VOL. 4
FEBRUARY 2026

IEEE COMPUTER SOCIETY
SSN Student Branch Chapter

git informed

UPDATES YOU'LL ACTUALLY WANT TO PULL

What's inside this issue?

Section 1

The Curious Case of the QR Code

-By Shivani Garimella

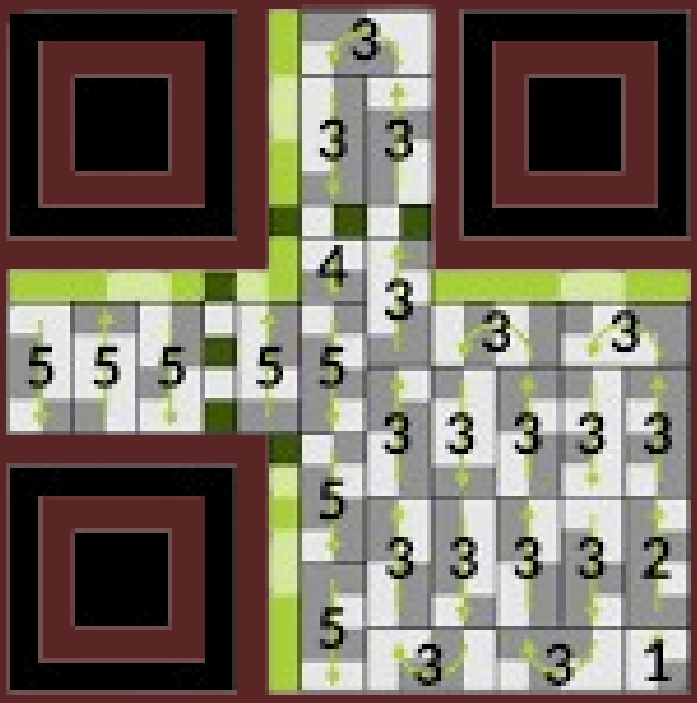
The humble QR. From little neighbourhood corner shops to Anirudh concert ticket bookings, even for attendance at our own IEEECS events (trust us, we need that OD as much as you do), not a day goes by where we don't feel its presence, and for good reason. It's fast and easy to use (hence the name QR – Quick Response), cheap to mass produce or print, and honestly, the fact that it's a cute little square you can stick in any corner helps.

One sunny day, on a much needed ice-cream run, I pulled out my phone to pay by UPI. I scanned the QR code, and it occurred to me that I never stopped to question how the QR code knows that I want to make a payment to a given account. So I ended up down a rabbit hole, and am here to clue you in on what I found!

The QR code was not the first attempt of encoding of its kind. One closely related technology that comes up is the bar code. Bar codes are linear, and hence are more limited in the amount of information they can hold. This results in very localised, small scale applications à la storing product codes at supermarkets. The QR code is a 2-dimensional grid, which allows for more storage, and hence, a wider field of application.

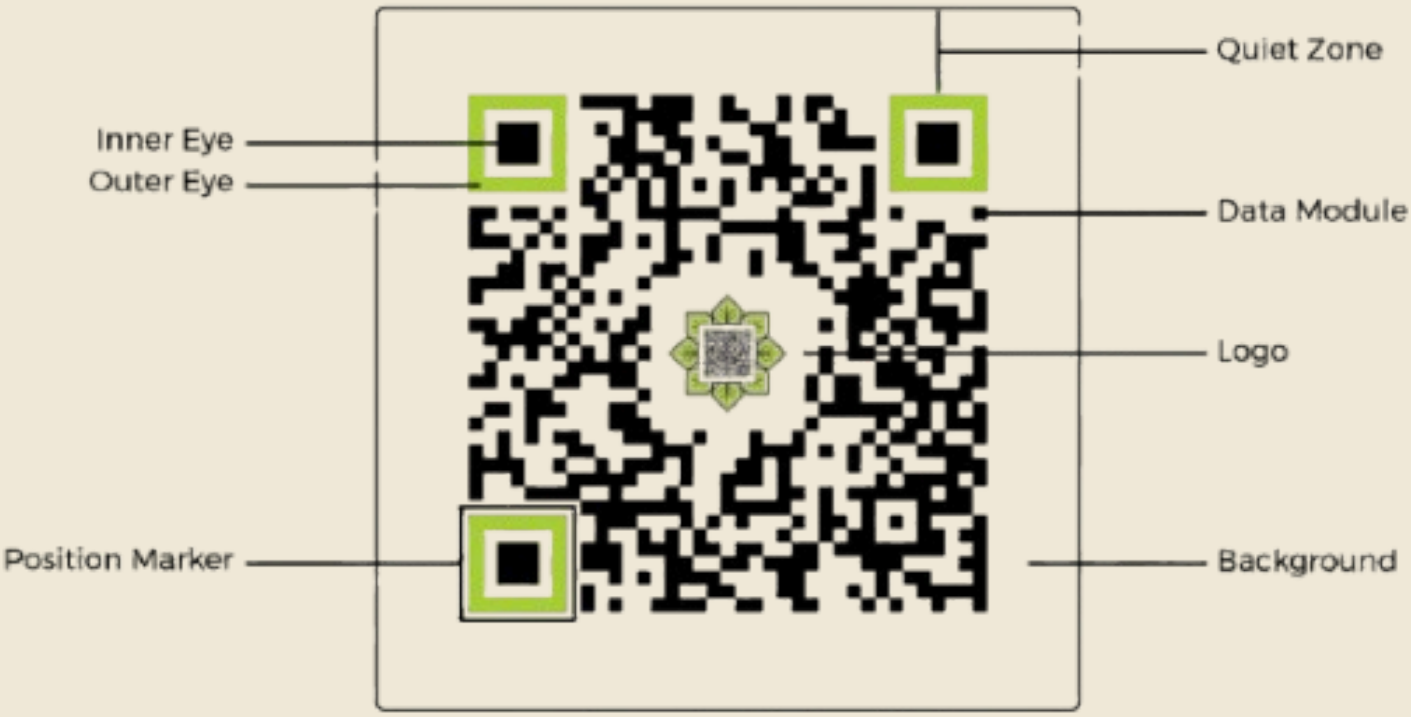
The QR code was created by Masahiro Hara in 1994. He invented it to track automobile parts during manufacturing, inspired by the grid-like pattern of the game Go.

QR works by encoding binary data into white and black squares. Each of these squares is called a module, and can represent a binary 0 or 1, depending on its color. This binary data can range from contact cards to café menus to payment links. QR codes have position markers, which indicate the orientation of the QR and allow for scanning regardless of position. They have other features like alignment markers, timing strips to communicate the QR version being used, etc. Depending on the version, one can expect different amounts of information from it. Version 1 was a 21 x 21 module square that holds up to 17 bytes of data. The latest version is Version 40, a 117 x 117 module square that can hold around 3 kB of information.



Another distinguishing feature of QR codes is their robustness. This is achieved through error correction. There are four levels of error corrections possible in QRs, which entail adding redundant bits. As with any information encoding technique, the higher levels allow for more robustness at the cost of data storage space. Some QR codes can withstand up to 30% distortion while still being scannable!

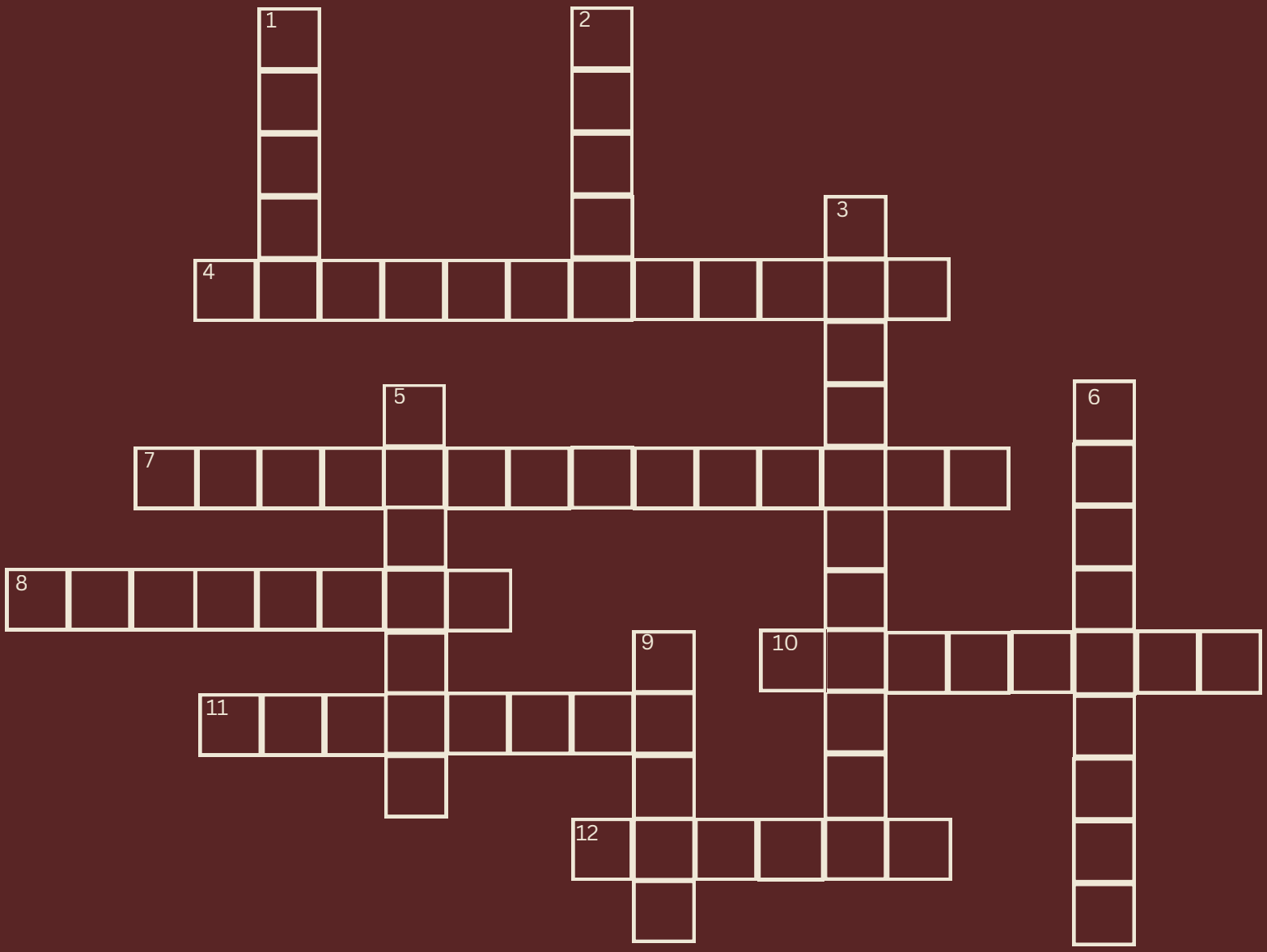
There are two types of QR codes: dynamic and static. As the names suggest, dynamic QR codes allow for the encoded content to be changed, whilst static codes don't. Dynamic QR codes usually contain a redirect / index, which in turn points to said source. Since the encoded data and the actual contents of the QR are decoupled, the contents can be changed. These are the QRs that we usually use for displaying links, quickly sharing text, etc. Static QR codes, on the other hand, are directly encoded with the link, and its contents cannot be changed.



What started as a simple shopping trip ended up as hours of research and an article in a magazine. The world we live in is no less than magical, if you know where to look for it. Ask about the smallest things around you, and let the answers surprise you. Until the next time one of us in the Editorial team gets curious and decides to stay up all night about it, this is Shivani, signing off.

Section 2

Crossword



Across

4. Fundamental structure of a computer system's components.

7. Integrated circuit that contains the core of a computer.

8. Programs and applications that run on a computer.

10. Formal documents defining hardware and protocol rules.

11. Computation type where multiple processes run simultaneously.

12. Pioneer recognized as the father of computer science.

Down

1. Prefix describing the security field protecting networks.

2. Iterative and incremental approach to development.

3. A fundamental unit of work in a database system.

5. Peer-reviewed academic publication of the society.

6. Step-by-step procedure for solving a problem.

9. Online platform where data and services are stored and accessed via the internet.

Look forward to the answers in the next edition!

Section 3

When Your Fridge Knows More About You Than Your Friends

-By Dona Merlin Rony

We live in a world where refrigerators have Wi-Fi, door locks have Bluetooth, and light bulbs nag you for firmware updates. This is the age of the Internet of Things, where once-simple appliances have quietly evolved into computers with sensors, processors, and network interfaces. The humble fridge has gone from cooling leftovers to running lightweight embedded operating systems, complete with AI integrations and cloud connectivity. It monitors temperature with sensors, tracks groceries with RFID, uses weight detection to know when you've finished the milk, and then happily uploads all this data for safekeeping. Your midnight snack is no longer just between you and your conscience, it's logged, timestamped, and potentially monetized.

At its core, IoT is about billions of devices working as distributed systems, continuously sensing, transmitting, and adapting. The pitch is convenience: homes that anticipate your habits, appliances that reorder groceries before you notice they're gone, predictive maintenance that saves time and money. But with every connected device comes an unblinking eye. A fridge suggesting salad over ice cream is funny until you realize it's profiling your diet and feeding it into a cloud service refining its ad algorithms. Underneath the glossy promise, IoT is a showcase of computer science's toughest problems: security, scalability, and interoperability.

Security remains the weakest link. Many devices still ship with hardcoded passwords, making them easy prey for attackers. Entire botnets, like Mirai, have already exploited vulnerable cameras and routers to launch massive, distributed denial-of-service attacks. Then there's scalability: billions of devices pushing streams of data need lean communication protocols such as MQTT and resource-efficient encryption to keep latency and bandwidth under control. Interoperability might be the messiest of all. Manufacturers often insist on their own closed ecosystems, leaving you with a "smart" home that functions more like a digital Tower of Babel, where your thermostat and vacuum can't agree on who controls the Wi-Fi.

The rise of autonomy adds another twist. IoT devices are no longer just passive sensors; machine learning models increasingly allow them to recommend or enforce actions. A fridge could, in theory, deny dessert until your smartwatch verifies you've burned enough calories. That might sound dystopian, but we already have thermostats that "decide" energy-efficient ranges and voice assistants that quietly filter what content you can access. When appliances evolve into decision-makers, the line between convenience and control begins to blur.

Zoom out, and the same principles drive entire smart cities. Traffic sensors, connected cars, energy grids, and surveillance networks rely on IoT backbones to generate vast, real-time datasets. Handling these flows requires distributed algorithms, fault tolerance, and privacy-preserving analytics at scales that rival supercomputers. Yet, every new device also introduces a new attack surface. The challenge is no longer just building IoT systems that function; it is ensuring they can scale securely, interoperate globally, and protect user data in a world where failure can cascade from one device to millions.

In the end, IoT is less about things and more about data, who collects it, who owns it, and who profits from it. Every step your watch tracks, every sensor reading your fridge emits, every command your smart speaker records add to a digital ecosystem where the intelligence may belong to corporations or advertisers more than to you. Until we resolve those questions of ownership and trust, the running joke will stay uncomfortably true: your fridge isn't just your roommate, it might also be your most dedicated spy.

FROM THE Alumni

Nithin Balaji
Software Engineer at
Barclays

"The first step is simply building the habit of working on projects, regardless of how big or small the idea is."

1. Back in college, what pulled you towards web development, and when did you start exploring backend development?

I found the idea of Progressive Web Apps (PWA) fascinating. I was instantly captivated by the potential of it when I first learned about how a single codebase could seamlessly operate across different platforms, browsers, and devices. The simplicity of the setup required for web development, compared to app development, was another appealing factor. That made me choose web development. It gave me the ability to build and ship whatever small ideas I brainstormed, and I fell into the cycle of learning and experimenting for my personal projects. Initially, the projects I worked on were simple frontend ones, but after I started participating in hackathons, the need for backend arose, and that is how I started exploring backend development.

2. From your experience, how should someone start learning web development from scratch?

I would recommend starting with HTML and CSS and spending about 1–2 weeks understanding their basics. Once you're comfortable with basic CSS, take another week to explore simple DOM manipulations using JavaScript. Many basic websites and interactions can be built using just HTML, CSS, and JavaScript.
Once you have a solid grasp of these three, usually after about a month, experiment with a CSS framework like Tailwind or Bootstrap. In parallel, if you're proficient in Python, you can start backend development with Flask and later move on to Django for more robust applications, dedicating around 3–4 weeks to each. If you prefer JavaScript, Express is a good option for backend development.
After understanding the fundamentals of both frontend and backend, dedicate a month or more to learning a frontend framework like React. For backend, I'd recommend Spring Boot. You can also refer to roadmap.sh for a detailed learning path.
That said, instead of spending too much time mastering frameworks, it's far more important to build foundations on core concepts like design patterns, API architectural styles, CSR vs SSR, database optimization, etc. These concepts are transferable across frameworks and technologies.

3. With so many tutorials available online, which resources genuinely helped you grow, and are there any underrated websites, GitHub repos, or channels students often overlook?

FreeCodeCamp tutorials really helped me a lot. I learnt web dev basics and React from them. YouTube channels like Fireship and Web Dev Simplified help me rekindle my passion to explore development whenever I lose it. One of the most underrated resources beginners tend to overlook is official developer documentation.

Learning how to read documentation is one of the most important skills a developer can develop. It helps build a strong foundation and keeps you updated with the latest changes. Once you understand the basics of any framework, I strongly recommend referring to documentation instead of relying solely on tutorials.

4. Many students struggle to even decide what project to work on. We often see the same generic, AI-generated project titles everywhere. From your experience, how can students actually come up with meaningful project ideas on their own?

There's absolutely nothing wrong with starting with a generic project. Once you complete it by following a tutorial, go the extra mile by adding your own touch—introduce an additional feature, tweak the UI, or implement the logic differently from what you learned. This process helps you think independently and experiment creatively.
After working on a few such projects and gaining confidence, new ideas will start coming naturally. If the goal is to build projects for your resume, taking inspiration from tutorial-based projects is actually helpful, since they're usually structured to cover most aspects of a framework. For hobby projects, meaningful ideas are all around you. Identifying a problem to solve is a skill in itself and takes time to develop. The first step is simply building the habit of working on projects, regardless of how big or small the idea is.

5. What kind of projects actually helped you stand out during placements?

One of my standout projects was a Discord bot (now inactive) that had around 10,000 users. Having real-time traffic and experience handling it seemed to impress many interviewers. Another interesting project was a Python package published on PyPI, which made it installable through pip.

That was a nice gimmick to showcase to my interviewers if they had Pip in their machine. I also had an interest-based people matching website in my resume to showcase my full-stack skills. At that time, my GitHub was consistently active due to numerous small side projects. A strong GitHub profile is definitely a bonus during off-campus job searches. Additionally, based on the job description, I often switched the list of projects in my resume to match the skills required for the job. All of these helped me secure off-campus internships and gave me an edge in post-DSA rounds during on-campus interviews.

6. Is there any skill you underestimated in college that later became crucial in your job?

Code collaboration.
I had some exposure to it during the development of college fest websites, hackathons, and similar activities, but most of the other times, it was just VS Code and me. I enjoyed the creative freedom of working independently and wasn't particularly inclined towards teamwork.
But after I started working professionally, I realized how essential collaboration truly is. Some of the greatest and most useful software on this planet was built not by a single person but by the collaboration of developers from different countries and backgrounds working towards a single goal. In my current role, I work with multiple developers remotely on a single codebase under time constraints. Clear communication, adaptability, and effective collaboration are essential not just for writing code but also for aligning on requirements, reviewing changes, resolving conflicts, and shipping features on time. Collaboration has directly improved development speed, debugging efficiency, and overall code quality in my work.

Editorial

Dona Merlin Rony

Sanjana Y

Shivani Garimella

Sowkya Bellana

Akshayalakshmi

Design

Shriya Alladi