ДОДАТОК А
Лістинг програми

```
// This is Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {
  provider = "prisma-client-js"
  output   = "./client"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

model User {
  id          Int      @id @default(autoincrement())
  first_name   String?
  last_name    String?
  username     String   @unique
  password_hash String
  role         String   @default("USER")
  is_active    Boolean  @default(true)
  created_at   DateTime @default(now())
  updated_at   DateTime @updatedAt
  preferences  UserPreferences?

  @@map("users")
}

model UserPreferences {
  id          Int      @id @default(autoincrement())
  user_id      Int      @unique
  language     String   @default("uk")
  sidenav_type String   @default("DARK")
  navbar_fixed Boolean  @default(true)
  light_theme  Boolean  @default(true)
  created_at   DateTime @default(now())
  updated_at   DateTime @updatedAt
  user         User     @relation(fields: [user_id], references: [id], onDelete: Cascade)

  @@map("user_preferences")
}

model LicenseInfo {
  id              Int      @id @default(autoincrement())
  expiration_date DateTime
  max_users       Int
  created_at      DateTime @default(now())
  updated_at      DateTime @updatedAt

  @@map("license_info")
}

model ImportTemplate {
```

```
  id         Int      @id @default(autoincrement())
  name       String
  type       Int
  mapping    Json
  created_at DateTime @default(now())
  updated_at DateTime @updatedAt
}

//
// Моделі для системи управління конвеєрною лінією
//

/// Модель "Підприємства"
/// Підприємства, на яких впроваджена система управління конвеєрною лінією
model Company {
  id          Int      @id @default(autoincrement())
  designation String   @unique
  name        String   @unique
  created_at  DateTime @default(now())
  updated_at  DateTime @updatedAt

  departments Department[]
  production_lines ProductionLine[]

  @@map("companies")
}

/// Модель "Виробничі підрозділи"
/// Підрозділи підприємства, де розгорнута система управління
model Department {
  id          Int      @id @default(autoincrement())
  company_id  Int
  designation String   @unique
  name        String   @unique
  created_at  DateTime @default(now())
  updated_at  DateTime @updatedAt

  company     Company  @relation(fields: [company_id], references: [id])
  employees   Employee[]
  production_lines ProductionLine[]

  @@map("departments")
}

/// Модель "Конвеєрні лінії"
/// Технологічні лінії, що підлягають автоматизації
model ProductionLine {
  id           Int      @id @default(autoincrement())
  company_id   Int
  department_id Int
  designation  String   @unique
  name         String   @unique
  description  String?
  created_at   DateTime @default(now())
  updated_at   DateTime @updatedAt

  company      Company   @relation(fields: [company_id], references: [id])
```

```
  department    Department @relation(fields: [department_id], references: [id])
  technological_conditions TechnologicalCondition[]
  control_systems ControlSystem[]
  hmi_interfaces HMIInterface[]

  @@map("production_lines")
}

/// Модель "Технологічні умови"
/// Умови технологічного процесу конвеєрної лінії
model TechnologicalCondition {
  id              Int      @id @default(autoincrement())
  production_line_id Int
  designation       String   @unique
  name             String   @unique
  description       String?
  created_at        DateTime @default(now())
  updated_at        DateTime @updatedAt

  production_line   ProductionLine @relation(fields: [production_line_id], references: [id])
  process_parameters ProcessParameter[]

  @@map("technological_conditions")
}

/// Модель "Технологічні параметри"
/// Параметри технологічного процесу
model ProcessParameter {
  id          Int      @id @default(autoincrement())
  condition_id Int
  designation String   @unique
  name        String   @unique
  description String?
  created_at  DateTime @default(now())
  updated_at  DateTime @updatedAt

  technological_condition TechnologicalCondition @relation(fields: [condition_id], references: [id])
  control_points ControlPoint[]

  @@map("process_parameters")
}

/// Модель "Системи управління"
/// Автоматизовані системи управління конвеєрною лінією
model ControlSystem {
  id              Int      @id @default(autoincrement())
  production_line_id Int
  designation       String   @unique
  name             String   @unique
  system_type       String
  description       String?
  created_at        DateTime @default(now())
  updated_at        DateTime @updatedAt

  production_line   ProductionLine @relation(fields: [production_line_id], references: [id])
  control_points    ControlPoint[]
  hmi_interfaces    HMIInterface[]
```

```
  @@map("control_systems")
}

/// Модель "Інтерфейси HMI"
/// Людино-машинні інтерфейси системи управління
model HMIInterface {
  id              Int      @id @default(autoincrement())
  production_line_id Int
  control_system_id Int
  designation      String   @unique
  name             String   @unique
  interface_type    String
  description      String?
  created_at       DateTime @default(now())
  updated_at       DateTime @updatedAt

  production_line   ProductionLine @relation(fields: [production_line_id], references: [id])
  control_system    ControlSystem @relation(fields: [control_system_id], references: [id])
  control_points    ControlPoint[]

  @@map("hmi_interfaces")
}

/// Модель "Точки контролю"
/// Контрольні точки системи управління
model ControlPoint {
  id              Int      @id @default(autoincrement())
  parameter_id     Int
  control_system_id Int
  hmi_interface_id  Int
  designation      String   @unique
  name             String   @unique
  location         String?
  description      String?
  created_at       DateTime @default(now())
  updated_at       DateTime @updatedAt

  process_parameter ProcessParameter @relation(fields: [parameter_id], references: [id])
  control_system    ControlSystem @relation(fields: [control_system_id], references: [id])
  hmi_interface     HMIInterface @relation(fields: [hmi_interface_id], references: [id])
  sensors           Sensor[]
  actuators         Actuator[]

  @@map("control_points")
}

/// Модель "Посади співробітників"
model EmployeePosition {
  id          Int      @id @default(autoincrement())
  designation String   @unique
  name        String   @unique
  created_at  DateTime @default(now())
  updated_at  DateTime @updatedAt

  employees   Employee[]
```

```
  @@map("employee_positions")
}

/// Модель "Ролі співробітників"
model EmployeeRole {
  id          Int      @id @default(autoincrement())
  designation String   @unique
  name        String   @unique
  created_at  DateTime @default(now())
  updated_at  DateTime @updatedAt

  employees   Employee[]

  @@map("employee_roles")
}

/// Модель "Співробітники"
/// Персонал, що взаємодіє з системою управління
model Employee {
  id           Int      @id @default(autoincrement())
  department_id Int
  position_id   Int
  role_id       Int
  last_name     String
  first_name    String
  middle_name   String
  short_full_name String
  created_at    DateTime @default(now())
  updated_at    DateTime @updatedAt

  department  Department @relation(fields: [department_id], references: [id])
  position    EmployeePosition @relation(fields: [position_id], references: [id])
  role        EmployeeRole @relation(fields: [role_id], references: [id])

  @@map("employees")
}

/// Модель "Типи датчиків"
model SensorType {
  id          Int      @id @default(autoincrement())
  sensor_type String   @unique
  name        String   @unique
  description String?
  created_at  DateTime @default(now())
  updated_at  DateTime @updatedAt

  sensors Sensor[]

  @@map("sensor_types")
}

/// Модель "Датчики"
/// Вимірювальні пристрої системи управління
model Sensor {
  id           Int      @id @default(autoincrement())
  type_id       Int
  control_point_id Int
```

```
  serial_number String
  production_year Int
  designation   String   @unique
  name          String   @unique
  description   String?
  created_at    DateTime @default(now())
  updated_at    DateTime @updatedAt

  sensor_type   SensorType @relation(fields: [type_id], references: [id])
  control_point ControlPoint @relation(fields: [control_point_id], references: [id])

  @@map("sensors")
}

/// Модель "Виконавчі механізми"
/// Пристрої, що реалізують керуючі впливи
model Actuator {
  id           Int      @id @default(autoincrement())
  control_point_id Int
  designation   String   @unique
  name          String   @unique
  actuator_type String
  description   String?
  created_at    DateTime @default(now())
  updated_at    DateTime @updatedAt

  control_point ControlPoint @relation(fields: [control_point_id], references: [id])

  @@map("actuators")
}

/// Модель "Типи технічних одиниць"
model EngineeringUnitType {
  id         Int      @id @default(autoincrement())
  designation String   @unique
  name       String   @unique
  created_at  DateTime @default(now())
  updated_at  DateTime @updatedAt

  units EngineeringUnit[]

  @@map("engineering_unit_types")
}

/// Модель "Технічні одиниці"
model EngineeringUnit {
  id         Int      @id @default(autoincrement())
  type_id     Int
  designation String   @unique
  name       String   @unique
  created_at  DateTime @default(now())
  updated_at  DateTime @updatedAt

  type EngineeringUnitType @relation(fields: [type_id], references: [id])

  @@map("engineering_units")
}
```

Основний компонент серверу

```
// Завантаження змінних оточення
import dotenv from "dotenv";
dotenv.config();

import express from "express";
import session from "express-session";
import SQLiteStore from "connect-sqlite3";
import chalk from "chalk";
import fs from "fs";
import cookieParser from "cookie-parser";

// Імпорти локальних модулів
import { config } from "./config/index.js";
import { isAuthenticated } from "./middleware/auth.js";
import packageJson from "../package.json" with { type: "json" };
import { prisma } from "./lib/prisma.js";

// Імпорти роутерів
import { authRouter } from "./routes/auth.js";
import { usersRouter } from "./routes/users.js";
import { preferencesRouter } from "./routes/preferences.js";
import licenseRouter from "./routes/license.js";
import importRouter from "./routes/import.js";
import importTxtRouter from "./routes/import-txt.js";
import importExcelRouter from "./routes/import-excel.js";
import { importTemplatesRouter } from "./routes/import-templates.js";

// Роути організаційної структури
import { companiesRouter } from "./routes/companies.js";
import { departmentsRouter } from "./routes/departments.js";
import { employeePositionsRouter } from "./routes/employee-positions.js";
import { employeeRolesRouter } from "./routes/employee-roles.js";
import { employeesRouter } from "./routes/employees.js";

// Роути системи управління конвеєрною лінією
import { productionLinesRouter } from "./routes/production-lines.js";
import { technologicalConditionsRouter } from "./routes/technological-conditions.js";
import { processParametersRouter } from "./routes/process-parameters.js";
import { controlSystemsRouter } from "./routes/control-systems.js";
import { hmiInterfacesRouter } from "./routes/hmi-interfaces.js";
import { controlPointsRouter } from "./routes/control-points.js";

// Роути обладнання
import { sensorTypesRouter } from "./routes/sensor-types.js";
import { sensorsRouter } from "./routes/sensors.js";
import { actuatorsRouter } from "./routes/actuators.js";

// Роути вимірювань
import { engineeringUnitTypesRouter } from "./routes/engineering-unit-types.js";
import { engineeringUnitsRouter } from "./routes/engineering-units.js";

// Функція для обробки шляхів зі змінними оточення Windows
function expandWindowsPath(pathWithEnvVars) {
  return pathWithEnvVars.replace(/%([^%]+)%/g, (_, n) => process.env[n] || "");
```

```
}

// Константи
const { description, version } = packageJson;
const isDev = ((process.env.NODE_ENV ?? "development").trim().toLowerCase() === "development");
const SQLiteStoreInstance = SQLiteStore(session);

// Підготовка шляхів для баз даних
const dataPath = expandWindowsPath(
  process.env.DATA_PATH || "./",
);

// Створюємо директорію, якщо не існує
if (!fs.existsSync(dataPath)) {
  try {
    fs.mkdirSync(dataPath, { recursive: true });
  } catch (error) {
    console.error("Помилка створення директорії даних:", error);
  }
}

// Ініціалізація додатку
const app = express();

// Middleware
app.use(express.json({ limit: '100mb' }));
app.use(cookieParser());

// Налаштування CORS до сесії
app.use((req, res, next) => {
  const origin = req.headers.origin || "http://127.0.0.1:3000";

  res.header("Access-Control-Allow-Origin", origin);
  res.header("Access-Control-Allow-Credentials", "true");
  res.header("Access-Control-Allow-Methods", "GET,HEAD,PUT,PATCH,POST,DELETE");
  res.header(
    "Access-Control-Allow-Headers",
    "Content-Type, Accept, Authorization, Cookie",
  );
  res.header("Access-Control-Expose-Headers", "Set-Cookie");

  if (req.method === "OPTIONS") {
    return res.sendStatus(200);
  }

  next();
});

// Створюємо сховище SQLite
const store = new SQLiteStoreInstance({
  table: "sessions",
  dir: dataPath,
  db: "sessions.db",
  concurrentDB: true,
  verbose: true,
});
```

```
// Обробка помилок сховища
store.on("error", function (error) {
  console.error("Помилка SQLite сховища:", error);
});

// Налаштування сесій
const sessionConfig = {
  ...config.session,
  store: store,
};

// Очищаємо старі сесії при запуску
sessionConfig.store.clear((error) => {
  if (error) {
    console.error("Помилка очищення сесій:", error);
  }
});

// Налаштування сесій
app.use(session(sessionConfig));

// Перевіряємо стан сесії перед кожним запитом
app.use((req, res, next) => {
  if (!req.session.userId && req.path !== "/api/auth/login") {
    console.log("Перевірка сесії не пройдена:", {
      path: req.path,
      sessionId: req.sessionID,
      userId: req.session.userId,
    });
  }
  next();
});

// Роути API
app.use("/api/auth", authRouter);
app.use("/api/users", isAuthenticated, usersRouter);
app.use("/api/preferences", isAuthenticated, preferencesRouter);
app.use("/api/license", isAuthenticated, licenseRouter);
app.use("/api/import", isAuthenticated, importRouter);
app.use("/api/import-txt", isAuthenticated, importTxtRouter);
app.use("/api/import-excel", isAuthenticated, importExcelRouter);
app.use("/api/import-templates", isAuthenticated, importTemplatesRouter);

// Роути організаційної структури
app.use("/api/companies", isAuthenticated, companiesRouter);
app.use("/api/departments", isAuthenticated, departmentsRouter);
app.use("/api/employee-positions", isAuthenticated, employeePositionsRouter);
app.use("/api/employee-roles", isAuthenticated, employeeRolesRouter);
app.use("/api/employees", isAuthenticated, employeesRouter);

// Роути системи управління конвеєрною лінією
app.use("/api/production-lines", isAuthenticated, productionLinesRouter);
app.use("/api/technological-conditions", isAuthenticated, technologicalConditionsRouter);
app.use("/api/process-parameters", isAuthenticated, processParametersRouter);
app.use("/api/control-systems", isAuthenticated, controlSystemsRouter);
app.use("/api/hmi-interfaces", isAuthenticated, hmiInterfacesRouter);
app.use("/api/control-points", isAuthenticated, controlPointsRouter);
```

```
// Роути обладнання
app.use("/api/sensor-types", isAuthenticated, sensorTypesRouter);
app.use("/api/sensors", isAuthenticated, sensorsRouter);
app.use("/api/actuators", isAuthenticated, actuatorsRouter);

// Роути вимірювань
app.use("/api/engineering-unit-types", isAuthenticated, engineeringUnitTypesRouter);
app.use("/api/engineering-units", isAuthenticated, engineeringUnitsRouter);

// Глобальний обробник необроблених помилок
process.on('uncaughtException', (error) => {
  console.error('Необроблена помилка:', error);
});

// Глобальний обробник необроблених Promise rejection
process.on('unhandledRejection', (reason, promise) => {
  console.error('Необроблений Rejection у:', promise, 'причина:', reason);
});

// Обробник помилок Express
app.use((err, req, res, next) => {
  console.error('Помилка Express:', err);
  res.status(500).json({
    error: isDev ? err.message : "Внутрішня помилка сервера",
  });
});

// Очищаємо всі сесії при запуску
await new Promise((resolve) => {
  store.clear((error) => {
    if (error) {
      console.error("Помилка очищення сесій:", error);
    }
    resolve();
  });
});

// Запуск сервера
const server = app.listen(config.server.port, () => {
  const sessionMaxAge = config.session.cookie.maxAge / 1000;
  const hours = Math.floor(sessionMaxAge / 3600);
  const minutes = Math.floor((sessionMaxAge % 3600) / 60);

  console.log("\n" + chalk.white.bold(`${description} v${version}`));
  console.log(chalk.gray("----------------------------------------"));
  console.log(
    chalk.white(" Режим       ") +
      (isDev ? chalk.yellow("Розробка") : chalk.blue("Продукція")),
  );
  console.log(
    chalk.white(" Адреса     ") +
      chalk.cyan(`${config.server.host}:${config.server.port}`),
  );
  console.log(
    chalk.white(" Сесія      ") +
      chalk.cyan(
```

```
      `${hours.toString().padStart(2, "0")}:${minutes.toString().padStart(2, "0")} (ГГ:ХХ)`,
    ),
  );
  console.log(chalk.white(" CORS       ") + chalk.cyan(config.cors.origin));
  console.log(chalk.gray("----------------------------------------\n"));
});

// Обробка завершення роботи
[
  "EXIT",
  "SIGINT",
  "SIGUSR1",
  "SIGUSR2",
  "uncaughtException",
  "SIGTERM",
].forEach((eventType) => {
  const previousListeners = process.listeners(eventType);
  process.removeAllListeners(eventType);

  process.once(eventType, async (param) => {
    await gracefulShutdown();

    previousListeners.forEach((l) => l(param));

    if (eventType !== "EXIT") {
      process.exit();
    }
  });
});

async function gracefulShutdown() {
  console.log("Початок планового завершення...");

  const forceExitTimeout = setTimeout(() => {
    console.error(
      "Не вдалося закрити з'єднання вчасно, примусове завершення",
    );
    process.exit(1);
  }, 15000);

  try {
    await new Promise((resolve, reject) => {
      server.close((err) => {
        if (err) {
          reject(err);
          return;
        }
        console.log("HTTP сервер закрито");
        resolve();
      });
    });

    await prisma.$disconnect();
    console.log("З'єднання з базою даних закрито");

    await new Promise((resolve) => {
      store.destroy((err) => {
```

```
      if (err) {
        console.error("Помилка закриття сховища сесій:", err);
      } else {
        console.log("Сховище сесій закрито");
      }
      resolve();
    });
  });

  clearTimeout(forceExitTimeout);
  console.log("Планове завершення завершено");
 } catch (error) {
  console.error("Помилка під час завершення:", error);
  process.exit(1);
 }
}
```

Апі між сервером та клієнтом

```
import axios from "axios";
import { API_URL, API_ENDPOINTS } from "./api.config";

const api = axios.create({
 baseURL: API_URL,
 withCredentials: true,
});

// Інтерцептор для обробки помилок токена
api.interceptors.response.use(
 (response) => response,
 (error) => {
  const errorCode = error.response?.data?.error;
  const requestUrl = error.config?.url;
  const requestMethod = error.config?.method;

  // Перевіряємо помилки токена
  if (
    errorCode === "TOKEN_MISSING" ||
    errorCode === "TOKEN_INVALID" ||
    errorCode === "TOKEN_EXPIRED" ||
    errorCode === "TOKEN_SESSION_INVALID" ||
    errorCode === "TOKEN_READER_MISSING"
  ) {
    // Генеруємо подію для показу модального вікна
    window.dispatchEvent(
      new CustomEvent("token-validation-required", {
        detail: { error: errorCode, message: error.response?.data?.message },
      })
    );
  }

  return Promise.reject(error);
 }
);

export const auth = {
 login: (username, password) => api.post(API_ENDPOINTS.auth.login, { username, password }),
```

```
  logout: () => api.post(API_ENDPOINTS.auth.logout),
  getMe: () => api.get(API_ENDPOINTS.auth.me),
  checkToken: () => api.get(API_ENDPOINTS.auth.checkToken),
};

export const users = {
  getAll: () => api.get(API_ENDPOINTS.users.base),
  create: (userData) => api.post(API_ENDPOINTS.users.base, userData),
  update: (id, userData) => api.put(API_ENDPOINTS.users.byId(id), userData),
  delete: (id) => api.delete(API_ENDPOINTS.users.byId(id)),
};

export const preferences = {
  get: () => api.get(API_ENDPOINTS.preferences.base),
  update: (preferencesData) => api.put(API_ENDPOINTS.preferences.base, preferencesData),
};

export const license = {
  get: () => api.get(API_ENDPOINTS.license.base),
  update: (data) => api.put(API_ENDPOINTS.license.base, data),
};

export const importTemplates = {
  getAll: () => api.get(API_ENDPOINTS.importTemplates.base).then((res) => res.data),
  create: (data) => api.post(API_ENDPOINTS.importTemplates.base, data).then((res) => res.data),
  update: (id, data) =>
    api.put(API_ENDPOINTS.importTemplates.byId(id), data).then((res) => res.data),
  delete: (id) => api.delete(API_ENDPOINTS.importTemplates.byId(id)).then((res) => res.data),
};

// Роути організаційної структури
export const companies = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) =>
    api.get(API_ENDPOINTS.companies.base, {
      params: {
        page,
        ...(search && { search }),
        ...(sortField && { sortField, sortOrder }),
        ...otherParams,
      },
    }),
  getAllWithoutPagination: () => api.get(`${API_ENDPOINTS.companies.base}/all`),
  create: (data) => api.post(API_ENDPOINTS.companies.base, data),
  update: (id, data) => api.put(API_ENDPOINTS.companies.byId(id), data),
  delete: (id) => api.delete(API_ENDPOINTS.companies.byId(id)),
};

export const departments = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
    const params = {
      page,
      ...(search && { search }),
      ...(sortField && { sortField }),
      ...(sortOrder && { sortOrder }),
      ...otherParams,
    };
    return api.get(API_ENDPOINTS.departments.base, { params });
```

```
  },
  getAllWithoutPagination: () => api.get(API_ENDPOINTS.departments.all),
  create: (data) => api.post(API_ENDPOINTS.departments.base, data),
  update: (id, data) => api.put(API_ENDPOINTS.departments.byId(id), data),
  delete: (id) => api.delete(API_ENDPOINTS.departments.byId(id)),
};

export const employeePositions = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
    return api
      .get(API_ENDPOINTS.employeePositions.base, {
        params: {
          page,
          ...(search && { search }),
          ...(sortField && { sortField, sortOrder }),
          ...otherParams,
        },
      })
      .then((response) => {
        return response;
      });
  },

  getAllWithoutPagination: () => {
    return api.get(`${API_ENDPOINTS.employeePositions.base}/all`);
  },

  create: (data) => {
    return api.post(API_ENDPOINTS.employeePositions.base, data);
  },

  update: (id, data) => {
    return api.put(API_ENDPOINTS.employeePositions.byId(id), data);
  },

  delete: (id) => {
    return api.delete(API_ENDPOINTS.employeePositions.byId(id));
  },
};

export const employeeRoles = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
    return api
      .get(API_ENDPOINTS.employeeRoles.base, {
        params: {
          page,
          ...(search && { search }),
          ...(sortField && { sortField, sortOrder }),
          ...otherParams,
        },
      })
      .then((response) => {
        return response;
      });
  },

  getAllWithoutPagination: () => {
```

```
    return api.get(`${API_ENDPOINTS.employeeRoles.base}/all`);
  },

  create: (data) => {
    return api.post(API_ENDPOINTS.employeeRoles.base, data);
  },

  update: (id, data) => {
    return api.put(API_ENDPOINTS.employeeRoles.byId(id), data);
  },

  delete: (id) => {
    return api.delete(API_ENDPOINTS.employeeRoles.byId(id));
  },
};

export const employees = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
    return api
      .get(API_ENDPOINTS.employees.base, {
        params: {
          page,
          ...(search && { search }),
          ...(sortField && { sortField, sortOrder }),
          ...otherParams,
        },
      })
      .then((response) => {
        return response;
      });
  },

  getAllWithoutPagination: () => {
    return api.get(`${API_ENDPOINTS.employees.base}/all`);
  },

  getById: (id) => {
    return api.get(API_ENDPOINTS.employees.byId(id));
  },

  getByDepartment: (departmentId) => {
    return api.get(`${API_ENDPOINTS.employees.base}/department/${departmentId}`);
  },

  create: (data) => {
    return api.post(API_ENDPOINTS.employees.base, data);
  },

  update: (id, data) => {
    return api.put(API_ENDPOINTS.employees.byId(id), data);
  },

  delete: (id) => {
    return api.delete(API_ENDPOINTS.employees.byId(id));
  },
};
```

```
// Роути системи управління конвеєрною лінією
export const productionLines = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
    return api
      .get(API_ENDPOINTS.productionLines.base, {
        params: {
          page,
          ...(search && { search }),
          ...(sortField && { sortField, sortOrder }),
          ...otherParams,
        },
      })
      .then((response) => {
        return response;
      });
  },

  getAllWithoutPagination: () => {
    return api.get(`${API_ENDPOINTS.productionLines.base}/all`);
  },

  getById: (id) => {
    return api.get(API_ENDPOINTS.productionLines.byId(id));
  },

  create: (data) => {
    return api.post(API_ENDPOINTS.productionLines.base, data);
  },

  update: (id, data) => {
    return api.put(API_ENDPOINTS.productionLines.byId(id), data);
  },

  delete: (id) => {
    return api.delete(API_ENDPOINTS.productionLines.byId(id));
  },
};

export const technologicalConditions = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
    return api
      .get(API_ENDPOINTS.technologicalConditions.base, {
        params: {
          page,
          ...(search && { search }),
          ...(sortField && { sortField, sortOrder }),
          ...otherParams,
        },
      })
      .then((response) => {
        return response;
      });
  },

  getAllWithoutPagination: () => {
    return api.get(`${API_ENDPOINTS.technologicalConditions.base}/all`);
  },
```

```
  getById: (id) => {
    return api.get(API_ENDPOINTS.technologicalConditions.byId(id));
  },

  create: (data) => {
    return api.post(API_ENDPOINTS.technologicalConditions.base, data);
  },

  update: (id, data) => {
    return api.put(API_ENDPOINTS.technologicalConditions.byId(id), data);
  },

  delete: (id) => {
    return api.delete(API_ENDPOINTS.technologicalConditions.byId(id));
  },
};

export const processParameters = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
    return api
      .get(API_ENDPOINTS.processParameters.base, {
        params: {
          page,
          ...(search && { search }),
          ...(sortField && { sortField, sortOrder }),
          ...otherParams,
        },
      })
      .then((response) => {
        return response;
      });
  },

  getAllWithoutPagination: () => {
    return api.get(`${API_ENDPOINTS.processParameters.base}/all`);
  },

  getById: (id) => {
    return api.get(API_ENDPOINTS.processParameters.byId(id));
  },

  create: (data) => {
    return api.post(API_ENDPOINTS.processParameters.base, data);
  },

  update: (id, data) => {
    return api.put(API_ENDPOINTS.processParameters.byId(id), data);
  },

  delete: (id) => {
    return api.delete(API_ENDPOINTS.processParameters.byId(id));
  },
};

export const controlSystems = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
```

```
    return api
      .get(API_ENDPOINTS.controlSystems.base, {
        params: {
          page,
          ...(search && { search }),
          ...(sortField && { sortField, sortOrder }),
          ...otherParams,
        },
      })
      .then((response) => {
        return response;
      });
  },

  getAllWithoutPagination: () => {
    return api.get(`${API_ENDPOINTS.controlSystems.base}/all`);
  },

  getById: (id) => {
    return api.get(API_ENDPOINTS.controlSystems.byId(id));
  },

  create: (data) => {
    return api.post(API_ENDPOINTS.controlSystems.base, data);
  },

  update: (id, data) => {
    return api.put(API_ENDPOINTS.controlSystems.byId(id), data);
  },

  delete: (id) => {
    return api.delete(API_ENDPOINTS.controlSystems.byId(id));
  },
};

export const hmiInterfaces = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
    return api
      .get(API_ENDPOINTS.hmiInterfaces.base, {
        params: {
          page,
          ...(search && { search }),
          ...(sortField && { sortField, sortOrder }),
          ...otherParams,
        },
      })
      .then((response) => {
        return response;
      });
  },

  getAllWithoutPagination: () => {
    return api.get(`${API_ENDPOINTS.hmiInterfaces.base}/all`);
  },

  getById: (id) => {
    return api.get(API_ENDPOINTS.hmiInterfaces.byId(id));
```

```
  },

  create: (data) => {
    return api.post(API_ENDPOINTS.hmiInterfaces.base, data);
  },

  update: (id, data) => {
    return api.put(API_ENDPOINTS.hmiInterfaces.byId(id), data);
  },

  delete: (id) => {
    return api.delete(API_ENDPOINTS.hmiInterfaces.byId(id));
  },
};

export const controlPoints = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
    return api
      .get(API_ENDPOINTS.controlPoints.base, {
        params: {
          page,
          ...(search && { search }),
          ...(sortField && { sortField, sortOrder }),
          ...otherParams,
        },
      })
      .then((response) => {
        return response;
      });
  },

  getAllWithoutPagination: () => {
    return api.get(`${API_ENDPOINTS.controlPoints.base}/all`);
  },

  getById: (id) => {
    return api.get(API_ENDPOINTS.controlPoints.byId(id));
  },

  create: (data) => {
    return api.post(API_ENDPOINTS.controlPoints.base, data);
  },

  update: (id, data) => {
    return api.put(API_ENDPOINTS.controlPoints.byId(id), data);
  },

  delete: (id) => {
    return api.delete(API_ENDPOINTS.controlPoints.byId(id));
  },
};

// Роути обладнання
export const sensorTypes = {
  getAll: () => api.get(API_ENDPOINTS.sensorTypes.base),
  create: (data) => api.post(API_ENDPOINTS.sensorTypes.base, data),
  update: (id, data) => api.put(API_ENDPOINTS.sensorTypes.byId(id), data),
```

```
  delete: (id) => api.delete(API_ENDPOINTS.sensorTypes.byId(id)),
};

export const sensors = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
    return api
      .get(API_ENDPOINTS.sensors.base, {
        params: {
          page,
          ...(search && { search }),
          ...(sortField && { sortField, sortOrder }),
          ...otherParams,
        },
      })
      .then((response) => {
        return response;
      });
  },

  getAllWithoutPagination: () => {
    return api.get(`${API_ENDPOINTS.sensors.base}/all`);
  },

  getById: (id) => {
    return api.get(API_ENDPOINTS.sensors.byId(id));
  },

  create: (data) => {
    return api.post(API_ENDPOINTS.sensors.base, data);
  },

  update: (id, data) => {
    return api.put(API_ENDPOINTS.sensors.byId(id), data);
  },

  delete: (id) => {
    return api.delete(API_ENDPOINTS.sensors.byId(id));
  },
};

export const actuators = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", ...otherParams } = {}) => {
    return api
      .get(API_ENDPOINTS.actuators.base, {
        params: {
          page,
          ...(search && { search }),
          ...(sortField && { sortField, sortOrder }),
          ...otherParams,
        },
      })
      .then((response) => {
        return response;
      });
  },

  getAllWithoutPagination: () => {
```

```
    return api.get(`${API_ENDPOINTS.actuators.base}/all`);
  },

  getById: (id) => {
    return api.get(API_ENDPOINTS.actuators.byId(id));
  },

  create: (data) => {
    return api.post(API_ENDPOINTS.actuators.base, data);
  },

  update: (id, data) => {
    return api.put(API_ENDPOINTS.actuators.byId(id), data);
  },

  delete: (id) => {
    return api.delete(API_ENDPOINTS.actuators.byId(id));
  },
};

// Роути вимірювань
export const engineeringUnitTypes = {
  getAll: () => api.get(API_ENDPOINTS.engineeringUnitTypes.base),
  create: (data) => api.post(API_ENDPOINTS.engineeringUnitTypes.base, data),
  update: (id, data) => api.put(API_ENDPOINTS.engineeringUnitTypes.byId(id), data),
  delete: (id) => api.delete(API_ENDPOINTS.engineeringUnitTypes.byId(id)),
};

export const engineeringUnits = {
  getAll: ({ page = 1, search, sortField, sortOrder = "asc", isActive, ...otherParams } = {}) =>
    api.get(API_ENDPOINTS.engineeringUnits.base, {
      params: {
        page,
        ...(search && { search }),
        ...(sortField && { sortField, sortOrder }),
        ...(isActive !== undefined && { isActive: isActive.toString() }),
        ...otherParams,
      },
    }),

  getAllWithoutPagination: () =>
    api.get(API_ENDPOINTS.engineeringUnits.all).then((res) => res.data),

  create: (data) => {
    const requestData = {
      designation: data.designation,
      name: data.name,
      type_id: Number(data.type_id),
    };
    return api.post(API_ENDPOINTS.engineeringUnits.base, requestData);
  },

  update: (id, data) => {
    const requestData = {
      designation: data.designation,
      name: data.name,
      type_id: Number(data.type_id),
```

```
    };
    return api.put(API_ENDPOINTS.engineeringUnits.byId(id), requestData);
  },

  delete: (id) => api.delete(API_ENDPOINTS.engineeringUnits.byId(id)),
};

// Імпорт даних
export const importApi = {
  importData: (data) => api.post(API_ENDPOINTS.import.base, data),

  importTxt: (file, params, onProgress) => {
    const urlParams = new URLSearchParams();
    Object.entries(params).forEach(([key, value]) => {
      urlParams.append(key, value);
    });

    return new Promise((resolve, reject) => {
      const xhr = new XMLHttpRequest();
      const url = `${API_URL}${API_ENDPOINTS.import.txt}?${urlParams}`;
      xhr.open("POST", url);
      xhr.setRequestHeader("Content-Type", "application/octet-stream");
      xhr.withCredentials = true;
      xhr.responseType = "text";

      if (xhr.overrideMimeType) {
        xhr.overrideMimeType("text/plain; charset=utf-8");
      }

      const updates = [];
      let lastProcessedLength = 0;

      xhr.upload.onprogress = (progressEvent) => {
        if (onProgress && progressEvent.lengthComputable) {
          const percentCompleted = Math.round((progressEvent.loaded·100) / progressEvent.total);
          onProgress({
            progress: { stage: "upload", current: percentCompleted, total: 100 },
            status: "Завантаження TXT файлу на сервер...",
          });
        }
      };

      const processResponseData = () => {
        if (!xhr.responseText) {
          return;
        }

        const currentLength = xhr.responseText.length;
        if (currentLength > lastProcessedLength) {
          const newData = xhr.responseText.slice(lastProcessedLength);
          lastProcessedLength = currentLength;

          const lines = newData.split("\n").filter((line) => line.trim());
          lines.forEach((line) => {
            try {
              const update = JSON.parse(line);
              updates.push(update);
```

```
            if (onProgress) {
              onProgress(update);
            }
          } catch (e) {
            // Ігноруємо неповні JSON рядки
          }
        });
      }
    };

    xhr.onreadystatechange = () => {
      if (xhr.readyState === 3 || xhr.readyState === 4) {
        processResponseData();
      }
    };

    xhr.onprogress = () => {
      processResponseData();
    };

    xhr.onload = () => {
      processResponseData();
      if (xhr.status === 200) {
        resolve({ data: updates });
      } else {
        const error = new Error(`Помилка імпорту TXT! статус: ${xhr.status}`);
        if (onProgress) {
          onProgress({ error: error.message });
        }
        reject(error);
      }
    };

    xhr.onerror = () => {
      const error = new Error("Мережева помилка імпорту TXT");
      if (onProgress) {
        onProgress({ error: error.message });
      }
      reject(error);
    };

    xhr.send(file);
  });
},

importExcel: (file, params, onProgress) => {
  const urlParams = new URLSearchParams();

  // Обов'язкові параметри
  const requiredParams = ["positionRange", "valueRange", "dateRange", "timeRange"];
  const missingParams = requiredParams.filter((param) => !params[param]);

  if (missingParams.length > 0) {
    return Promise.reject({
      type: "MISSING_PARAMETERS",
      params: missingParams,
    });
```

```
    }

    // Дозволені параметри
    const allowedParams = [
      "positionRange",
      "valueRange",
      "dateRange",
      "timeRange",
      "modeRange",
      "forceOverwrite",
    ];

    // Обробляємо параметри
    Object.entries(params).forEach(([key, value]) => {
      if (allowedParams.includes(key) && value !== undefined && value !== null && value !== "") {
        let processedValue = value;
        if (key.endsWith("Range")) {
          processedValue = value.replace(/\s/g, "").toUpperCase();
        }
        urlParams.append(key, processedValue);
      }
    });

    return new Promise((resolve, reject) => {
      const xhr = new XMLHttpRequest();
      const url = `${API_URL}${API_ENDPOINTS.import.excel}?${urlParams}`;

      xhr.open("POST", url, true);
      xhr.setRequestHeader("Content-Type", "application/octet-stream");
      xhr.withCredentials = true;
      xhr.responseType = "text";

      let buffer = "";
      let finalResult = null;
      const allErrors = [];

      const normalizeError = (error) => {
        const getErrorSeverity = (errorType) => {
          const criticalErrors = [
            "MISSING_PARAMETER",
            "FILE_EMPTY",
            "FILE_TOO_LARGE",
            "PARSE_ERROR",
            "DATABASE_ERROR",
          ];
          const warningErrors = ["VALUE_BELOW_MIN", "VALUE_ABOVE_MAX",
"EMPTY_VALUE"];

          if (criticalErrors.includes(errorType)) return "error";
          if (warningErrors.includes(errorType)) return "warning";
          return "error";
        };

        const normalized = {
          type: error.type || "UNKNOWN_ERROR",
          count: error.count || 1,
          severity: error.severity || getErrorSeverity(error.type || "UNKNOWN_ERROR"),
```

```
    timestamp: error.timestamp || new Date().toISOString(),
    row: error.row,
    column: error.column,
    cell: error.cell,
    value: error.value,
    ...Object.keys(error).reduce((acc, key) => {
      if (!["type", "count", "severity", "timestamp", "row", "column", "cell", "value"].includes(key)) {
        acc[key] = error[key];
      }
      return acc;
    }, {}),
  };

  Object.keys(normalized).forEach((key) => {
    if (normalized[key] === undefined) {
      delete normalized[key];
    }
  });

  return normalized;
};

xhr.upload.onprogress = (progressEvent) => {
  if (onProgress && progressEvent.lengthComputable) {
    const progress = Math.round((progressEvent.loaded·100) / progressEvent.total);
    onProgress({
      stage: "upload",
      progress: progress,
      isIntermediate: true,
    });
  }
};

xhr.onprogress = () => {
  try {
    const newData = xhr.responseText.slice(buffer.length);
    buffer = xhr.responseText;

    const lines = newData
      .split("\n")
      .filter((line) => line.startsWith("data: ") && line.length > 6);

    for (const line of lines) {
      try {
        const update = JSON.parse(line.substring(6).trim());

        if (update.isFinal) {
          finalResult = update;
        }

        if (update.errorDetails && Array.isArray(update.errorDetails)) {
          update.errorDetails.forEach((error) => {
            if (!error.type) {
              return;
            }

            const normalizedError = normalizeError(error);
```

```
          const errorKey = JSON.stringify({
            type: error.type,
            cell: normalizedError.cell,
            value: normalizedError.value,
          });

          const existingIndex = allErrors.findIndex(
            (e) =>
              JSON.stringify({
                type: e.type,
                cell: e.cell,
                value: e.value,
              }) === errorKey
          );

          if (existingIndex === -1) {
            allErrors.push(normalizedError);
          } else {
            allErrors[existingIndex].count = normalizedError.count;
          }
        });
      }

      if (onProgress) {
        onProgress({
          ...update,
          stage: update.stage || "import",
          isIntermediate: !update.isFinal,
          errorDetails: allErrors,
        });
      }
    } catch (e) {
      const errorObj = {
        type: "SSE_PARSE_ERROR",
        message: e.message,
        timestamp: new Date().toISOString(),
        count: 1,
        severity: "error",
      };
      allErrors.push(errorObj);
    }
  }
 } catch (e) {
  const errorObj = {
    type: "SSE_PROCESS_ERROR",
    message: e.message,
    timestamp: new Date().toISOString(),
    count: 1,
    severity: "error",
  };
  allErrors.push(errorObj);
 }
};

xhr.onload = () => {
 const totalErrorCount = allErrors.reduce((sum, error) => sum + (error.count || 1), 0);
 const uniqueErrorTypes = [...new Set(allErrors.map((e) => e.type))].length;
```

```
if (xhr.status === 200) {
  if (finalResult?.fatal) {
    if (onProgress) {
      onProgress({
        stage: "error",
        statusKey: finalResult.statusKey || "import_failed",
        errors: [...new Set(allErrors.map((e) => e.type))],
        errorDetails: allErrors,
        fatal: true,
        isFinal: true,
      });
    }

    const rejectError = {
      type: "IMPORT_FAILED",
      message: finalResult.statusKey || "Імпорт не вдався",
      details: finalResult,
      errors: allErrors,
      totalErrors: totalErrorCount,
      uniqueErrorTypes: uniqueErrorTypes,
      httpStatus: xhr.status,
    };

    reject(rejectError);
  } else {
    if (onProgress) {
      onProgress({
        stage: "complete",
        statusKey: finalResult?.statusKey || "import_successful",
        summary: finalResult?.summary,
        errors: allErrors,
        errorDetails: allErrors,
        isFinal: true,
      });
    }

    const resolveResult = {
      ...finalResult,
      errors: allErrors,
      totalErrors: totalErrorCount,
      uniqueErrorTypes: uniqueErrorTypes,
      warnings: allErrors.filter((e) => e.severity === "warning"),
    };

    resolve(resolveResult);
  }
} else {
  if (finalResult?.fatal) {
    if (onProgress) {
      onProgress({
        stage: "error",
        statusKey: finalResult.statusKey || "import_failed",
        errors: [...new Set(allErrors.map((e) => e.type))],
        errorDetails: allErrors,
        fatal: true,
        isFinal: true,
```

```
        });
      }

      const rejectError = {
        type: "IMPORT_FAILED",
        message: finalResult.statusKey || "Імпорт не вдався",
        details: finalResult,
        errors: allErrors,
        totalErrors: totalErrorCount,
        uniqueErrorTypes: uniqueErrorTypes,
        httpStatus: xhr.status,
      };

      reject(rejectError);
    } else {
      if (onProgress) {
        onProgress({
          stage: "error",
          statusKey: "server_error",
          errors: ["SERVER_ERROR"],
          errorDetails: allErrors,
          fatal: true,
          isFinal: true,
        });
      }

      const rejectError = {
        type: "SERVER_ERROR",
        status: xhr.status,
        response: xhr.responseText,
        errors: allErrors,
        totalErrors: totalErrorCount,
        uniqueErrorTypes: uniqueErrorTypes,
      };

      reject(rejectError);
    }
  }
};

xhr.onerror = () => {
  const totalErrorCount = allErrors.reduce((sum, error) => sum + (error.count || 1), 0);
  const uniqueErrorTypes = [...new Set(allErrors.map((e) => e.type))].length;

  if (onProgress) {
    onProgress({
      stage: "error",
      statusKey: "network_error",
      errors: ["NETWORK_ERROR"],
      errorDetails: allErrors,
      fatal: true,
      isFinal: true,
    });
  }

  const rejectError = {
    type: "NETWORK_ERROR",
```

```
    errors: allErrors,
    totalErrors: totalErrorCount,
    uniqueErrorTypes: uniqueErrorTypes,
   };

   reject(rejectError);
  };

  xhr.onabort = () => {
   const totalErrorCount = allErrors.reduce((sum, error) => sum + (error.count || 1), 0);
   const uniqueErrorTypes = [...new Set(allErrors.map((e) => e.type))].length;

   if (onProgress) {
    onProgress({
     stage: "error",
     statusKey: "aborted",
     errors: ["ABORTED"],
     errorDetails: allErrors,
     fatal: true,
     isFinal: true,
    });
   }

   const rejectError = {
    type: "ABORTED",
    errors: allErrors,
    totalErrors: totalErrorCount,
    uniqueErrorTypes: uniqueErrorTypes,
   };

   reject(rejectError);
  };

  try {
   xhr.send(file);
  } catch (e) {
   const totalErrorCount = allErrors.reduce((sum, error) => sum + (error.count || 1), 0);
   const uniqueErrorTypes = [...new Set(allErrors.map((e) => e.type))].length;

   if (onProgress) {
    onProgress({
     stage: "error",
     statusKey: "send_error",
     errors: ["SEND_ERROR"],
     errorDetails: allErrors,
     fatal: true,
     isFinal: true,
    });
   }

   const rejectError = {
    type: "SEND_ERROR",
    message: e.message,
    errors: allErrors,
    totalErrors: totalErrorCount,
    uniqueErrorTypes: uniqueErrorTypes,
   };
```

```
        reject(rejectError);
      }
    });
  },

  importExcelForce: (file, mapping, onProgress) => {
    return importApi.importExcel(file, { ...mapping, forceOverwrite: true }, onProgress);
  },
};

export default {
  auth,
  users,
  preferences,
  license,
  importTemplates,
  companies,
  departments,
  employeePositions,
  employeeRoles,
  employees,
  productionLines,
  technologicalConditions,
  processParameters,
  controlSystems,
  hmiInterfaces,
  controlPoints,
  sensorTypes,
  sensors,
  actuators,
  engineeringUnitTypes,
  engineeringUnits,
  import: importApi,
};
```

ДОДАТОК Б

Апробація наукових результатів досліджень

GLOBAL TRENDS IN SCIENCE AND EDUCATION

Proceedings of X International Scientific and Practical Conference

Kyiv, Ukraine

20-22 October 2025

Kyiv, Ukraine

2025

ПЕРЕВАГИ ПРОТОКОЛУ TCP/IP ДЛЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ УПРАВЛІННЯ КОНВЕЄРНОЮ ЛІНІЄЮ ІЗ ЗАСТОСУВАННЯМ ЗАСОБІВ ЛЮДИНО-МАШИННОГО ІНТЕРФЕЙСУ

Глєбов Євген Олександрович,
студент
Харківський національний університет радіоелектроніки
м. Харків, Україна

Вступ.

Сучасні виробничі лінії, особливо конвеєрні системи, критично залежать від високої швидкості та надійності Автоматизованих Систем Управління (АСУ). Для оперативного моніторингу та керування через людино-машинний інтерфейс (HMI) необхідний комунікаційний протокол із максимально низькою затримкою. Протокол TCP/IP (Transmission Control Protocol/Internet Protocol) є основою промислового Ethernet, чиї архітектурні переваги роблять його ідеальним вибором для високодинамічних АСУ.

Мета та завдання роботи.

– метою роботи є обґрунтування переваг протоколу TCP/IP як транспортного рівня для створення високоінтегрованої та швидкісної системи управління конвеєрною лінією із застосуванням HMI, порівняно зі спеціалізованими протоколами (наприклад, DNP3). Для цього необхідно проаналізувати динамічні та інтеграційні характеристики TCP/IP.

Аналіз архітектури та функціональності TCP/IP.

Протокол TCP/IP є стеком, що забезпечує надійну (TCP) та маршрутизовану (IP) доставку даних, слугуючи транспортним рівнем для таких промислових протоколів, як Modbus TCP та EtherNet/IP. Це лежить в основі конвергенції операційних (OT) та інформаційних (IT) технологій [2].

Ключові архітектурні переваги:

– висока пропускна здатність: Використання фізичного рівня Ethernet (100 Мбіт/с – 1 Гбіт/с) дозволяє передавати об'ємні дані HMI (графіки,

відеопотоки) практично миттєво;

– масштабованість: Система IP-адресації забезпечує гнучку та необмежену топологію мережі, що критично для довгих конвеєрних ліній.

Порівняння з DNP3: Протокол DNP3 розроблений для низькошвидкісних каналів зв'язку критичної інфраструктури, де пріоритетом є передача даних за подіями. Для динамічної конвеєрної лінії, де HMI потребує постійного швидкого оновлення візуальних даних, швидкість і нативна підтримка TCP/IP є вирішальними.

Динамічні переваги та інтеграція системи.

Перевага TCP/IP в управлінні конвеєром визначається його здатністю мінімізувати затримку (Latency). Час передачі даних обернено пропорційний пропускній здатності.

Динамічна перевага: В той час як низькошвидкісні протоколи, як DNP3, можуть витрачати сотні мілісекунд на передачу типового пакета, мережа TCP/IP (на швидкості 100 Мбіт/с) забезпечує передачу за частки мілісекунди [1]. Таким чином, TCP/IP забезпечує скорочення часу передачі даних у тисячі разів, що є критично важливим для оперативного керування в реальному часі. Крім того, висока пропускна здатність мінімізує час очікування в черзі, забезпечуючи необхідну детермінованість системи навіть при високому трафіку [3].

Інтеграційна функціональність: Використання TCP/IP є основою для вертикальної та горизонтальної інтеграції. Протокол забезпечує високошвидкісний обмін даними між різними ПЛК для точної синхронізації конвеєрних секцій (горизонтальна інтеграція). Крім того, TCP/IP є нативним для корпоративних IT-систем, дозволяючи ПЛК безпосередньо передавати виробничі дані у системи MES та ERP (вертикальна інтеграція) без потреби у складних шлюзах.

Висновки.

Протокол TCP/IP є ефективним інструментом для побудови високодинамічних та інтегрованих систем керування конвеєрною лінією із

застосуванням сучасних засобів HMI. Його висока пропускна здатність, нативна сумісність із широким спектром обладнання та підтримка вертикальної інтеграції роблять його оптимальним вибором для більшості промислових застосувань.

Таким чином, використання TCP/IP забезпечує мінімізацію затримки, високу швидкість обміну даними, значне зниження витрат на впровадження завдяки використанню стандартного обладнання та створює архітектурну основу для Індустрії 4.0, що є критично важливим для сучасної промисловості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. J. W. Zhang, Z. Y. Wang and Y. L. Wang (2023), "Research on Real-Time Data Transmission System for Industrial Internet of Things Based on TCP/IP," *2023 5th International Conference on Computing and Pattern Recognition (ICCPR)*, P. 136–140.

2. K. A. J. Al-Rubaie, Z. A. Abdo, M. I. Al-Hamadi and A. T. Al-Dujaili (2020), "A Survey on Industrial Internet of Things and Its Application in Industrial Automation," *2020 International Conference on Computer Science and Software Engineering (CSSE)*, P. 200–205.

3. M. T. F. El-Nainay, M. Y. Al-Aref and A. E. M. Al-Mously (2020), "Analysis of SCADA Protocols (DNP3 and IEC 60870-5-104) in Smart Grid Environment," *2020 2nd International Conference on Electrical, Control and Instrumentation Engineering (ICECIE)*, P. 1–6.

4. M. H. Mahbub, M. H. Zaman, J. Alam, M. N. Al-Arefin and S. S. A. Khan (2022), "A Survey on Security Challenges and Countermeasures for Industrial Control Systems (ICS)," *2022 International Conference on Innovation in Engineering and Technology (ICIET)*, P. 1–7.

ДОДАТОК В

Демонстраційний матеріал

№ докумен-та	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК. 464430.018 ПЗ	Пояснювальна записка	А4, 99 с.
		Додаткові матеріали	
2		Демонстраційний матеріал у	А4, 11 с.
		вигляді презентації	

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК.464430.018 ПЗ			
Розроб.		Глєбов Є.О.			Розроблення автоматизованої системи управління конвеєрною лінією із застосуванням засобів людино-машинного інтерфейсу з урахуванням технологічних умов виробництва	Літера	Аркуш	Аркушів
Перевір.		Демська Н.П.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Стародубцев М.Г.						
Затв.		Невлюдов І.Ш.						