



**SOFTWARE
ARCHITECTURE**

2025-26

Jose Emilio Labra Gayo
Pablo González
Diego Martín
Celia Melendi

Lab 1



Escuela de
Ingeniería
Informática



Universidad de Oviedo

Using AI in Software
Development Projects

How AI affects software development

The way we program is changing

More and more powerful tools are emerging to complete code, explain code, generate documentation, refactor, etc.

How does the role of the analyst/programmer change in a project?

We went from writing code line by line to designing and monitoring the code generated

- It changes the role of the developer to a higher level one.

Problem: it presupposes a high level of technical knowledge to ensure the quality of the code generated.

What can an AI assistant do?

Some of the functions to perform with AI are the following:

- Complete code
- Explain parts of code already written
- Generate documentation or tests
- Troubleshooting and explaining errors
- Perform complex tasks in agent mode
 - Create multiple code files
 - Create configuration files
 - Run routines (such as installing required dependencies)

Level 1: Smart Autocomplete

With smart autocomplete we can:

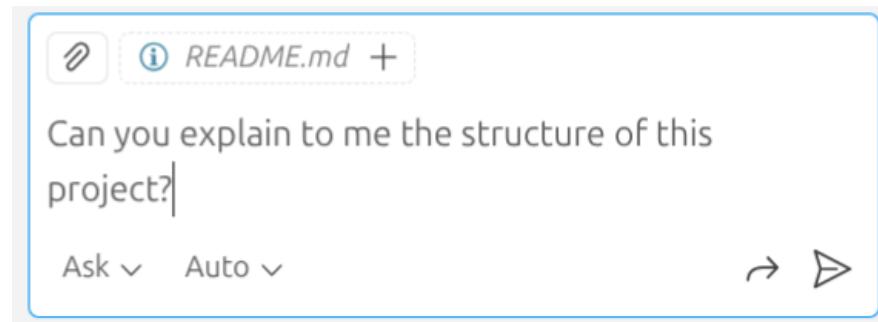
- Avoid writing repetitive code
- Very useful for writing pieces of code that can be easily explained with a comment
- In tools like VSCode, we will write the comment and the AI will start generating code that we can accept by pressing TAB

```
// Create a new method for authenticating users
app.post('/authenticate', async (req, res) => {
  const { username, password } = req.body;
  try {
    // Simulate a 1 second delay to mimic processing/network latency
    await new Promise((resolve) => setTimeout(resolve, 1000));
  }
});
```

Level 2: Code Explanation

With the code explanation we can:

- Ask technical questions about an entire project.
- Understand parts of code not written by us.
- In general, ask questions that do not involve a direct modification of our project's code.



Components

Webapp ([webapp](#))

A React single-page application built with Vite and TypeScript:

- **Main purpose:** User registration interface
- **Key files:**
 - [src/App.tsx](#) - Main app component
 - [src/RegisterForm.tsx](#) - Registration form
 - [vite.config.ts](#) - Build configuration
- **Testing:** Vitest unit tests + Playwright E2E tests with Cucumber
- **Deployment:** Dockerized Nginx server

Users Service ([users](#))

A REST API backend built with Node.js and Express:

- **Main purpose:** Handles user creation and returns welcome messages
- **Key endpoint:** [POST /createuser](#) - Creates a user and returns a greeting
- **Key files:**
 - [users-service.js](#) - Main API server
- **Monitoring:** Prometheus metrics + Grafana dashboard integration
- **Testing:** Vitest with Supertest for API testing

Level 3: Code Edit

With code modification we can:

- Make complex modifications to multiple files.
- We have the ability to accept or reject the proposed changes.
- We can refine the generated code by iterating with the system.
- We can access the reasoning chain to understand how the code has been generated.

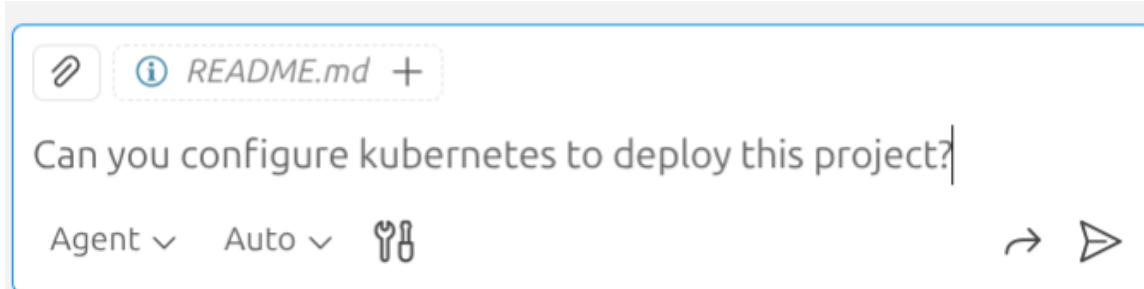
```
① README.md +  
add a new call to the users service to authenticate a user  
Edit Auto ➔
```

```
① README.md ② JS users-service.js ③ X  
users > JS users-service.js > app.post('/authenticate') callback  
  ↵ app.post('/authenticate', async (req, res) =>  
  ↵   40   );  
  ↵  
  ↵   41  
  ↵   42 app.post('/authenticate', (req, res) => {  
  ↵   43     const { username, password } = req.body;  
  ↵   44  
  ↵   45     if (!username || !password) {  
  ↵   46       return res.status(400).json({ error: 'Username and password' }  
  ↵   47     }  
  ↵  
  ↵   48  
  ↵   49     // Simulate authentication (replace with real logic)  
  ↵   50     if (password.length >= 6) {  
  ↵   51       res.json({  
  ↵   52         success: true,  
  ↵   53         message: `User ${username} authenticated successfully`,  
  ↵   54         token: `token_${username}_${Date.now()}`  
  ↵   55       });  
  ↵   56     } else {  
  ↵   57       res.status(401).json({ success: false, error: 'Invalid crede  
  ↵   58     }  
  ↵  
  ↵   59   });  
  ↵  
  ↵   60  
  ↵   61  
  ↵   62     if (require.main === module) {  
  ↵   63       app.listen(port, () => {  
  ↵   64         console.log(`User Service listening at http://localhost:3001`)  
  ↵   65       })  
  ↵
```

Level 4: Agent Mode

With agent mode we can:

- It performs the same functions as in the previous mode, but can also **invoke terminal commands** and other tools.
- Useful for complex tasks involving multiple steps.
- It can take several minutes for agents to perform assigned tasks.



README.md +

Can you configure kubernetes to deploy this project?

Agent ▾ Auto ▾ 🔧

→ ➔



Run bash command?

```
chmod +x /home/pablo/Proyectos/asw2526_0/k8s/*.sh
```

Allow | Skip

Perfect! Now let me create one final comprehensive README for the k8s directory:

✓ Created README.md

Perfect! Let me verify the structure and create a summary:

✓ Read k8s

Excellent! Let me create one final file with quick reference commands:

✓ Created COMMANDS.md

Perfect! Now let me create a summary document:

✓ Created KUBERNETES.md

Perfect! Let me verify all files were created properly:

```
find k8s -type f -name "*.yaml" -o -name "*.md" -o -name "*.sh" | sort
```

✓ k8s

- \$ cleanup.sh
- ⬇ COMMANDS.md
- \$ deploy.sh
- ⬇ DEPLOYMENT.md
- ! gamey-deploy...
- ! gamey-service.y...
- ! grafana-deploy...
- ! grafana-service....
- ! ingress.yaml
- ! kind-config.yaml
- ! kustomization.y...
- ⬇ LOCAL_DEVELO...
- ! namespace.yaml
- \$ port-forward.sh
- ! prometheus-co...
- ! prometheus-de...
- ! prometheus-ser...
- ⓘ README.md
- ! users-deploy...
- ! users-service.yaml
- ! webapp-deploy...
- ! webapp-service....



Risks and best practices

Quality of the generated code

- We have no guarantee that the generated code is correct.
- We must be able to understand and supervise the generated code to guarantee the quality of the software generated.

Dependency

- Using code generators limits our current ability to write code.
- This effect is amplified in the learning/training phase of a programmer.

Basic rules

- If you don't understand it, don't use it.
- Use it to learn by asking, get explanations, and get better as a programmer.
- Use it for repetitive tasks that you already know how to perform.
- Add tests (code without tests = dangerous code)

Links with more info

- Code review with AI
 - <https://addyosmani.com/blog/code-review-ai/>
- Your job is to deliver code that Works:
 - <https://simonwillison.net/2025/Dec/18/code-proven-to-work/>