינכרוני, הפוקציות בשפה הן first class citizens , הן יכולות להיות מועברות כמשתנה לפןנקציות אחרות. ככה נולדו callbacks –אתה מעביר פונקציה אחת לפןנקציה אחרת כפרמטר, וכשה-execution של פונקציה מסתיים היא קוראת לפונקציה השניה. אין return values , רק פונקציות שקוראות אחת לשניה ומעבירות את הvalue .

**שקף 4 :**

אז יש כאן דוגמה מאוד בסיסית לתכנות א-סינכרוני, יש פה אובייקט שמבצע פעולת save , ולאחר שפעולת ה-save מתבצעת נקראת הפונקציה שמוגדרת כאן.

שימו לב שהארגומנט הראשון שמועבר לפוקציה הוא ה-error . זו קונבנציה ידועה בתכנות א-סינכרוני. אם יש error כדי שהפונקציה בעצם תטפל בו. ספציפית כאן אין return value והפונקציה מדפיסה Done , אבל יש return value הוא יבוא אחרי ה-error כארגומנט שני.

אני גם רוצה להוסיף שיכולים להיות כמה return values שיבואו כארגומנטים לפונקציה לאחר הארגומנט השני.

**שקף 5:**

אז בואו קצת נדבר על הבעיות של הגישה הזו.

**שקף 6:**

אז אני רוצה להראות לכם את הקוד הנפלא הזה. קחו 30 שניות ותנסו להבין מה כתוב כאן (-:

זה קוד client אבל גם ב-server אפשר לייצר כאלה דברים ממש בקלות. מה שאנחנו מנסים לעשות כאן זה ללכת ל-main window , לפתוח את ה-file menu , לפתוח עוד menu , אנחנו בסופו של דבר יצור alert שבו כתוב Doom .

הנקודה היא שזה קוד ספגטי ממש מטורף קודם כל, והנקודה השניה היא שזה ממש אבל ממש לא קריא. זה בעצם נקרא callback hell – בטח שמעתם על המושג הזה, ומאוד קל לייצר את זה .

**שקף 7:**

הדבר הבא שבדרך כלל יוצר בעיות הוא טיפול בשגיאות – או אי טיפול בשגיאות. מאוד קל לפספס את הטיפול ב-error כשהוא מועבר ככה, הרבה יותר קשה מאשר בתכנות סינכרוני. שימו לב לאיך שהקוד כתוב בדוגמא הזו – כדי לטפל ב-error אנחנו קוראים ל-callback עם ה-error , אחרת הוא נקרא עם null ועם ה-return value . זו נקודה שיכולה מאוד לבלבל.

יש כאן גם פעולה סינכרונית בקוד, json parse , וצריך גם להכניס כאן try ו-catch כדי לטפל בשגיאות של ה-json parse ולא רק בשגיאות שחוזרות מה-read file . בכל אופן נקבל בלאגן של error handling כאשר נסיים עם הקוד .

**שקף 8:**

אז כאן בעצם יש המשך של הדוגמא אחרי שסיימנו עם הקוד – אחרי שסיימנו עם הקוד הא-סינכרוני אתם רואים שיש כאן בלאגן והקוד נראה ממש לא טוב.

**שקף 8:**

הדבר האחרון שאני רוצה להגיד, הוא שאם עבדתם הרבה עם callbacks , ובמיוחד עם plane callabcks , אז אתם יודעים כמה מסובך זה להריץ דברים ב-parallel , ב-series , אפילו פעולות כמו מיפוי מערכים הן מאוד קשות לכתיבה כשכל מה שיש לנו הוא רק callbacks פשוטים.

**שקף 9:**

אז הפתרון הראשון לבעיה היה ה-async library , אתם יכולים לעבור עליו, יש שם הרבה מאוד פתרונות לבעיות , parallel , series , retry ועוד. אבל זה עדיין לא היה מספיק פשוט וחיפשו פתרונות אחרים.עדיין היה מסובך לכתוב ככה קוד.

**שקף 10:**

אז עכשיו בואו נגיע לנושא של השיעור – promises , מה הם , איך אפשר להתשתמש בהם ואיזה בעיות הם פותרים.

**שקף 11:**

אז מה זה promise בעצם ?

Promise הוא proxy עבור ערך שאנחנו עדיין לא יודעים כאשר ה-promise יתבצע. הוא מאפשר לנו בעצם לרשום handlers לevent של הצלחה או כישלון של action א סינכרוני .

ככה גם מתודות א סינכרוניות מחזירות ערך כמו מתודות סינכרוניות – מתודה א סינכרונית תחזיק promise עבור הערך של הפונקציה בעתיד.

**שקף 12:**

אז לpromise בעצם יש 3 מצבים: pending , fulfilled או rejected .

כאשר promise נוצר הוא נוצר בעצם במצב של pending , בשלב זה אנחנו עדיין לא יודעים מה יהיה הערך שלו . אין לו עדיין ערך והוא עדיין לא התבצע.

לאחר שהפעולה הא-סינכרונית מתבצעת , במידה והיא חוזרת בהצלחה , אז ה-promise עבור למצב של fulfilled , ואם הפעולה נכשלת , ה-promise עובר למצב של rejected . בעצם מ-pending הוא יכול לעבור או למצב של fulfilled או למצב של rejected , ובמידה ומשתמשים באותו אובייקט promise , אז במיידי מגיעים ל-event של ההצלחה או הכישלון מבלי שהפעולה הא-סינכרונית מתבצעת בשנית.

ברגע שפרןמיס מגיע לstate של fulfilled או Rejected – הstate שלו לא ישתנה ויישאר קבוע

**שקף 13:**

אז בואו נדבר על איך הקוד נראה עם promises .

אז שימו לב שה-promises משתמשים גם ב-callbacks . גם ל-then ולcatch רושמים callbacks , כאשר ה-callback של ה-then יתבצע כשה-async activity יחזור בהצלחה ןה-promise יהיה במצב fulfilled , וה-catch יתבצע כאשר ה-activity יכשל וה-promise יהיה במצב rejected .

**שקף 14:**

אז בואו עכשיו נדבר על איך בעצם אנחנו משתמשים ב-promises כאשר כל הספריות ב-Node חושפות async API . יש כאן דוגמא שבה בעצם אנחנו כותבים פונקציה שמחזירה new promise , כאשר ה-promise מבצע פעולת database . במידה וחוזרות תוצאות נקרא ל-resolve function , ובמידה וחוזר error מה-DB נקרא ל-Reject function . כך נוכל לעטוף כל API ולהשתמש ב-promise מבחוץ.

**שקף 15:**

עכשיו בואו נדבר על control flows - ואיך ממשמשים series –parallel עם promises .

**שקף 16:**

תסתכלו על השקף הבא. כאן בעצם יש דוגמא לאיך עושים chaining עם promises . ה-pattern הזה יכול להתחליף גם מימושים של async series ו-async waterfall (async waterfall נותן גם להעביר תוצאות בין הקריאות) כאשר השימוש ב-promises נותן לנו קוד הרבה יותר שטוח וקוד שהרבה יותר קל לקרוא.

Chaining אפשר לעשות כאשר מחזירים promise מתוך then . על ה-promise שחוזר, ניתן להפעיל then שוב, כאשר אנחנו מקבלים promise עבור אותו הערך מחוץ ל-Then .

**שקף 17:**

הדרך הכי קצרה בעצם לעשות parallel execution היא לעבוד עם promise.all . Promise.all מקבל מערך של promises, עושה scheduling ל-asynchronous activities אחת אחרי השניה, ומחזיר promise אחד שהוא resolved אחרי שכל ה-promises הן resolved. יהיה פה parallel execution אם כל async activity תהיה offloaded ל-worker pool thread , וכך באמת נקבל parallel , כי ה-scheduling יכול לקרות רק אחד אחרי השני בגלל טבעה של ה-event loop של Node .

**שקף 18:**

אז רק כדי להכיר לכם כמה ספריות promise שיש. אז מאז ES6 ישנן ספריות promise built in ב-Node .

אבל אני ממש ממליצה לכם להסתכל על bluebird . זו לדעתי הספריה הכע טובה שיש. גם נותנת הרבה מאוד פונקציונליות שאין ב-built in promises , נתתי כאן כמה מתודות מהAPI , גם נותנת performance הרבה הרבה יותר טוב וכו.

**שקף 19:**

קצת על מתי promises שימשו אותי – הייתי צריכה לפתור בעיה של first initialization ב-javascript , כמו למשל , אם אתם מכירים java אז יש לכם singleton ואתם צריכים שהוא יהיה thread safe כדי שה-instance לא יווצר פעמיים , אז גם פה – הייתי צריכה לדאוג שהאתחול הראשוני יקרה פעם אחת. מימשתי את זה עם promise שפתרו לי את הבעיה הזו. בשלב מאוחר יותר בקורס ארחיב על איך פתרנו את הבעיה הזו ואיך השתמשנו ב-promise כדי לפתור אותה.