

-Team 6-

# Jira - AI Integration

AppsFlyer



In partnership with:



# Content Overview

1. Opening
2. Content Overview
3. Our Team
4. About AppsFlyer
5. Our Part - What everybody did
- 6 - 10. Explanation of the project
11. What Languages & Softwares We Used
- 12- 13. Solution Flow
- 14-24. Code Example
- 25 - 27. Example of Usage
28. Product Value
- 29 - 32. What we learned
33. Thanks

# Our Team

Let's meet our  
team members



GADI KOGEL

324218817



ARIEL HARRIS

212666655



NISIM COHEN

315088609



MOSHE EVYATAR GIDONI

213660368

Mentors: Mark Samuelson | Nir Peled





# About AppsFlyer

AppsFlyer is a company that provides data analytics and performance tracking solutions for mobile apps. It helps developers and advertisers understand the sources of app installs, user behavior, and the effectiveness of advertising campaigns, thereby helping to maximize the return on investment in digital advertising.



# What everybody did

## 1. Gadi and Moshe were in charge of:

putting up a node.js server, connecting the localhost to a static domain via Ngrok, making a http handler that receives tickets from Jira and stores them as JS Objects, making a handler function to send the data obtained to OpenAI API, Sending the response from OpenAI back to Jira as a comment for the administrators.

## Nissim and Ariel were in charge of:

writing an AI prompt, refining it, adjusting the prompt format and temperature, building a jira service management project resembling the company's, making a form request in jira portal with real life categories, making custom fields for each field in the form, sending issues constantly for testing and analysis, making a jira webhook that sends the jira ticket as a json to the node.js server upon ticket creation, making a function that combined regex and custom functions to filter out sensitive data, making a MongoDB and sending the data obtained from the tickets + OpenAI response to the database.

## 2. Gadi and Moshe helped the project with the coding, server management, and architecture.

Nissim and Ariel did the Jira Service management project, managed and configured it, and made and refined the AI prompting

## 3. Our Project helps the company's IT team agents by giving them automatic comments that are added to the ticket by an AI assistant, that'll help the agents optimize their time and figure out solutions faster.





# What is JSM?

## (Jira Service Management)

Jira Service Management (JSM) is a comprehensive platform designed to streamline IT service management processes and optimize communication between users and IT agents.

It offers a user-friendly interface for easy ticket submission, progress tracking, and collaboration. JSM features intuitive ticket submission, assignment to relevant IT agents, efficient communication tools, and robust reporting and analytics capabilities.

The screenshot shows the Jira Service Management (JSM) interface. At the top, there's a navigation bar with 'Jira', 'Your work', 'Projects', 'Filter', 'Dashboards', 'Teams', 'Apps', and 'Create'. Below the navigation is a search bar and a 'GROUP BY' dropdown set to 'None'. The main area is titled 'Projects / Beyond Gravity Board'. On the left, a sidebar menu includes 'Planning' (Timeline, Backlog, Board), 'Development' (Code, Releases), 'Project Pages', 'Add shortcut', and 'Project settings'. The 'Board' option is selected. The board itself has four columns: 'TO DO' (6 items), 'IN PROGRESS' (6 items), 'IN REVIEW' (6 items), and 'DONE' (6 items). Each column contains cards with ticket details like 'Optimize experience for mobile web' (NUC-344), 'Fast trip search' (NUC-342), 'Affiliate links integration - frontend' (NUC-335), 'Onboard workout options (OWO)' (NUC-360), 'Quick booking for accommodations - website' (NUC-336), 'Revise and streamline booking flow' (NUC-367), 'Travel suggestion experiments' (NUC-358), 'Adapt web app no new payments provider' (NUC-346), 'Customers reporting shopping cart purchasing issues with the BG web store' (NUC-344), and 'BugFix BG Web-store app crashing' (NUC-337). Each card includes a progress bar and a 'More' button.



# Ticketing system challenges

The screenshot shows a ticket creation form in a web-based help center. The form includes fields for raising the request on behalf of a customer, entering a title (e.g., "broken keyboard"), viewing suggested articles related to input devices or monitors, detailing the problem (e.g., "my keyboard is broken, I need a new one plzzzzzzz"), setting priority (Medium), selecting hardware as the type of issue, and categorizing the ticket under "Hardware - Peripherals" with a subcategory of "Hardware-Peripherals-Keyboard". The impact is listed as "The whole department". At the bottom, there are "Send" and "Cancel" buttons.

Help Center / IT-demo

IT-demo

Welcome! You can raise a request for IT-demo using the options provided.

What can we help you with?

Submit a request or report a problem.

Raise this request on behalf of \*

Example Customer (example@atlassian-demo.invalid)

Title \*

broken keyboard

Suggested articles

- ISSUES WITH CONNECTING INPUT DEVICES (KEYBOARD AND OTHERS)  
Keys Not Registering or Typing Incorrect Characters:
- MONITOR INFORMATION AND TROUBLESHOOTING GUIDE  
Perform a monitor self-test (if available) to check for hardware defects.

Details Of Your Problem Or Request \*

Normal text  B  I  ...  A        my keyboard is broken, I need a new one plzzzzzzz

Priority \*

Critical

High

Medium

Low

Hardware Or Software? \*

Hardware

Software Or/And Applications

Category \*

Hardware - Peripherals

Subcategory \*

Hardware-Peripherals-Keyboard

Impact \*

The whole department

Send Cancel

Often, the customer may not select the correct category that accurately reflects the issue or request for which they opened the ticket for.

This could waste some of the IT agent's time and impact how the data is analyzed, causing the BI to be less accurate.

# Our Project Mission



• • •

1. to enhance ticket categorization.
2. to enhance prioritization.
3. to present quick and effective solutions.

# Tackling the Task:

- Meeting with the IT team.
- Researching relevant software/platforms (Jira, OpenAI API).
- Brainstorming and developing a blueprint.



# Splitting up the work

We split up the project to 2 teams -  
Gadi and Moshe were on building a  
server and making routers and etc.  
Ariel and Nissim were on making a  
Jira project and managing the  
system, and making an AI prompt.

# What Languages & Softwares We Used

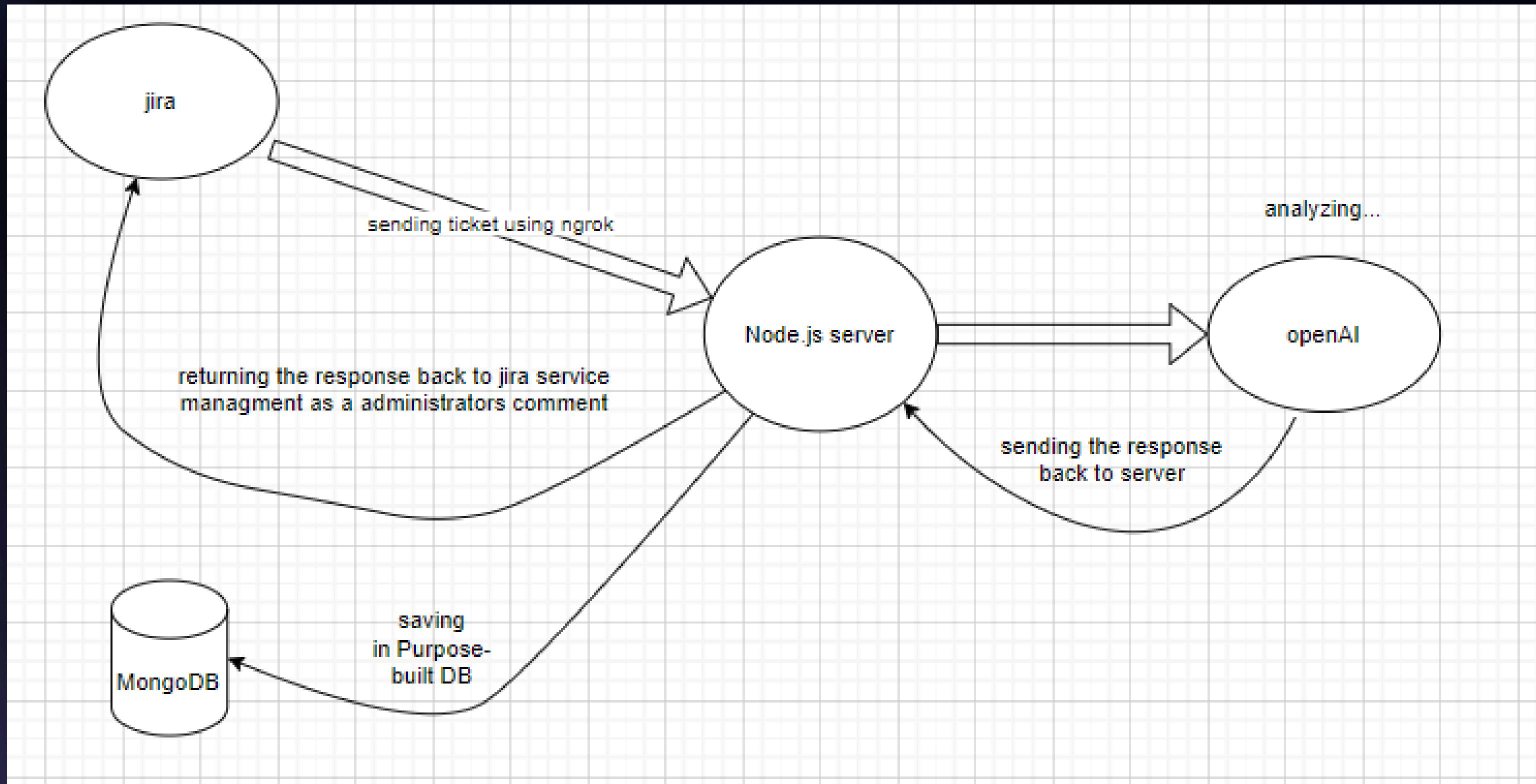
We used Node.js with Express for the coding part of the project. We also worked with Jira, MongoDB, and OpenAI assistant API.

We learned how to use Node.js with Express, how to make a MongoDB, how to build an API, how to handle HTTP requests, how to make an AI Assistant, and how to work with Jira.

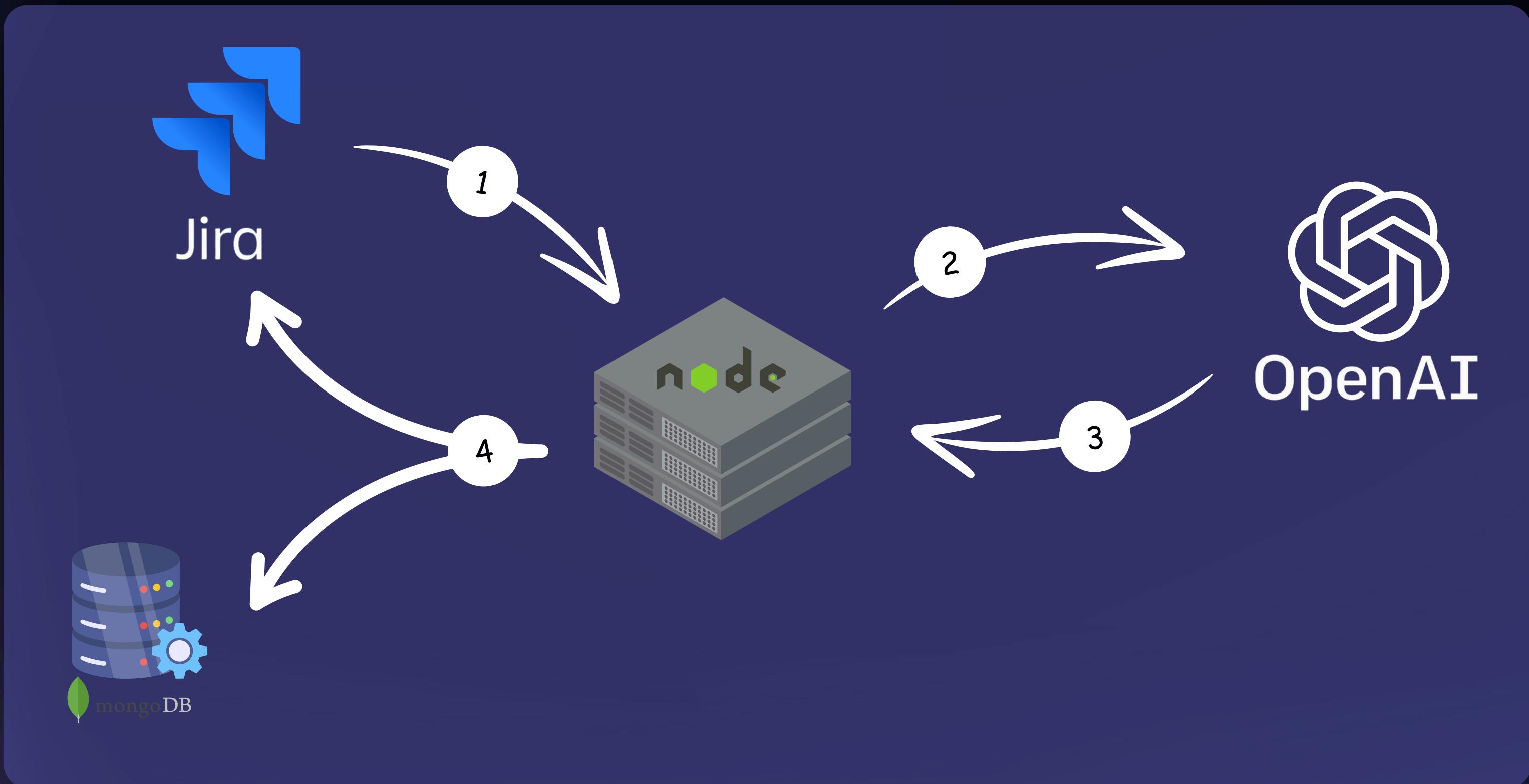
- Node.js is a JavaScript framework that is used to make a server.
- Express is a web application framework for Node.js. It's built for making it easier to handle HTTP requests and responses.
- OpenAI API is an API to connect your application or program to OpenAI Assistant, and gives options of assigning a role for the assistant.
- Jira Service Management is an IT platform. Specifically in our project, we focused on the tickets system, which allows workers to contact the IT team.
- ngrok - ngrok is a secure tunnel service that allows you to expose local servers to the internet effortlessly , using ngrok involves simple setup, configuration, and secure tunneling.
- MongoDB is a database that allows storing data.



# Solution Flow



# 2th Solution Flow



# Moshe & Gadi

## Building Node.js / Express Server:

This is the code of the Node.js Server running with the Express framework. It makes a server with a dedicated port, that upon getting HTTP requests will call the router issuesRouter.

```
1 const express = require("express");
2 const http = require("http");
3 require("dotenv").config();
4 require("./db/mongoConnect");
5 const issuesRouter = require("./issuesRouter.routes");
6
7 // Initialize Express app
8 const app = express();
9
10 app.use(express.json()); // Parse JSON bodies
11 app.use("/", issuesRouter)
12
13 // Create HTTP server using Express app
14 const server = http.createServer(app);
15
16 // Define port number
17 const port = process.env.PORT || 3001
18
19 // Start server and listen for incoming requests
20 server.listen(port, () => {
21     console.log("Server is running on port", port);
22 })
23
24
25
```

# Moshe & Gadi

## Ticket Handler:

Upon sending of ticket by Jira, it is received by our API server that calls these functions:  
deconstruction and filtering - filterIssueData.

Sending the ticket to OpenAI API - NewAIChat.

Sending the AI feedback back to Jira as a comment - addCommentToJira.

Sending the finalized ticket to MongoDB.



```
1 //Route to post issue by trigger
2 router.post("/post", async (req, res) => {
3     try {
4         // Filter issue data
5         const filteredIssueData = filterIssueData(req.body);
6
7         // Get AI feedback for the issue
8         const aiFeedback = await NewAiChat(filteredIssueData)
9
10        // Adding AI feedback to filtered issue data
11        filteredIssueData.aiFeedback = aiFeedback;
12
13        // Add AI feedback as a comment to the issue in Jira
14        await addCommentToJira(filteredIssueData.issueKey, aiFeedba
15            ck);
16
17        // Save issue data to the database
18        const issue = new IssueModel(filteredIssueData);
19        await issue.save();
20
21        console.log("Post issue successful");
22        res.json(issue);
23    } catch (err) {
24        console.error("Error while saving issue:", err);
25        res.status(502).json({ error: "Failed to save issue" });
26    });
}
```

# Nissim & Ariel

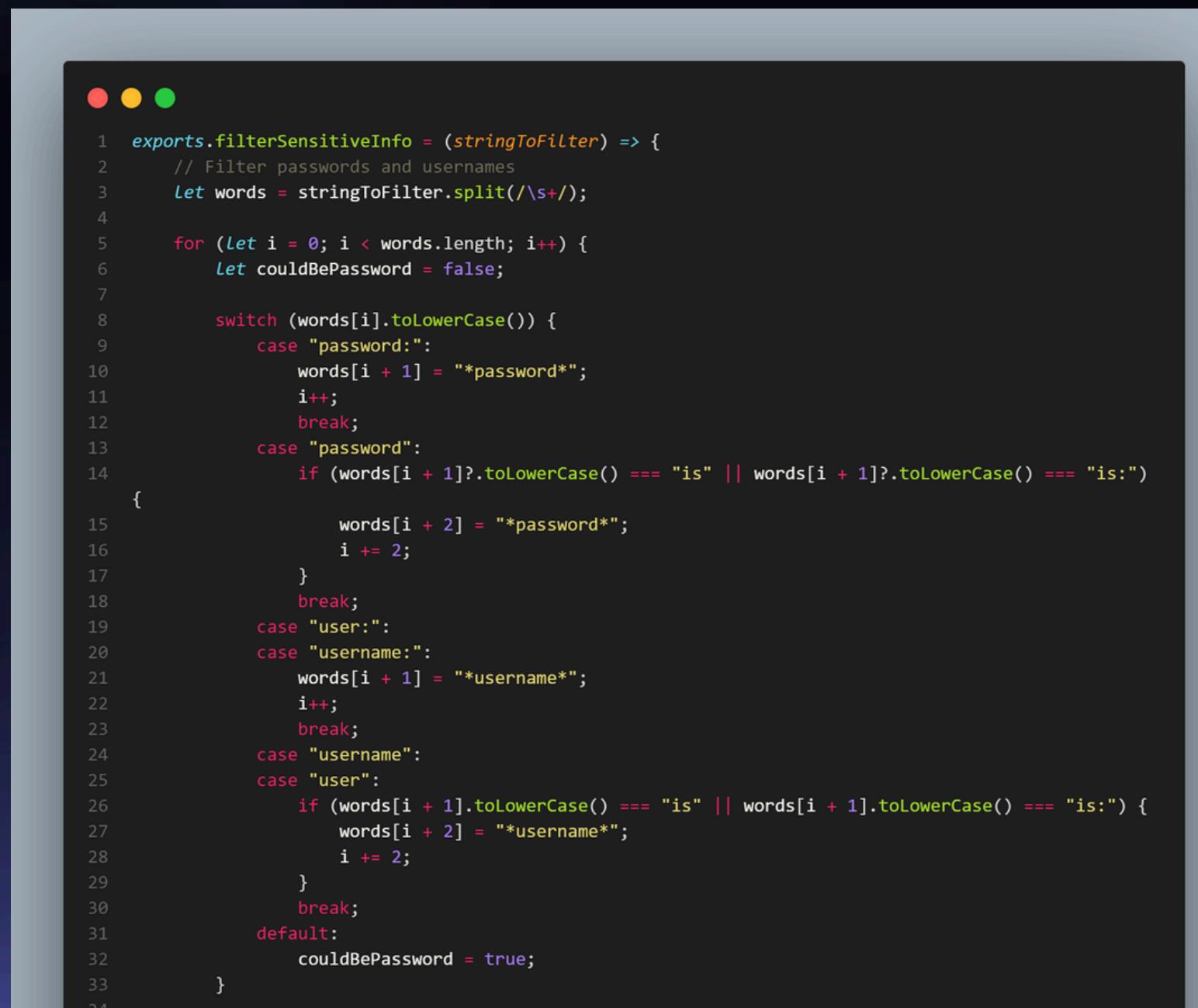
**Deconstructuring the ticket:** taking the ticket data in its raw format and storing it in a new object, filtering the fields that may have sensitive information along the way using a filter function filterSensitiveInfo(), finally storing the ticket with key-value pairs as const filterIssueData for a filtered and deconstructed ticket.

```
1  const filterIssueData = (data) => {
2      // Extracting relevant data from the issue object and filtering
3      // sensitive information
4      const issueKey = data.issue?.key;
5      const issueTitle = filterSensitiveInfo(data.issue?.fields?.summary);
6      const issueDescription = filterSensitiveInfo(data.issue?.fields?.description);
7      const priority = data.issue?.fields?.customfield_10059?.value;
8      const hardOrSoft = data.issue?.fields?.customfield_10062?.value;
9      const category = data.issue?.fields?.customfield_10070?.value;
10     const subCategory = data.issue?.fields?.customfield_10071[0]?.value;
11     const impact = data.issue?.fields?.customfield_10065?.value;
12
13     // Constructing a filtered issue data object
14     const issueData = {
15         issueKey: issueKey,
16         issueTitle: issueTitle,
17         issueDescription: issueDescription,
18         priority: priority,
19         HardwareOrSoftware: hardOrSoft,
20         category: category,
21         subCategory: subCategory,
22         impact: impact,
23     };
24     return issueData;
}
```

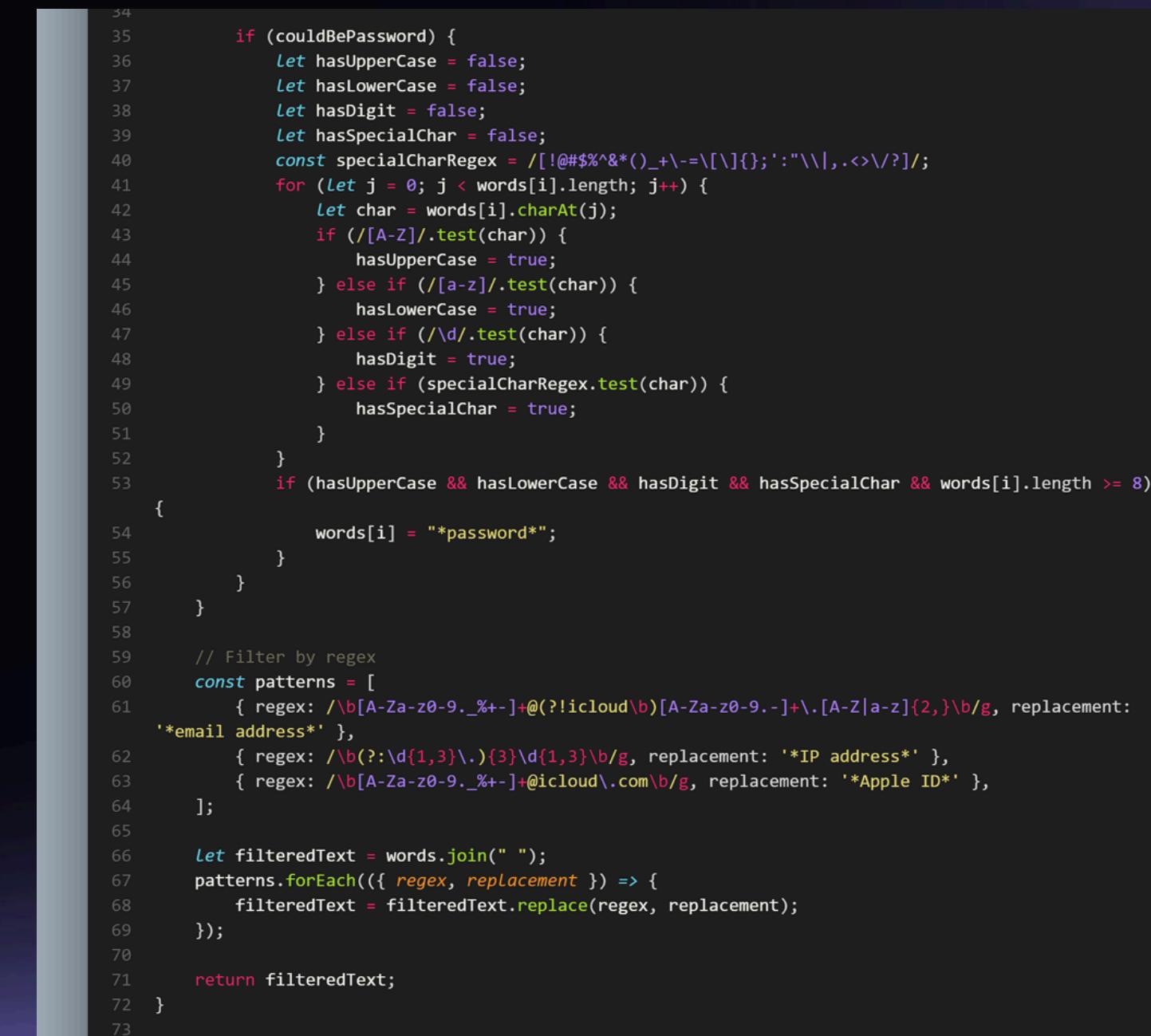
# Nissim & Ariel

## Filtering function:

The function goes over the string that is passed into it, and checks if there are words that may be passwords, usernames, emails, and IP addresses, that we would not want to be passed into the AI Assistant in concern of the safety of our information and data. This is done with a custom function, and regex.



```
1 exports.filterSensitiveInfo = (stringToFilter) => {
2   // Filter passwords and usernames
3   let words = stringToFilter.split(/\s+/);
4
5   for (let i = 0; i < words.length; i++) {
6     let couldBePassword = false;
7
8     switch (words[i].toLowerCase()) {
9       case "password":
10         words[i + 1] = "*password*";
11         i++;
12         break;
13       case "password":
14         if (words[i + 1]?.toLowerCase() === "is" || words[i + 1]?.toLowerCase() === "is:")
15           {
16             words[i + 2] = "*password*";
17             i += 2;
18           }
19         break;
20       case "user:":
21       case "username:":
22         words[i + 1] = "*username*";
23         i++;
24         break;
25       case "username:":
26       case "user:":
27         if (words[i + 1].toLowerCase() === "is" || words[i + 1].toLowerCase() === "is:")
28           {
29             words[i + 2] = "*username*";
30             i += 2;
31           }
32         break;
33       default:
34         couldBePassword = true;
35     }
36   }
37
38   return couldBePassword ? words : stringToFilter;
39 }
```



```
34
35   if (couldBePassword) {
36     let hasUpperCase = false;
37     let hasLowerCase = false;
38     let hasDigit = false;
39     let hasSpecialChar = false;
40
41     const specialCharRegex = /[!@#$%^&*()_+=\[\]\{\};:'"\\"\\,.<>\?]/;
42
43     for (let j = 0; j < words[i].length; j++) {
44       let char = words[i].charAt(j);
45
46       if (/^[A-Z]$/.test(char)) {
47         hasUpperCase = true;
48       } else if (/^[a-z]$/.test(char)) {
49         hasLowerCase = true;
50       } else if (/^\d$/.test(char)) {
51         hasDigit = true;
52       } else if (specialCharRegex.test(char)) {
53         hasSpecialChar = true;
54       }
55
56     }
57
58     if (hasUpperCase && hasLowerCase && hasDigit && hasSpecialChar && words[i].length >= 8)
59     {
60       words[i] = "*password*";
61     }
62
63   }
64
65   // Filter by regex
66   const patterns = [
67     { regex: /\b[A-Za-z0-9._%+-]+@[?!icloud\b][A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b/g, replacement: '*email address*' },
68     { regex: /\b(?:\d{1,3}\.){3}\d{1,3}\b/g, replacement: '*IP address*' },
69     { regex: /\b[A-Za-z0-9._%+-]+@icloud\.com\b/g, replacement: '*Apple ID*' },
70   ];
71
72   let filteredText = words.join(" ");
73   patterns.forEach(({ regex, replacement }) => {
74     filteredText = filteredText.replace(regex, replacement);
75   });
76
77   return filteredText;
78 }
```

# Moshe & Gadi

Defines an asynchronous function named `NewAiChat` that takes an `issueData` object as input and returns a Promise. This function uses the OpenAI API to create a new conversation thread and send a message based on the provided `issueData`.

```
1 exports.NewAiChat = async (issueData) => {
2     return new Promise(async (resolve, reject) => {
3         try {
4             // Check if issueData exists
5             if (issueData) {
6                 // Initialize OpenAI client
7                 const openai = new OpenAI({
8                     apiKey: process.env.OPENAI_API_KEY
9                 });
10
11             // If there's no existing conversation thread, create a new one
12             if (!conversationThreadId) {
13                 console.log("new conversation thread");
14                 const thread = await openai.beta.threads.create();
15                 conversationThreadId = thread.id;
16             }
17         } catch (error) {
18             reject(error);
19         }
20     });
21 }
```

# Moshe & Gadi

The following code executes all the internal AI functions and catches any errors that occur in the promise.

```
● ● ●  
1 // Send initial message to the AI assistant  
2         const message =`  
3             This is the object with the details:  
4             ${JSON.stringify(issueData)}  
5             `;  
6             await addMessage(conversationThreadId, message);  
7  
8             // Start the AI assistant and periodically check its status  
9             const run = await runAssistant(conversationThreadId);  
10            const runId = run.id;  
11  
12            // Periodically check the status of the AI assistant  
13            const pollingInterval = setInterval(() => {  
14                checkingStatus(conversationThreadId, runId);  
15            }, 5000);  
16            }  
17        } catch (error) {  
18            // Reject the promise if any error occurs  
19            reject(error);  
20        }  
21    );  
22 }
```

# Moshe & Gadi

**The following code sends all the details of the incoming ticket into the conversation with the AI, based on the prompt defined on the site, which specifies what response to provide in detail.**

```
● ● ●  
1 // Function to add a message to the conversation thread  
2           async function addMessage(threadId, message) {  
3             const response = await openai.beta.threads.messages.create(  
4               threadId,  
5               {  
6                 role: "user",  
7                 content: message  
8               }  
9             );  
10            return response;  
11          }
```

# Moshe & Gadi

**The following code runs the conversation with the AI using the provided message.**

```
1 // Function to run the AI assistant on the conversation thread
2 async function runAssistant(threadId) {
3   const response = await openai.beta.threads.runs.create(
4     threadId,
5     {
6       assistant_id: process.env.ASSISTANT_ID
7     }
8   );
9   return response;
10 }
```

# Moshe & Gadi

● ● ●

```
1 // Function to periodically check the status of the AI assistant
2 async function checkingStatus(threadId, runId) {
3     const runObject = await openai.beta.threads.runs.retrieve(
4         threadId,
5         runId
6     );
7
8     const status = runObject.status;
9
10    if (status == 'completed') {
11        // Once assistant run is completed, fetch messages from the thread
12        const messagesList = await openai.beta.threads.messages.list(threadId);
13        let messages = [];
14
15        // Extract content from messages
16        messagesList.body.data.forEach(message => {
17            messages.push(message.content);
18        });
19
20        // Resolve the promise with AI response
21        resolve(messages[0][0]?.text?.value);
22    }
23}
```

The following code is executed every 5 seconds by the main code section of the AI, which triggers the AI functions and checks the status of the AI response. It determines whether the AI is still in the midst of generating a response, if an error has occurred, or if the response has been completed. If the response has been completed, it then concludes the Promise with the AI's response.

# Nissim & Ariel

The following code returns the response generated by the AI to the Gerrit system using their API as an administrative response to the specific ticket. If there is an error in the process, it throws a message with the error content.

```
1  exports.addCommentToJira=async(issueKey, comment)=> {
2
3      const url = `${process.env.JIRA_URL}/rest/api/2/issue/${issueKe
y}/comment`;
4      const auth = 'Basic ' + Buffer.from(process.env.USER_NAME + ':'
+ process.env.TOKEN).toString('base64');
5      const requestBody = {
6          body: comment,
7          visibility: { type: 'role', value: 'Administrators' }
8      };
9      const response = await fetch(url, {
10          method: 'POST',
11          headers: {
12              'Authorization': auth,
13              'Content-Type': 'application/json'
14          },
15          body: JSON.stringify(requestBody)
16      });
17
18      if (!response.ok) {
19          throw new Error(`Failed to add comment to Jira: ${response.
statusText}`);
20      }
21
22      return response.json();
23  }
```

# Nissim & Ariel

The following code summarizes the final commands in the main code section responsible for storing all ticket details along with the AI response into a custom-defined model. It then saves this data into a MongoDB database structure specifically created for this purpose and was connected to the code.

```
1  const mongoose = require('mongoose');
2  require("dotenv").config();
3
4  main().catch(err => console.log(err));
5
6  async function main() {
7    await mongoose.connect(process.env.DB_CONNECT);
8    console.log("mongo connect jira");
9
10 }
```

```
1  const mongoose=require('mongoose');
2
3  const issueSchema = new mongoose.Schema({
4    issueKey:String,
5    issueTitle:String,
6    issueDescription:String,
7    priority:String,
8    hardOrSoft:String,
9    category:String,
10   subCategory:String,
11   impact:String,
12   aiFeedback:String,
13 }, { timestamps: true })
14
15 exports.IssueModel = mongoose.model("issues", issueSchema);
```

# Example of Usage

The projector gives an error about my user information

Create subtask    Link issue    Add form   Create ...

 Example Customer raised this request via Portal Hide details  
[View request in portal](#)

Description  
I tried to connect to the department's projector with username: EVI123 and with the password: 123456 and it gave me an error and didn't let me log in.  
Would appreciate help...  
Thanks!

Priority   Medium

Hardware Or Software?   Hardware

Category   Hardware - Peripherals

Subcategory   Other...

Impact   The whole department

I want pizza

Create subtask    Link issue    Add form   Create ...

 Moshe Evyatar Gidoni raised this request via Portal Hide details  
[View request in portal](#)

Description  
last night i dreeming about very big pizza with a lot of cheese and i want you to give me that

Priority   Critical

Hardware Or Software?   Hardware

Category   Network

Subcategory   Hardware-Peripherals-Keyb...

Impact   The Whole Company

# Returning to Jira

We've set up a system to add AI feedback as comments to Jira tickets, aiding administrators in quicker issue resolution. Additionally, it helps in reclassifying ticket categories and prioritizing requests, enhancing organizational efficiency.

 gadi 1 hour ago • Administrators

\*Issue Key:  IB-236: The projector gives an error about my user information TO DO

\*Issue Title: The projector gives an error about my user information

\*Suggested categories from the existing list\*

- Category: Hardware - Peripherals
- Subcategory: More.. - Audio / visual equipment

\*Priority and Impact Analysis\*

- Priority matches the content, as it affects departmental operations but is not immediately critical.
- Impact matches the content, assuming the projector's functionality is crucial for the whole department's activities.

\*Possible Solutions (Next Steps)\*

1. Verify the user credentials and ensure they have the correct permissions to access the projector.
2. Inspect the projector's connectivity and configuration settings for any misalignments or errors.
3. Arrange for a technical review or on-site support if the issue cannot be resolved remotely.

Edit · Delete · 

 gadi 44 minutes ago • Administrators

\*Issue Key:  IB-232: I want pizza TO DO

\*Issue Title: I want pizza

\*Suggested categories for creation\*

- Non-technical requests

\*Priority and Impact Analysis\*

- Priority does not match the content. Suggested priority: Low
- Impact does not match the content. Suggested impact: Only me

\*Possible Solutions (Next Steps)\*

1. Politely inform the customer this request is outside the scope of IT support.
2. Direct them to the correct department if applicable, or advise on proper channels for personal requests.
3. Update ticketing guidelines to clarify types of supported requests.

Edit · Delete · 

```
"_id": "663c8ee2af9781ae74737863",
"issueKey": "IB-232",
"issueTitle": "I want pizza",
"issueDescription": "last night i dreeming about very big pizza with a lot of cheese and i want you to give me that",
"priority": "Critical",
"category": "Network",
"subCategory": "Hardware-Peripherals-Keyboard",
"impact": "The Whole Company",
"aiFeedback": "***Issue Key:** IB-232\n**Issue Title:** I want pizza\n**Suggested categories for creation**\n- Non-  
- Non-technical requests\n- Priority and Impact Analysis\n- Priority does not match  
the content. Suggested priority: Low\n- Impact does not match the content. Suggested impact: Only me\n- Possible  
Solutions (Next Steps)\n1. Politely inform the customer this request is outside the  
scope of IT support.\n2. Direct them to the correct department if applicable, or advise on proper channels for  
personal requests.\n3. Update ticketing guidelines to clarify types of supported requests.",
"createdAt": "2024-05-09T08:52:50.567Z",
"updatedAt": "2024-05-09T08:52:50.567Z",
"_v": 0
```

```
"_id": "663c95c9af9781ae7473786b",
"issueKey": "IB-236",
"issueTitle": "The projector gives an error about my user information",
"issueDescription": "I tried to connect to the department's projector with username: *username* and with the password:  
*password* and it gave me an error and didn't let me log in. Would appreciate help... Thanks! ",
"priority": "Medium",
"category": "Hardware - Peripherals",
"subCategory": "Oter...",
"impact": "The whole department",
"aiFeedback": "***Issue Key:** IB-236\n**Issue Title:** The projector gives an error about my user  
information\n**Suggested categories from the existing list**\n- Category: Hardware - Peripherals\n- Subcategory: More.. - Audio / visual equipment\n- Priority and Impact Analysis\n- Priority matches  
the content, as it affects departmental operations but is not immediately critical.\n- Impact  
matches the content, assuming the projector's functionality is crucial for the whole department's activities.  
\n- Possible Solutions (Next Steps)\n1. Verify the user credentials and ensure they have  
the correct permissions to access the projector.\n2. Inspect the projector's connectivity and configuration  
settings for any misalignments or errors.\n3. Arrange for a technical review or on-site  
support if the issue cannot be resolved remotely.",
"createdAt": "2024-05-09T09:22:17.740Z",
"updatedAt": "2024-05-09T09:22:17.740Z",
"_v": 0
```



MongoDB®

# Product Value

1

## AI-Driven Category Suggestions

Implementing AI to suggest new categories based on the frequency of similar issues submitted enhances the categorization process.

2

## Reassessment of Ticket Priority and Impact

Regularly reassessing the priority and impact level of tickets to improve the efficiency and effectiveness of the IT department

3

## Provision of Quick Resolutions

Offering quick and common solutions for recurring issues, thereby resolving tickets more efficiently.

4

## Development of a Comprehensive Database:

Building a detailed database of ticket information and AI-generated comments to facilitate deeper analysis and continuous improvement.

# What I learned

Ariel Harris

I learned all about Jira Service management, how to manage it and how the system works. What the difference between that to Project management. How to integrate the platform with an API, and how HTTP requests work and are managed.

Furthermore, I learned how to build an API server that handles requests, deconstructs complex data, stores it in databases and sends HTTP requests. I also learned how to manage a database.

Lastly, I learned about prompt engineering basics, how to use OpenAI API, different types of AI APIs, and specifically how to set a role for the AI assistant. And, of course, how to integrate it into the project.



# What I learned

Nissim Cohen

In our work on this program, I learned to craft effective AI prompts and fine-tune them for optimal results. I also mastered Jira Service Management, building a project resembling real-world scenarios. This included creating custom forms, fields, and webhooks for seamless integration. Additionally, I developed a solution to filter sensitive data using regex and custom functions, ensuring data privacy. Finally, I integrated MongoDB to store ticket data alongside OpenAI responses, providing a comprehensive solution for project management and analysis.



# What I learned

Moshe Evyatar Gidoni

In our work, I gained hands-on experience with NGROK, which proved invaluable for testing applications locally. Its ability to create secure tunnels to local servers facilitated efficient debugging and testing processes, ultimately streamlining our development workflow. Moreover, delving into OPEN AI was a fascinating journey. From exploring its diverse language models to delving into its capabilities in natural language processing, I was truly amazed by its potential to advance AI solutions across various applications.



# What I learned

Gadi Kugel

Throughout the work in the company, I deepened my understanding of several key technologies essential for modern web development. NODE JS emerged as a powerful tool for server-side scripting, offering asynchronous, event-driven capabilities that are ideal for building scalable applications. Working with EXPRESS, I appreciated its minimalist framework, which accelerated the development of robust web applications. Additionally, exploring WEB HOOKS provided insights into their role in enabling real-time communication between web services like Jira Service Management.



# Thank you!

Presented by:

Gadi kogel, Ariel Harris, Nisim Cohen and Moshe Evyatar Gidoni

• • •

## Thanks

---

Ort Singalovski College

AppsFlyer

AppsFlyer's mentors:

Mark Samuelson & Nir Peled

IDF - Minhelet Charedim

DiversiTech - Avigail Michelson