readme.md for @api.global/typedserver

A powerful TypeScript-first web server framework for building modern full-stack applications. Features static file serving, live reload, type-safe API integration, decorator-based routing, service worker support, and edge computing capabilities. Part of the `@api.global` ecosystem.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

□ Features

- □□ **Type-Safe API** — Full TypeScript support with `@api.global/typedrequest` and `@api.global/typedsocket`
- □□ **Decorator Routing** — Clean, expressive routing with `@Route`, `@Get`, `@Post` decorators via smartserve
- □□ **Security Headers** — Built-in CSP, HSTS, X-Frame-Options, and comprehensive security configuration
- ⚡ **Live Reload** — Automatic browser refresh on file changes during development
- □□ **Service Worker** — Advanced caching, offline support, and background sync
- ☁ **Edge Workers** — Cloudflare Workers compatible edge computing with domain routing
- □□ **WebSocket** — Real-time bidirectional communication via TypedSocket
- □□ **SEO Tools** — Built-in sitemap, RSS feed, and robots.txt generation
- □□ **SPA Support** — Single-page application fallback routing
- □□ **PWA Ready** — Web App Manifest generation for progressive web apps
- □□ **Compression** — Automatic Brotli + Gzip response compression
- □□ **Bundled Content** — Serve pre-bundled content from memory for zero-filesystem deployments

□□ Installation

```
# Using pnpm (recommended)
pnpm add @api.global/typedserver

# Using npm
npm install @api.global/typedserver
```

□□ Quick Start

Basic Server

```
import { TypedServer } from '@api.global/typedserver';

const server = new TypedServer({
  serveDir: './public',
  cors: true,
  watch: true,           // Enable file watching
  injectReload: true,    // Inject live reload script
});

await server.start();
console.log('Server running on port 3000!');
```

Full Configuration

```
import { TypedServer } from '@api.global/typedserver';

const server = new TypedServer({
  port: 8080,
  serveDir: './dist',
  cors: true,

  // Development
```

```
  watch: true,
  injectReload: true,
  noCache: true,            // Disable browser caching

  // Production
  forceSsl: true,
  spaFallback: true,        // Serve index.html for client-side routes

  // SEO
  sitemap: true,
  feed: true,
  robots: true,
  domain: 'example.com',
  blockWaybackMachine: false,

  // PWA
  appVersion: 'v1.0.0',
  manifest: {
    name: 'My App',
    short_name: 'myapp',
    start_url: '/',
    display: 'standalone',
    background_color: '#ffffff',
    theme_color: '#000000',
  },

  // Compression
  compression: {
    enabled: true,
    algorithms: ['br', 'gzip'],
    threshold: 1024,
  },
});

await server.start();
```

□□ Routing

TypedServer uses a unified routing system powered by `@push.rocks/smartserve`. You can add routes using decorators or the programmatic API.

Decorator-Based Routing

Create clean, expressive controllers using decorators:

```
import * as smartserve from '@push.rocks/smartserve';

@smartserve.Route('/api/users')
class UserController {
  @smartserve.Get('/')
  async listUsers(ctx: smartserve.IRequestContext): Promise<Response> {
    const users = await getUsersFromDb();
    return new Response(JSON.stringify(users), {
      headers: { 'Content-Type': 'application/json' },
    });
  }

  @smartserve.Get('/:id')
  async getUser(ctx: smartserve.IRequestContext): Promise<Response> {
    const userId = ctx.params.id;
    const user = await getUserById(userId);
    return new Response(JSON.stringify(user), {
      headers: { 'Content-Type': 'application/json' },
    });
  }

  @smartserve.Post('/')
  async createUser(ctx: smartserve.IRequestContext): Promise<Response> {
    const userData = await ctx.json();
    const newUser = await createUserInDb(userData);
    return new Response(JSON.stringify(newUser), {
      status: 201,
      headers: { 'Content-Type': 'application/json' },
    });
  }
}
```

```
// Register the controller
smartserve.ControllerRegistry.registerInstance(new UserController());
```

Programmatic Routes with `addRoute()`

Add routes dynamically using the `addRoute()` API:

```
import { TypedServer } from '@api.global/typedserver';

const server = new TypedServer({ serveDir: './public', cors: true });

// Simple route
server.addRoute('/api/health', 'GET', async (ctx) => {
  return new Response(JSON.stringify({ status: 'ok' }), {
    headers: { 'Content-Type': 'application/json' },
  });
});

// Route with parameters (Express-style :param syntax)
server.addRoute('/api/items/:id', 'GET', async (ctx) => {
  const itemId = ctx.params.id;
  return new Response(JSON.stringify({ id: itemId }), {
    headers: { 'Content-Type': 'application/json' },
  });
});

// Wildcard routes
server.addRoute('/files/*path', 'GET', async (ctx) => {
  const filePath = ctx.params.path;
  return new Response(`Requested: ${filePath}`);
});

await server.start();
```

□□ Type-Safe API Integration

Adding TypedRequest Handlers

```
import { TypedServer } from '@api.global/typedserver';
import * as typedrequest from '@api.global/typedrequest';

// Define your typed request interface
interface IGetUser extends typedrequest.implementsTR<IGetUser> {
  method: 'getUser';
  request: { userId: string };
  response: { name: string; email: string };
}

const server = new TypedServer({ serveDir: './public', cors: true });

// Add a typed handler directly to the server's router
server.typedrouter.addTypedHandler<IGetUser>(
  new typedrequest.TypedHandler('getUser', async (data) => {
    return { name: 'John Doe', email: 'john@example.com' };
  })
);

await server.start();
```

Real-Time WebSocket Communication

TypedServer automatically sets up TypedSocket for real-time communication:

```
import { TypedServer } from '@api.global/typedserver';
import * as typedrequest from '@api.global/typedrequest';

interface IChatMessage extends typedrequest.implementsTR<IChatMessage> {
  method: 'sendMessage';
  request: { text: string; room: string };
  response: { messageId: string; timestamp: number };
}

const server = new TypedServer({ serveDir: './public', cors: true });
```

```
// Handle real-time messages
server.typedrouter.addTypedHandler<IChatMessage>(
  new typedrequest.TypedHandler('sendMessage', async (data) => {
    return { messageId: crypto.randomUUID(), timestamp: Date.now() };
  })
);

await server.start();

// Push messages to connected clients
const connections = await server.typedsocket.findAllTargetConnectionsByTag('allClients');
for (const conn of connections) {
  // Push to specific clients via TypedSocket
}
```

☁ Edge Worker (Cloudflare Workers)

Deploy your application to the edge with Cloudflare Workers:

```
import { EdgeWorker, DomainRouter } from '@api.global/typedserver/edgeworker';

const worker = new EdgeWorker();

// Configure domain routing with caching
worker.domainRouter.addDomainInstruction({
  domainPattern: '*.example.com',
  originUrl: 'https://origin.example.com',
  type: 'cache',
  cacheConfig: { maxAge: 3600 },
});

// Pass-through to origin for API routes
worker.domainRouter.addDomainInstruction({
  domainPattern: 'api.example.com',
  originUrl: 'https://api-origin.example.com',
```

```
  type: 'origin',
});

// Cloudflare Worker entry point
export default {
  fetch: worker.fetchFunction.bind(worker),
};
```

□□ Service Worker Client

Manage service workers in your frontend application:

```
import { getServiceworkerClient } from '@api.global/typedserver/web_serviceworker_client';

// Initialize and register service worker
const swClient = await getServiceworkerClient({
  pollInterval: 30000,  // Poll for updates every 30s
});

// The service worker handles:
// - Cache invalidation from server
// - Offline support
// - Background sync
// - Version updates
```

□□ Bundled Content

Serve pre-bundled content directly from memory — useful for single-binary deployments or embedding assets in server-side code:

```
import { TypedServer } from '@api.global/typedserver';

const server = new TypedServer({
  cors: true,
  bundledContent: [
    {
      path: '/index.html',
```

```
      contentBase64: Buffer.from('<html><body>Hello!</body></html>').toString('base64'),
    },
    {
      path: '/app.js',
      contentBase64: Buffer.from('console.log("loaded")').toString('base64'),
    },
  ],
  spaFallback: true,
});

await server.start();
```

Bundled content takes priority over filesystem serving and supports ETag-based conditional requests with immutable caching.

□□ Security Headers

Configure comprehensive security headers including CSP, HSTS, and more:

```
import { TypedServer } from '@api.global/typedserver';

const server = new TypedServer({
  serveDir: './dist',
  cors: true,

  securityHeaders: {
    // Content Security Policy
    csp: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'", "'unsafe-inline'", 'https://cdn.example.com'],
      styleSrc: ["'self'", "'unsafe-inline'"],
      imgSrc: ["'self'", 'data:', 'https:'],
      connectSrc: ["'self'", 'wss:', 'https://api.example.com'],
      fontSrc: ["'self'", 'https://fonts.gstatic.com'],
      frameAncestors: ["'none'"],
      upgradeInsecureRequests: true,
    },
```

```
    // HSTS (HTTP Strict Transport Security)
    hstsMaxAge: 31536000,         // 1 year
    hstsIncludeSubDomains: true,
    hstsPreload: true,

    // Other security headers
    xFrameOptions: 'DENY',
    xContentTypeOptions: true,
    xXssProtection: true,
    referrerPolicy: 'strict-origin-when-cross-origin',

    // Cross-Origin policies
    crossOriginOpenerPolicy: 'same-origin',
    crossOriginEmbedderPolicy: 'require-corp',
    crossOriginResourcePolicy: 'same-origin',

    // Permissions Policy
    permissionsPolicy: {
      camera: [],
      microphone: [],
      geolocation: ['self'],
    },
  },
});

await server.start();
```

Security Headers Reference

Header	Option	Description
`Content-Security-Policy`	`csp`	Controls resources the browser can load
`Strict-Transport-Security`	`hstsMaxAge`, `hstsIncludeSubDomains`, `hstsPreload`	Forces HTTPS connections
`X-Frame-Options`	`xFrameOptions`	Prevents clickjacking attacks
`X-Content-Type-Options`	`xContentTypeOptions`	Prevents MIME-sniffing
`X-XSS-Protection`	`xXssProtection`	Legacy XSS filter
`Referrer-Policy`	`referrerPolicy`	Controls referrer information

Header	Option	Description
`Permissions-Policy`	`permissionsPolicy`	Controls browser features
`Cross-Origin-Opener-Policy`	`crossOriginOpenerPolicy`	Isolates browsing context
`Cross-Origin-Embedder-Policy`	`crossOriginEmbedderPolicy`	Controls cross-origin embedding
`Cross-Origin-Resource-Policy`	`crossOriginResourcePolicy`	Controls cross-origin resource sharing

□□ Compression

TypedServer supports automatic response compression using Brotli and Gzip. Compression is powered by smartserve and enabled by default.

Configuration

```
import { TypedServer } from '@api.global/typedserver';

const server = new TypedServer({
  serveDir: './dist',
  cors: true,

  // Enable with defaults (brotli + gzip, threshold: 1024 bytes)
  compression: true,

  // Or disable completely
  // compression: false,

  // Or configure in detail
  // compression: {
  //   enabled: true,
  //   algorithms: ['br', 'gzip'],    // Preferred order
  //   threshold: 1024,               // Min size to compress (bytes)
  //   level: 4,                      // Compression level (1-11 for brotli, 1-9 for gzip)
  //   exclude: ['/api/stream/*'],    // Skip these paths
  // },
});
```

Compression Options Reference

Option	Type	Default	Description
`enabled`	`boolean`	`true`	Enable/disable compression
`algorithms`	`string[]`	`['br', 'gzip']`	Preferred algorithms in order
`threshold`	`number`	`1024`	Minimum response size (bytes) to compress
`level`	`number`	`4`	Compression level (1-11 for brotli, 1-9 for gzip)
`compressibleTypes`	`string[]`	auto	MIME types to compress
`exclude`	`string[]`	`[]`	Path patterns to skip

□□ Configuration Reference

IServerOptions

Option	Type	Default	Description
`serveDir`	`string`	—	Directory to serve static files from
`bundledContent`	`IBundledContentItem[]`	—	Base64-encoded files to serve from memory
`port`	`number | string`	`3000`	Port to listen on
`cors`	`boolean`	`true`	Enable CORS headers
`watch`	`boolean`	`false`	Watch files for changes
`injectReload`	`boolean`	`false`	Inject live reload script into HTML
`noCache`	`boolean`	`false`	Disable browser caching via response headers
`forceSsl`	`boolean`	`false`	Redirect HTTP to HTTPS
`spaFallback`	`boolean`	`false`	Serve index.html for non-file routes
`sitemap`	`boolean`	`false`	Generate sitemap at `/sitemap`

Option	Type	Default	Description
`feed`	`boolean`	`false`	Generate RSS feed at `/feed`
`robots`	`boolean`	`false`	Serve robots.txt
`domain`	`string`	—	Domain name for sitemap/feeds
`appVersion`	`string`	—	Application version string
`manifest`	`object`	—	Web App Manifest configuration
`publicKey`	`string`	—	SSL certificate
`privateKey`	`string`	—	SSL private key
`defaultAnswer`	`function`	—	Custom default response handler
`feedMetadata`	`object`	—	RSS feed metadata options
`blockWaybackMachine`	`boolean`	`false`	Block Wayback Machine archiving
`securityHeaders`	`ISecurityHeaders`	—	Security headers configuration
`compression`	`ICompressionConfig | boolean`	`true`	Response compression configuration

□□ Package Exports

```
@api.global/typedserver
├── .                          – Main server (TypedServer)
├── /backend                   – Alias for main server
├── /infohtml                  – Info HTML page generator
├── /edgeworker                – Cloudflare Workers edge computing
├── /web_inject                – Live reload script injection
├── /web_serviceworker         – Service Worker implementation
└── /web_serviceworker_client  – Service Worker client utilities
```

□□ Utility Servers

Pre-configured server templates with best practices built-in.

UtilityWebsiteServer

Optimized for modern web applications with SPA support, live reload, and caching disabled by default during development:

```
import { utilityservers } from '@api.global/typedserver';

const websiteServer = new utilityservers.UtilityWebsiteServer({
  serveDir: './dist',
  domain: 'example.com',

  // SPA fallback enabled by default
  spaFallback: true,   // default: true

  // Security headers
  securityHeaders: {
    csp: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'", "'unsafe-inline'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
    },
    xFrameOptions: 'SAMEORIGIN',
    xContentTypeOptions: true,
  },

  // Compression (enabled by default)
  compression: true,

  // Other options
  cors: true,          // default: true
  forceSsl: false,     // default: false
  appSemVer: '1.0.0',
  port: 3000,          // default: 3000

  // Optional ads.txt entries (only served if configured)
  adsTxt: [
    'google.com, pub-1234567890, DIRECT, f08c47fec0942fa0',
  ],
```

```
  // RSS feed metadata
  feedMetadata: {
    title: 'My Blog',
    description: 'A cool blog',
    link: 'https://example.com',
  },

  // Add custom routes
  addCustomRoutes: async (typedserver) => {
    typedserver.addRoute('/api/custom', 'GET', async () => {
      return new Response('Custom route!');
    });
  },
});

await websiteServer.start();
```

UtilityServiceServer

Optimized for API services with auto-generated info page:

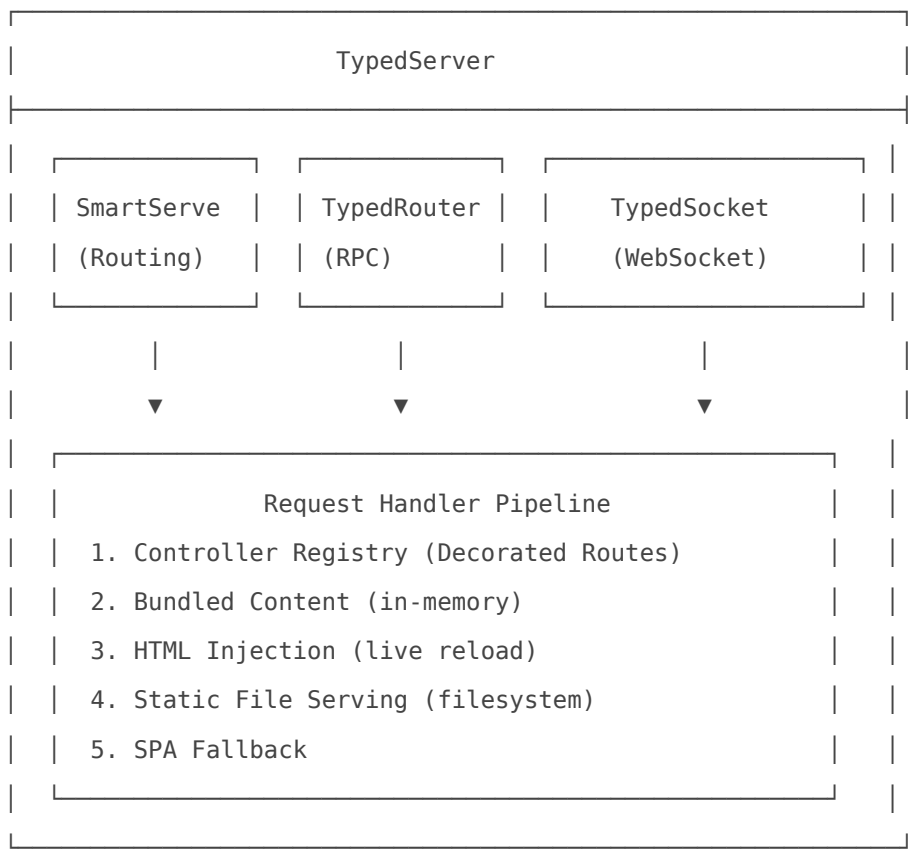
```
import { utilityservers } from '@api.global/typedserver';

const serviceServer = new utilityservers.UtilityServiceServer({
  serviceName: 'My API',
  serviceVersion: '1.0.0',
  serviceDomain: 'api.example.com',
  port: 8080,

  // Add custom routes
  addCustomRoutes: async (typedserver) => {
    typedserver.addRoute('/api/status', 'GET', async () => {
      return new Response(JSON.stringify({ status: 'healthy' }), {
        headers: { 'Content-Type': 'application/json' },
      });
    });
  },
});
```

```
await serviceServer.start();
```

□□ Architecture Overview



License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #7
Created 2026-03-28 10:48:09 UTC by foss.global Team
Updated 2026-03-28 12:13:19 UTC by foss.global Team