



SoftWare Engineering Project

Humbat Jamalov

Rauf Jabarov

MirSaid Hesenzade

Huseynova Fakhriyya



The Problem: Scheduling Challenges

Scheduling a meeting often turns into a long back-and-forth. It's time-consuming, inefficient, and frustrating for groups. Most teams struggle to quickly find a time that works for everyone.



Solution Overview

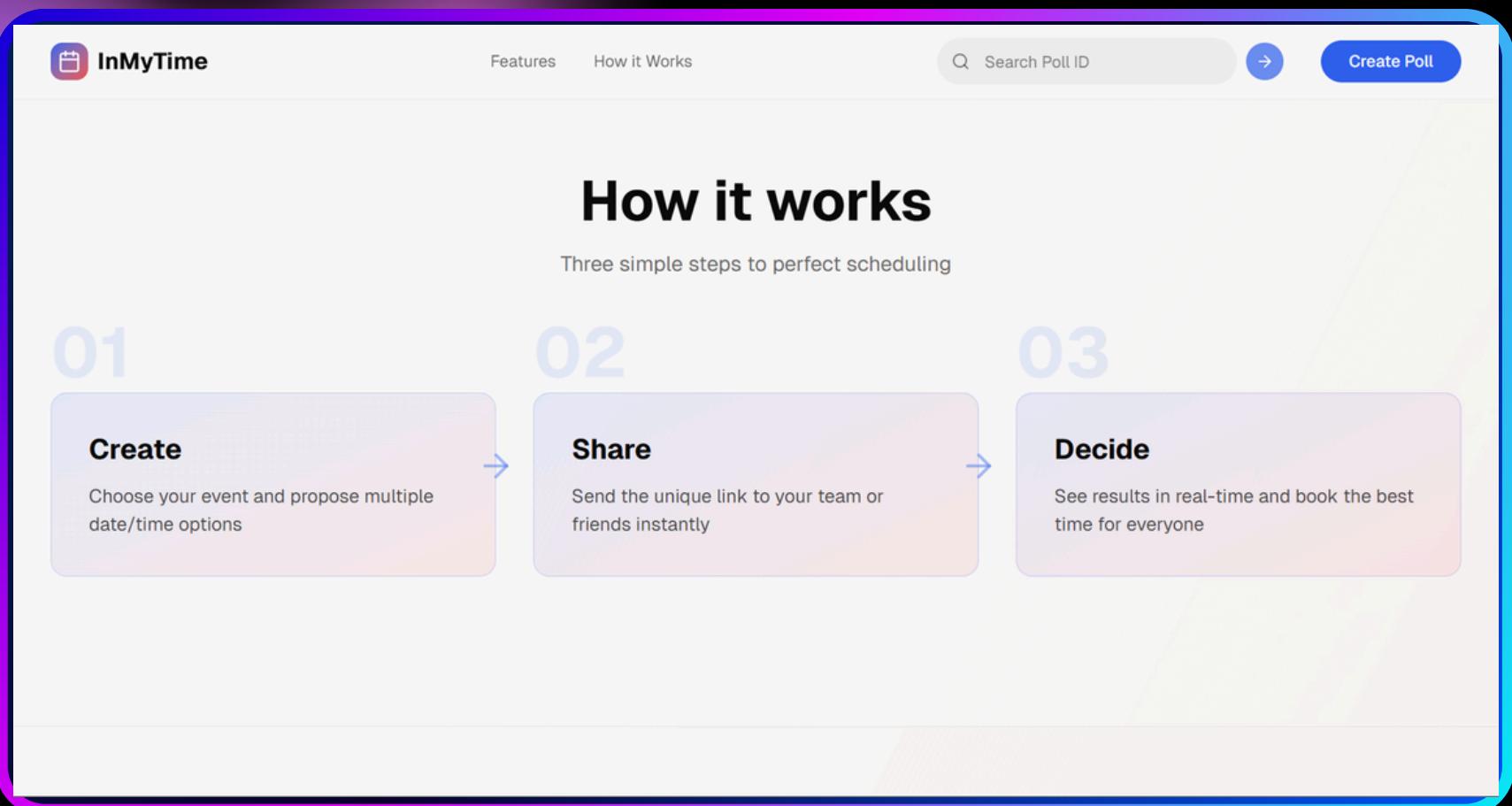
Our solution is a web-based platform that helps groups quickly find a common available time. It collects everyone's availability, visualizes the results, and automatically highlights the best meeting options. This makes scheduling fast, transparent, and effortless.



OUR COMPETITORS

	InMyTime	When2Meet	Doodle
Real-time availability updates			
No Registration Required			
Host-controlled final selection			

Overview of Our Platform

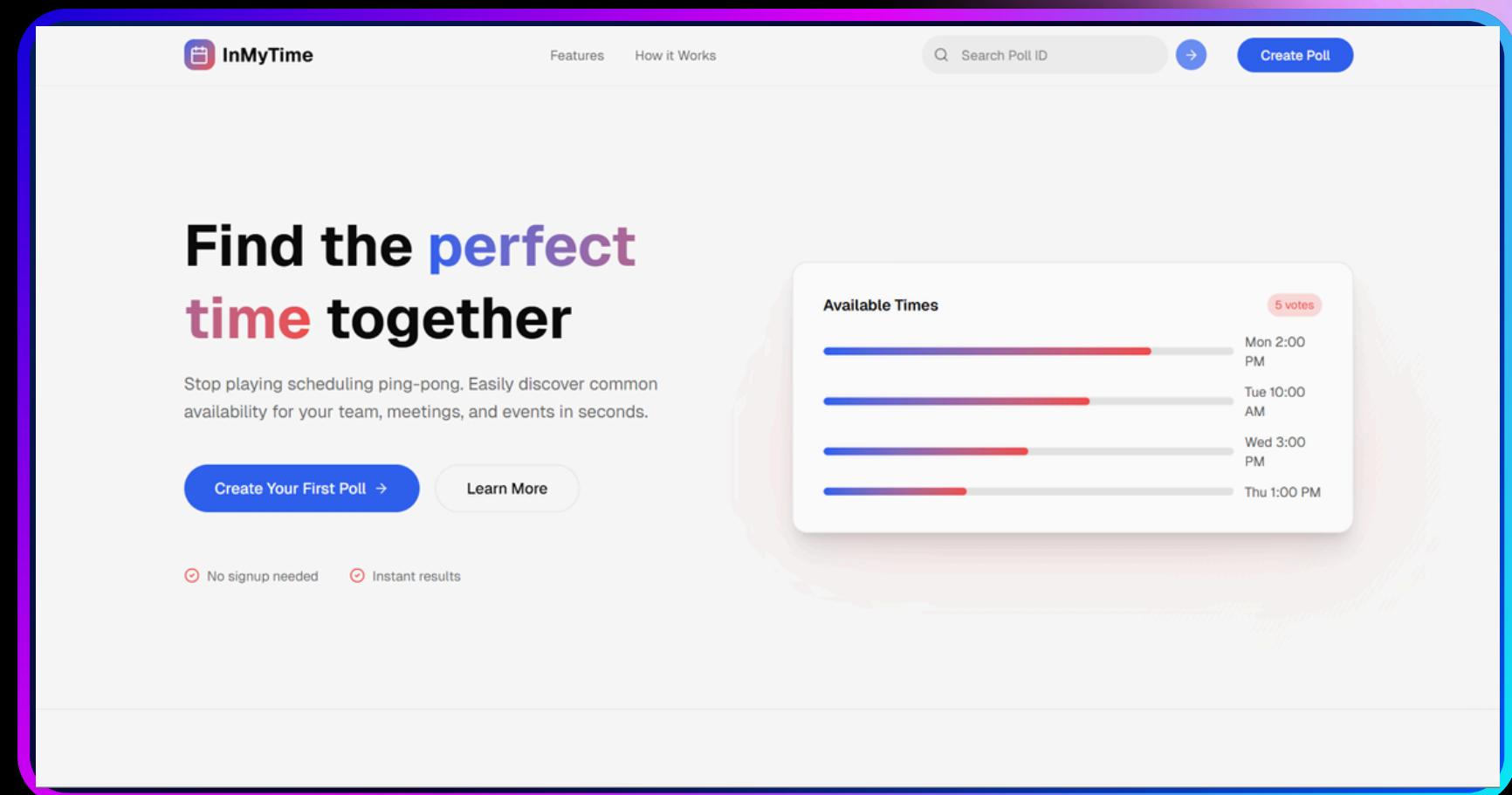


How it works

Three simple steps to perfect scheduling

- 01 Create**
Choose your event and propose multiple date/time options
- 02 Share**
Send the unique link to your team or friends instantly
- 03 Decide**
See results in real-time and book the best time for everyone

Users create a poll, share it with participants, and instantly see the combined availability. The process is simple and intuitive, making scheduling smooth and fast.



Find the perfect time together

Stop playing scheduling ping-pong. Easily discover common availability for your team, meetings, and events in seconds.

[Create Your First Poll →](#) [Learn More](#)

Available Times

Day	Time	Votes
Mon	2:00 PM	5 votes
Tue	10:00 AM	
Wed	3:00 PM	
Thu	1:00 PM	

No signup needed Instant results

The landing page highlights the core value of the platform: finding the perfect meeting time effortlessly. It clearly showcases the main features



Create Poll

The screenshot shows the 'CREATE POLL' section of the InMyTime app. On the left, there's a form with fields for 'Event title' (Product sync, sprint review, study session...), 'Description' (Add an optional note...), and 'Target dates'. Below these are buttons for 'SINGLE DAY' (dd.mm.yyyy) and 'DATE RANGE' (31.12.2025 - 30.01.2026). A '38 selected' badge is visible. To the right is a 'LIVE PREVIEW' titled 'Available Times Overview' showing a list of dates from Mon, Dec 15 to Thu, Jan 8, each with a color-coded bar indicating availability.

Users select dates, set time ranges, and choose slot duration. A live preview helps them understand how the final schedule will look before sharing.

Participants choose their available slots on an interactive grid, with real-time updates, color-coded availability, and clear statistics to guide scheduling decisions.

The screenshot shows the 'VOTING PAGE' for a poll titled 'salam'. It displays a grid of 28 time slots (4 days by 7 hours) with a legend: light green for 'Smal' (1 slot), medium green for 'Popular' (2 slots), and dark green for 'Your Choice' (3 or more slots). The grid shows various levels of participation across different time intervals. To the right, there are sections for 'POLL DETAILS' (how it works, 15m slots), 'Stats Overview' (total votes 79, best day Friday, Nov 7), and 'Participants' (9 participants).

Voting Page



Finalize Feature

After everyone submits their availability, the host selects the best meeting time and closes the poll. A clear confirmation is shown to all participants, making the final step simple and transparent.

The screenshot shows the 'Group Meet' screen in the InMyTime app. At the top, there's a banner with a trophy icon and the text 'Final Meeting Time Confirmed!' followed by 'This poll is officially closed.' To the right, it shows the time '11:00 AM' and the date 'Sunday, November 23'. Below the banner, it displays 'Slot Duration: 30 mins' and 'Total Participants: 13'. The main area is titled 'Participant Details' and shows a list of participants with their initials and names. Each participant has a small profile picture, their name, and a status indicator showing they have 6 documents. To the right of each participant, there's a column of three buttons labeled '0 slots', '0 slots', and '1 slots', with the last one being highlighted in blue. The total count '13' is also shown at the top right of this section.

Participant	Status
salam	0 slots
humbat	0 slots
Jamal	0 slots
Sadiq	4 slots
Matin	0 slots
Хохан	0 slots
humba	1 slots
Rauf	0 slots
Smallll	0 slots



Assignees

Assignee	Count
Camalzadeh	8
Copilot	2
fakhriyyaa	3
Rafetikus	5
Said2911	5

Filter by keyword or by field

Title

Backlog (3 Estimate: 0) This item hasn't been started ...

- 1 Change structure of Base and Home #29
- 2 Change Readme file #39
- 3 Modularize the project structure #49

+ Add item

In progress (2 Estimate: 0) This is actively being worked on ...

- 4 Update poll/create screen #17
- 5 Add presentation of our project #40

+ Add item

Done (16 Estimate: 0) This has been completed ...

- 6 Create new database architecture #8
- 7 Create well designed poll/create screen #6
- 8 Create model integration tests #12
- 9 Create unit tests #11
- 10 Update design of poll/[id] screen #9
- 11 Update Readme file #18
- 12 Update Project Version #48
- 13 Search by Id at home screen #28
- 14 Fix current app issues #33
- 15 Add Unit tests #37
- 16 Add Websocket for realtime communication #16
- 17 Create API integration tests #14
- 18 Set limits at the backend part also #27
- 19 Update home screen #24
- 20 Add Integration Test #38

+ Add item

Show empty values

DEVELOPMENT WORKFLOW

✓ Task Management

- GitHub Issues used to track features and bugs

🔧 Branching Strategy

- Feature branches kept main branch stable

📝 Pull Requests

- Code reviewed before merging

🧪 Testing

- Integration tests run before each merge



Project Structure

Our project is organized into clear modules: **app** for pages and API routes, **components** for reusable UI blocks, **lib** and **utils** for shared logic, **models** for data schemas, **types** for TypeScript definitions, and **tests** for automated testing.

```
> IN_MY_TIME
> ⚡ .github
> 📂 .next
> 📂 app
> 📂 lib
> 📂 models
> 📂 node_modules
> 📂 public
> 📂 tests
> 📂 TS types
```



Application Architecture

The project uses TypeScript across both frontend and backend. The frontend is built with Next.js and React, while the backend uses Next.js API routes with MongoDB and Mongoose. Jest is used for integration testing.

Example of FrontEnd (Header.tsx)

```
1 "use client";
2 import { useState } from "react";
3 import Link from "next/link";
4 import { Calendar, Search, X } from "lucide-react";
5 import { UI_PATHS } from "@/lib/routes";
6 import HeaderSearch from "@/app/components/search/HeaderSearch"
7
8
9 export default function Header() {
10   const [isSearchOpen, setIsSearchOpen] = useState(false);
11
12   return (
13     <nav className="sticky top-0 z-50 border-b border-bord
14       <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:p
15         <div className="flex items-center justify-between
16           <Link href="/" className="flex items-center
17             <div
18               className="w-8 h-8 bg-gradient-to-b
19                 <Calendar className="w-5 h-5 text-p
20               </div>
21             <span className="text-xl font-bold text
22           </Link>
23
24           <div className="hidden md:flex items-center
25             <a href="#features"
26               className="text-sm text-foreground/6
27               Features
28             </a>
29             <a href="#how-it-works"
30               className="text-sm text-foreground/6
31               How It Works
32             </a>
33           </div>
34         </div>
35       </div>
36     </nav>
37   );
38 }
```

Example of BackEnd (route.ts)

```
1 import { NextRequest, NextResponse } from 'next/server';
2 import { connectDB } from '@/lib/mongodb';
3 import { Poll } from '@/models/Poll';
4 import mongoose from 'mongoose';
5
6 interface RouteContext {
7   params: Promise<{
8     id: string;
9   }>;
10 }
11
12 export async function POST(
13   request: NextRequest,
14   context: RouteContext
15 ) {
16   const { id } = await context.params;
17   const pollId = id;
18   try {
19     const { status, finalDate } = await request.json();
20
21     if (status !== 'closed' || !finalDate) {
22       return NextResponse.json({ message: "Invalid payload" });
23     }
24
25     await connectDB();
26
27     const poll = await Poll.findById(pollId);
28
29     if (!poll) {
30       return NextResponse.json({ message: "Poll not found" });
31     }
32
33     poll.status = status;
34     poll.finalDate = finalDate;
35
36     await poll.save();
37
38     return NextResponse.json(poll);
39   } catch (error) {
40     console.error(error);
41     return NextResponse.json({ message: "Internal server error" }, { status: 500 });
42   }
43 }
```



Security Update: React RSC Vulnerability Fix

Why we updated

- A critical vulnerability was discovered in React Server Components (CVE-2025-55182)
- Affected version: 19.2.0
- The issue allowed unauthenticated remote code execution

What we changed

- Updated React RSC package from 19.2.0 → 19.2.1

Impact on our project

- Our app is now protected against the vulnerability
- No changes to UX/functionality
- Update required only back-end build dependencies



InMyTime

Page 12

Thank you for your attention.

With InMyTime, scheduling becomes faster, clearer, and more collaborative – helping teams stay organized and aligned.

We look forward to improving and expanding the platform together.

Follow Us

InMyTime

-

Our GitHub