מבוא ל-Backend ול-Node.js

ה-Backend הוא החלק של יישום אינטרנט הפועל על השרת ואחראי על עיבוד נתונים, ביצוע חישובים ותקשורת עם שרתים אחרים. הוא מורכב בדרך כלל ממסד נתונים, שרת אינטרנט ושרת יישומים המקיים אינטראקציה עם מסד הנתונים ומבצע את הלוגיקה העסקית של היישום.

ה-Backend הוא מרכיב חיוני בפיתוח אינטרנט מכיוון שהוא מאפשר אחסון ואחזור נתונים, עיבוד נתונים וביצוע לוגיקה בצד השרת. ללא backend, יישומי אינטרנט לא יוכלו לספק תוכן דינמי או לבצע חישובים מורכבים.

החשיבות של הלוגיקה העסקית ב-Backend

בעוד שמשימות צד שרת כגון פעולות CRUD ואימות משתמשים הן חיוניות, המטרה העיקרית של backend היא ליישם את הלוגיקה העסקית של יישום אינטרנט. לוגיקה עסקית היא מערכת הכללים והתהליכים המגדירים כיצד יש לעבד נתונים וכיצד האפליקציה צריכה להגיב לקלט של משתמשים.

הלוגיקה העסקית הוא מה שמייחד את האפליקציה ומה שנותן לה את הערך שלה. זה כולל אלגוריתמים מורכבים, בינה מלאכותית, למידת מכונה, עיבוד Big Data, ארכיטקטורה מורכבת וסקיילביליות.

לדוגמה, ה-backend של פייסבוק כולל אלגוריתמים מסובכים שמנתחים נתוני משתמשים וממליצים על תוכן מותאם אישית על סמך תחומי העניין של המשתמש. ה-backend של נטפליקס משתמש בלמידת מכונה כדי לחזות אילו סרטים ותוכניות טלוויזיה המשתמשים עשויים לאהוב וממליץ עליהם בהתאם. ה-backend של YouTube משתמש בעיבוד Big data כדי לנתח את התנהגות המשתמש ולהתאים אישית את חוויית המשתמש.

מרכיבי המפתח של ה-Backend

ה-backend מורכב משני מרכיבים מרכזיים: שרת האינטרנט ומסד הנתונים. שרת האינטרנט ומסד הנתונים הם שני מרכיבים מרכזיים של ה-backend הפועלים יחד כדי לספק את הפונקציונליות ואת אחסון הנתונים הנדרשים על ידי יישום אינטרנט.

שרת אינטרנט

שרת האינטרנט הוא מרכיב מרכזי ב-back end המשמש כממשק בין הלקוח הלוגיקה העסקית של האפליקציה. שרת האינטרנט אחראי לטיפול בבקשות נכנסות מלקוחות, ביצוע הלוגיקה המתאימה של האפליקציה והחזרת תגובה חזרה ללקוח.

ביישום אינטרנט מבוסס Node.js, שרת האינטרנט מיושם בדרך כלל באמצעות מודול HTTP או HTTPS המובנה, המספק דרך פשוטה ויעילה ליצור שרת אינטרנט. שרת האינטרנט עשוי להיות מתוגבר גם בתוכנת ביניים נוספת (Middleware), כגון אימות משתמשים, כדי לספק פונקציונליות ואבטחה נוספים.

בנוסף ללוגיקה העסקית של האפליקציה, שרת האינטרנט אחראי גם על הגשת קבצים סטטיים, כגון קבצי HTML, CSS ו-JavaScript, וכן כל נכס אחר הנדרש על ידי האפליקציה.

מסד הנתונים

מסד הנתונים הוא מרכיב מרכזי נוסף ב-back end שאחראי על אחסון וניהול נתוני האפליקציה. מסד הנתונים משמש בדרך כלל לאחסון נתונים שנוצרים או נצרכים על ידי האפליקציה, כגון פרופילי משתמשים, רשימות מוצרים או רשומות של עסקאות.

ביישום אינטרנט מבוסס Node.js, מסד הנתונים מיושם בדרך כלל באמצעות מערכת ניהול מסד נתונים (Database Management System - DBMS) כגון MongoDB, MySQL או PostgreSQL. ה-DBMS מספק דרך לאחסן ולשלוף נתונים באמצעות פורמט מובנה, כגון SQL או JSON.

ניתן גם להגביר את יכולות מסד הנתונים עם כלים נוספים, כגון מסגרות מיפוי יחסי אובייקט (Object-Relational Mapping - ORM) כדי לספק פונקציונליות וביצועים נוספים.

איך האינטרנט עובד

לפני שנצלול לפרטים של Node.js, חשוב להבין איך האינטרנט עובד.

האינטרנט מבוסס על ארכיטקטורת שרת-לקוח, כאשר הלקוח (בדרך כלל דפדפן אינטרנט) שולח בקשות לשרת, והשרת מגיב בנתונים המוצגים בדפדפן.

כאשר משתמש מקליד כתובת URL בדפדפן אינטרנט ולוחצים על אנטר, הדפדפן שולח בקשת HTTP לשרת, אשר מגיב במסמך HTML שמעובד בדפדפן.

הדפדפן יכול גם לשלוח בקשות נוספות לשרת כדי לאחזר תמונות, גיליונות CSS, קובצי JavaScript ומשאבים אחרים הדרושים לעיבוד הדף.

אתרים סטטיים מול דינמיים

ישנם שני סוגים של אתרים: סטטיים ודינמיים ושני סוגים של גישות לעיבוד תוכן HTML באינטרנט: עיבוד בצד השרת ועיבוד בצד הלקוח. נרחיב על שתי הגישות הללו בהמשך מאמר זה.

אתר סטטי הוא אתר שמורכב מקובצי HTML, CSS ו-JavaScript המוגשים ישירות לדפדפן המשתמש. התוכן באתר סטטי אינו משתנה בתדירות גבוהה והוא בדרך כלל מקודד בקובצי HTML קבועים שאינם משתנים כלל.

אתר דינמי, לעומת זאת, הוא אתר שמשתמש בסקריפטים בצד השרת כדי ליצור דפי HTML באופן דינמי על סמך קלט המשתמש או גורמים אחרים. אתרים דינמיים משתמשים בדרך כלל בשרת backend כדי לאחסן ולאחזר נתונים, והם משתמשים בשפות סקריפטים בצד השרת כמו PHP, Python או Ruby כדי ליצור דפי HTML.

עיבוד צד שרת (server-Side Rendering) לעומת עיבוד צד לקוח (Client-Side Rendering)

עיבוד בצד השרת ועיבוד בצד הלקוח הן שתי גישות לעיבוד תוכן HTML באינטרנט.

עיבוד בצד השרת הוא תהליך יצירת תוכן HTML בשרת ושליחתו לדפדפן של הלקוח. גישה זו משמשת בדרך כלל לאתרים בעלי כמות גדולה של תוכן סטטי, שאינם דורשים אינטראקטיביות רבה.

עיבוד בצד הלקוח, לעומת זאת, הוא תהליך של יצירת תוכן HTML בדפדפן של הלקוח באמצעות JavaScript. גישה זו משמשת בדרך כלל לאתרים הדורשים רמה גבוהה של אינטראקטיביות ותוכן דינמי. צד הלקוח של אתרים אלו בנוי בדרך כלל עם ספריות צד לקוח או פריימוורקים כגון React, Vue או Angular.

שרתי API

ראינו שיש שרתים שאחראים על עיבוד בצד השרת והם יוצרים דפי HTML שיצרכו על ידי הדפדפן של הלקוח. אבל עם התקדמות צד הלקוח, כפי שהוזכר לעיל, שבו התצוגות נוצרות ישירות בצד הלקוח, השרתים לא צריכים לספק יותר תוכן HTML. כעת הם יכולים לחשוף רק את הנתונים שיצרכו על ידי הלקוח בצד הלקוח בכל דרך אפשרית.

עם עלייתן של ספריות צד לקוח ופריימוורקים כגון React, Angular ו-Vue.js, העיבוד בצד הלקוח הפך לנפוץ יותר. בגישה זו, השרת אחראי רק לספק את הנתונים ללקוח בפורמט שניתן לצרוך בקלות על ידי קוד JavaScript, כגון JSON.

שרתי API הם סוג של שרתים שתוכננו במיוחד לספק נתונים ליישומי לקוח בצורה עקבית ואמינה. הם משמשים בדרך כלל בפיתוח אתרים כדי לספק גישה לבסיסי נתונים או לשירותי אינטרנט.

שרתי API בדרך כלל מספקים נתונים בפורמט כגון JSON או XML, שניתן לצרוך אותם בקלות על ידי קוד JavaScript בצד הלקוח. זה מאפשר ליישומי לקוח לצרוך נתונים משרת ה-API מבלי צורך להבין את הטכנולוגיה הבסיסית המשמשת ליצירת הנתונים.

אחד היתרונות העיקריים של שימוש בשרת API הוא בכך שהוא מאפשר גמישות וסקלביליות רבה יותר בפיתוח אפליקציות. על ידי ניתוק הרכיבים בצד הלקוח מאלה שבצד השרת של האפליקציה, מפתחים יכולים לשנות ולעדכן כל רכיב בקלות רבה יותר מבלי להשפיע אחד על השני.

יתרון נוסף בשימוש בשרת API הוא בכך שהוא מאפשר יכולת התממשקות גמישה יותר בין יישומים ופלטפורמות שונות. על ידי מתן API סטנדרטי לגישה לנתונים בלבד, שרתי API יכולים לשרת הן יישומי אינטרנט, הן יישומים של טלפונים ניידים והן יישומי שולחן עבודה בצורה אחידה, עקבית ואמינה.

מבוא ל-Node.js

Node.js היא סביבת זמן ריצה חוצה פלטפורמות בקוד פתוח המאפשרת למפתחים להריץ קוד JavaScript מחוץ לדפדפן אינטרנט. הוא נבנה על גבי מנוע V8 JavaScript של גוגל ומשתמש במודל I/O[1] מונע אירועים (event-driven), לא חוסם (non-blocking) שהופך אותו למתאים ביותר לבניית יישומי רשת סקיילביליים בעלי ביצועים גבוהים.

Node.js שוחרר לראשונה בשנת 2009 ומאז זכה לאימוץ נרחב בקהילת המפתחים בשל קלות השימוש, המהירות והסקלביליות שלו. הוא משמש לבניית שרתי אינטרנט, כלי שורת פקודה, יישומי שולחן עבודה והתקני IoT.

סקירה כללית של ארכיטקטורת Node.js

Node.js בנוי על גבי מספר רכיבים, כולל libuv , V8 , Node.js ו- C++ . בואו נסתכל מקרוב על כל אחד מהרכיבים הבאים:

1. **Node.js:** Node.js היא סביבת זמן הריצה שמבצעת קוד JavaScript מחוץ לדפדפן אינטרנט.
2. **V8:** V8 הוא מנוע JavaScript בעל ביצועים גבוהים שממיר קוד JavaScript לקוד מכונה.
3. **libuv:** libuv היא ספרייה חוצת פלטפורמות המספקת לולאת אירועים (event loop), יכולות קלט/פלט אסינכרונית ו-thread pool.
4. **C++:** סביבת Node.js כתובה ב-C++, מה שמאפשר לה להיות יעילה ובעלת ביצועים גבוהים במיוחד.

Node.js event loop

ה-event loop היא הליבה של Node.js ואחראית לטיפול בבקשות מהלקוח וביצוע callbacks. כאשר מתקבלת בקשה, היא מתווספת ל-event loop, וכאשר callback מבוצע, הוא מתווסף חזרה לevent loop.

ה-event loop משתמשת בתור לניהול סדר הביצוע של callbacks, והיא משתמשת בתבנית ה-event emitter[2] כדי להודיע למנויים כאשר מתרחשים אירועים.

Processes, Threads, and the Thread Pool

Node.js משתמש ב-event loop עם תהליכון יחידי כדי לטפל בבקשות, אך הוא משתמש גם ב-Thread Pool כדי לטפל בפעולות יקרות שלא ניתן לטפל בהן על ידי event loop.

---

[1] בהקשר של Node.js המונח I/O (קלט - Input ופלט - Output) מתייחס בדרך כלל לאופן שבו סביבת זמן הריצה של Node.js מטפלת בפעולות קלט ופלט כגון קריאה וכתיבת קבצים, ביצוע בקשות רשת וטיפול בקלט משתמש.

[2] בבסיסה, תבנית ה-event emitter כוללת שני מרכיבים עיקריים: event emitter ומאזינים לאירועים. ה-event emitter הוא אובייקט שפולט אירועים, ו-event liteners הם פונקציות הרשומות לטיפול באירועים ספציפיים.
כאשר מתרחש אירוע, ה-event emitter מודיע לכל מאזיני האירועים הרשומים על ידי קריאה לפונקציות המשויכות להם.
מאזיני האירועים יכולים לאחר מכן לבצע את כל הפעולות הנדרשות בתגובה לאירוע.

כאשר נתקלים בפעולה יקרת משאבים, כגון קריאת קובץ או בקשת רשת, Node.js מעביר את הפעולה ל-worker thread במאגר ה-threads, אשר מבצע את הפעולה ומחזיר את התוצאה ל-event loop הראשי.

זה מאפשר ל-Node.js לטפל במספרים גדולים של חיבורים במקביל מבלי לחסום את ה-event loop.

אירועים ואדריכלות מונעת אירועים (Events and Event-driven Architecture)

אירועים הם מושג ליבה ב-Node.js ומשמשים ליישום ארכיטקטורה מונעת אירועים.

אירוע הוא אות לכך שהתרחשה פעולה מסוימת, כגון קובץ שנקרא, בקשת רשת שהושלמה או טיימר שפג תוקפו.

ב-Node.js, אירועים מיושמים באמצעות מחלקת ה-`EventEmitter`, המאפשרת למנויים לרשום event liteners ולקבל התראות כאשר מתרחשים אירועים.

ארכיטקטורה מונעת אירועים מתאימה היטב לבניית יישומים סקיילביליים ובעלי ביצועים גבוהים מכיוון שהיא מאפשרת I/O אסינכרוני ולא חוסם ומנתקת את הרכיבים של אפליקציה.

Blocking vs Non-Blocking Code

קוד שאינו חוסם חיוני ב-Node.js מכיוון שהוא מאפשר לה לטפל במספרים גדולים של חיבורים במקביל מבלי לחסום את ה-event loop. לכן חשוב להבין את ההבדל בין קוד חוסם לקוד שאינו חוסם.

קוד חוסם הוא קוד שמחכה לסיום פעולה מסוימת לפני מעבר לשורת הקוד הבאה. לדוגמה, נראה את הקוד הבא שקורא קובץ מהדיסק:

```
const fs = require('fs');

const data = fs.readFileSync('myfile.txt');

console.log(data.toString());
```

בדוגמה זו, `readFileSync` היא מתודה סינכרונית כך שהיא חוסמת את ה-event loop עד לקריאת הקובץ, ומונעת ביצוע של פעולות אחרות. תארו לעצמכם אם הקובץ הזה היה קובץ גדול מאוד לקריאה מהדיסק, אז פעולה זו עלולה היתה להימשך זמן רב.

קוד לא חוסם, לעומת זאת, אינו חוסם את ה-event loop ומאפשר המשך פעולות אחרות בזמן שהפעולה הנוכחית מתבצעת. לדוגמה, נראה את הקוד הבא שקורא קובץ עם המתודה האסינכרונית `readFile`:

```
const fs = require('fs');
fs.readFile('myfile.txt', (err, data) => {
if (err) throw err;

console.log(data.toString());
});
```

בדוגמה זו, מתודת `readFile` לוקחת פונקציית callback המופעלת כאשר הקובץ נקרא, ומאפשרת לפעולות אחרות להמשיך בזמן קריאת הקובץ.

מה אנחנו לא יכולים לעשות עם Node.js

בעוד ש-Node.js היא סביבת זמן ריצה חזקה ויעילה שמתאימה היטב לבניית יישומי רשת סקיילביליים וביצועים גבוהים, ישנם סוגים מסוימים של יישומים שעבורם היא עשויה להיות לא הבחירה הטובה ביותר.

לדוגמה, ייתכן ש-Node.js אינה מתאימה ליישומים הדורשים כוח עיבוד רב. הסיבה לכך היא ש-Node.js היא single-threaded ומשתמשת במודל קלט/פלט מונע-אירועים שאינו חוסם, אשר מותאם לטיפול בפעולות קלט/פלט אך עשוי להיות לא יעיל באותה מידה עבור פעולות הקשורות במעבד. עבור יישומים כאלה, שפות תכנות כגון C++, Java או Python עשויות להיות מתאימות יותר.

ייתכן שגם Node.js אינה הבחירה הטובה ביותר עבור יישומים הדורשים זיכרון רב, שכן ל-Node.js יש תקורה גבוהה יחסית של זיכרון עקב השימוש במנוע V8 JavaScript. במקרים כאלה, שפות תכנות כגון Go או Rust, המיועדות לשימוש יעיל בזיכרון, עשויות להיות מתאימות יותר.

שיקול נוסף הוא הזמינות של ספריות. בעוד של-Node.js יש מערכת אקולוגית גדולה וצומחת של ספריות, ייתכנו סוגים מסוימים של יישומים, ששפות תכנות אחרות שלהן מגוון בוגר יותר או נרחב יותר של ספריות זמינות, יהיו מתאימות יותר.

דוגמאות למקרים שבהם אין להשתמש ב-Node.js

הנה כמה דוגמאות מהחיים האמיתיים לסוגי היישומים שעבורם Node.js היא אולי לא הבחירה הטובה ביותר:

1. **יישומים הדורשים כוח עיבוד רב:** כפי שצויין קודם לכן, ייתכן ש-Node.js אינה מתאימה ליישומים הדורשים כוח עיבוד רב. לדוגמה, יישומים הכוללים משימות חישוביות כבדות כמו קידוד וידאו, למידת מכונה או סימולציות מדעיות עלולות לא להתבצע היטב ב-Node.js. במקרים אלה, שפות תכנות כגון Java ,++C או Python עשויות להיות מתאימות יותר.
2. **יישומים עתירי זיכרון:** ל-Node.js יש תקורה גבוהה יחסית של זיכרון עקב השימוש במנוע V8 JavaScript, שאולי אינו אידיאלי עבור יישומים הדורשים זיכרון רב. לדוגמה, יישומים הכוללים עיבוד כמויות גדולות של נתונים או הפקת דוחות גדולים עשויים שלא לספק ביצועים טובים ב-Node.js. במקרים אלו, שפות תכנות כגון Go או Rust, המיועדות לשימוש יעיל בזיכרון, עשויות להתאים יותר.
3. **יישומי שולחן עבודה:** בעוד ש-Node.js משמשת לעתים קרובות לבניית יישומי רשת ושרתי אינטרנט, ייתכן שהיא אינה הבחירה הטובה ביותר לבניית יישומי שולחן עבודה. הסיבה לכך היא שיישומי שולחן העבודה דורשים בדרך כלל ממשקי משתמש מורכבים יותר ועשויים לכלול לוגיקה ומניפולציית מידע מורכבת יותר מאשר יישומי רשת. עבור יישומי שולחן עבודה, שפות תכנות כגון Java ,#C או Python עשויות להתאים יותר.
באופן כללי, בגלל ש-Node.js היא single thread נחשב מקובל לומר ש-Node.js לא מתאימה כשיש חישוב כבד, כגון משחקים, עיבוד תמונה או ML. אפליקציות כלליות אפשר לבנות עם Node.js. לדוגמה, VSCODE בנויה עם Node.js.
4. **יישומי גרפיקה בזמן אמת:** Node.js עשוי שלא להתאים ליישומים הדורשים עיבוד גרפי בזמן אמת, כגון משחקי תלת מימד או יישומי מציאות מדומה. הסיבה לכך היא שיישומים כאלה דורשים גישה ברמה נמוכה לחומרה ולעיתים קרובות כרוכים באלגוריתמים גרפיים מורכבים שעשויים לא להתבצע בצורה מיטבית ב-Node.js. עבור יישומים כאלה, שפות תכנות כגון ++C או אפילו מנועי משחק מיוחדים עשויים להתאים יותר.

בסך הכל, בחירת שפת התכנות וסביבת זמן הריצה תלויה במגוון גורמים, לרבות דרישות האפליקציה, הניסיון והמומחיות של צוות הפיתוח וזמינות הספריות והפריימוורקים.

יתרונות השימוש ב-Node.js

Node.js מציע מספר יתרונות על פני טכנולוגיות מסורתיות בצד השרת כגון PHP, Ruby ו-Java. חלק מהיתרונות הללו כוללים:

1. **מהירות וסקיילביליות:** Node.js בנוי על גבי V8, מנוע JavaScript בעל ביצועים גבוהים, המאפשר לו לבצע קוד JavaScript הרבה יותר מהר מאשר טכנולוגיות מסורתיות בצד השרת.
2. **קלט/פלט לא חוסם:** Node.js משתמש במודל קלט/פלט לא חוסם המאפשר לו להתמודד עם מספר רב של חיבורים בו-זמניים מבלי לחסום את ה-event loop, מה שהופך אותו לסקיילבילי ויעיל ביותר.
3. **JavaScript:** סביבת Node.js מאפשרת למפתחים להשתמש באותה שפת תכנות (JavaScript) הן בצד הלקוח והן בצד השרת, מה שמקל על פיתוח ותחזוקה של יישומי אינטרנט.
4. **מערכת אקולוגית עשירה:** ל-Node.js יש מערכת אקולוגית עשירה של מודולים וספריות שניתן לשלב בקלות ביישומי אינטרנט, מה שהופך את הפיתוח למהיר ויעיל יותר.

דוגמאות למקרים שבהם מומלץ להשתמש ב-Node.js

הנה כמה דוגמאות מהחיים האמיתיים לסוגי היישומים המתאימים היטב ל-Node.js:

1. **יישומי אינטרנט בזמן אמת:** Node.js אידיאלית לבניית יישומי אינטרנט בזמן אמת הדורשים הרבה אינטראקטיביות וזמני תגובה מהירים, כגון יישומי צ'אט, פלטפורמות מדיה חברתית וכלי שיתוף פעולה. לדוגמה, יישומים כמו Slack, Trello ו-Asana משתמשים כולם ב-Node.js כדי להפעיל את תכונות שיתוף הפעולה שלהם בזמן אמת.
2. **יישומי סטרימינג:** Node.js מתאימה היטב לבניית יישומי סטרימינג הדורשים עיבוד בזמן אמת של כמויות גדולות של נתונים, כגון שירותי הזרמת וידאו, שירותי הזרמת מוזיקה ופלטפורמות משחק מקוונות. לדוגמה, יישומים כמו Netflix ו-Twitch משתמשים כולם ב-Node.js כדי להפעיל את שירותי הסטרימינג שלהם.
3. **ממשקי API ומיקרו-סרביסים:** Node.js אידיאלית לבניית ממשקי API ומיקרו-סרביסים הדורשים עיבוד בצד שרת מהיר וסקיילבילי. לדוגמה, יישומים כמו PayPal, LinkedIn ואובר משתמשים כולם ב-Node.js כדי להפעיל את ממשקי ה-API והמיקרו-סרביסים שלהם.
4. **יישומי Single page:** סביבת Node.js מתאימה היטב לבניית יישומי SPA הדורשים ממשקי משתמש מהירים ומגיבים. לדוגמה, יישומים כמו PayPal, LinkedIn ו-Walmart משתמשים כולם ב-Node.js כדי להפעיל את היישומים שלהם בעמוד אחד.
5. **יישומי IoT:** סביבת Node.js אידיאלית לבניית יישומי האינטרנט של הדברים (IoT) הדורשים עיבוד בזמן אמת של נתוני חיישנים וסוגים אחרים של זרמי נתונים. לדוגמה, יישומים כמו Nest, Raspberry Pi ו-Electrical Imp משתמשים כולם ב-Node.js כדי להפעיל את פלטפורמות ה-IoT שלהם.

בסך הכל, Node.js היא סביבת זמן ריצה רב-תכליתית ועוצמתית שמתאימה היטב לבניית מגוון רחב של יישומים, במיוחד אלה הדורשים עיבוד צד שרת מהיר וסקיילבילי, עיבוד נתונים בזמן אמת וממשקי משתמש מהירי תגובה.

סיכום

לסיכום, backend הוא מרכיב קריטי בפיתוח אתרים, והוא כולל הרבה יותר מסתם פעולות CRUD ואימות משתמשים. הלוגיקה העסקית של אפליקציית אינטרנט הוא המקום בו נמצא הערך האמיתי, והוא כולל אלגוריתמים מורכבים, בינה מלאכותית, למידת מכונה, עיבוד ביג דאטה, ארכיטקטורה מורכבת וסקיילביליות.

Node.js היא סביבת זמן ריצה רבת עוצמה ויעילה שמתאימה היטב לבניית שרתי עיבוד צד שרת נרחבים, סקיילביליים ובעלי ביצועים גבוהים או שרתי API עבור יישומי רשת. היא משתמשת במודל קלט/פלט מונחה אירועים שאינו חוסם המאפשר לה להתמודד עם מספר רב של חיבורים בו-זמניים מבלי לחסום את ה-event loop, מה שהופך אותה לבחירה האידיאלית לבניית יישומי אינטרנט בזמן אמת, יישומי צ'אט, שרתי משחקים ויישומים אחרים עתירי רשת.