Novi - Operational Documentation

Novi
Smart Commerce Suite

Generated on: 28/06/2025

Novi - Operational Documentation

Ø=Ý' <strong>Operations, Maintenance &amp; Security Guide</strong>

<strong>Table of Contents</strong>

Ø=ÜÊ <strong>System Monitoring</strong>

<strong>Real-time Monitoring</strong>

<h4>Memory Usage Monitoring</h4>

```
const memoryThreshold = 500 * 1024 * 1024; // 500MB

const checkInterval = 5 * 60 * 1000; // 5 minutes

// Monitor memory usage

setInterval(() =&gt; {

    const usage = process.memoryUsage();

    logger.info(&#39;Memory usage:&#39;, {

        heapUsed: usage.heapUsed,

        heapTotal: usage.heapTotal,
        external: usage.external,
        rss: usage.rss
    });
    if (usage.heapUsed &gt; memoryThreshold) {
        logger.warn(&#39;Memory usage exceeded threshold&#39;);
        performMemoryCleanup();
    }
}, checkInterval);
```

<h4>WhatsApp Connection Monitoring</h4>

```
client.on(&#39;disconnected&#39;, (reason) =&gt; {
    logger.error(&#39;WhatsApp disconnected:&#39;, reason);
    // Attempt reconnection
    setTimeout(() =&gt; {
        initializeWhatsApp();
    }, 30000);
});
client.on(&#39;auth_failure&#39;, (message) =&gt; {
    logger.error(&#39;WhatsApp authentication failed:&#39;, message);
    // Notify administrators
    notifyAdmins(&#39;WhatsApp authentication failed&#39;);
});
```

<h4>Database Connection Monitoring</h4>

```
knex.raw(&#39;SELECT 1&#39;)
    .then(() =&gt; {
        logger.info(&#39;Database connection healthy&#39;);
    })
    .catch((error) =&gt; {
        logger.error(&#39;Database connection failed:&#39;, error);
        // Attempt reconnection
        reconnectDatabase();
    });
```

<strong>Performance Metrics</strong>

<h4>Key Performance Indicators (KPIs)</h4>

- <strong>Response Time</strong>: Average API response time
- <strong>Throughput</strong>: Orders processed per hour
- <strong>Uptime</strong>: System availability percentage
- <strong>Error Rate</strong>: Percentage of failed requests
- <strong>Memory Usage</strong>: Current memory consumption
- <strong>Database Performance</strong>: Query execution times

<h4>Monitoring Dashboard</h4>

```
app.get(&#39;/health&#39;, (req, res) =&gt; {
  const health = {
    status: &#39;healthy&#39;,
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    memory: process.memoryUsage(),
    database: &#39;connected&#39;,
    whatsapp: client.isConnected ? &#39;connected&#39; : &#39;disconnected&#39;
  };
  res.json(health);
});
```

<strong>Alert System</strong>

<h4>Alert Configuration</h4>

```
const alerts = {
  memoryUsage: 80, // Percentage
  errorRate: 5,    // Percentage
  responseTime: 5000, // Milliseconds
  downtime: 300    // Seconds
};
// Send alerts
function sendAlert(type, message, severity) {
  const alert = {
    type,
        message,
        severity,
        timestamp: new Date().toISOString(),
        system: &#39;Novi Platform&#39;
    };
    // Send to monitoring system
    logger.error(&#39;ALERT:&#39;, alert);
    // Notify administrators
    if (severity === &#39;critical&#39;) {
        notifyAdmins(alert);
    }
}
```

Ø=Ý' <strong>Maintenance Procedures</strong>

<strong>Daily Maintenance</strong>

<h4>System Health Check</h4>

<li><p><strong>Check Application Status</strong></p>

```
pm2 status
# Check memory usage
pm2 monit
```

<li><p><strong>Database Health Check</strong></p>

```
SELECT count(*) FROM pg_stat_activity;
-- Check for long-running queries
SELECT pid, now() - pg_stat_activity.query_start AS duration, query
FROM pg_stat_activity
WHERE (now() - pg_stat_activity.query_start) &gt; interval &#39;5 minutes&#39;;
```

<li><p><strong>WhatsApp Connection Check</strong></p>

```
if (!client.isConnected) {
    logger.warn(&#39;WhatsApp not connected, attempting reconnection&#39;);
    await initializeWhatsApp();
}
```

<h4>Log Review</h4>

```
tail -f logs/combined.log

# Check error logs

tail -f logs/error.log

# Check WhatsApp logs

tail -f logs/whatsapp.log
```

<strong>Weekly Maintenance</strong>

<h4>Database Maintenance</h4>

```
ANALYZE;

-- Vacuum tables

VACUUM ANALYZE;

-- Check for table bloat

SELECT schemaname, tablename, n_tup_ins, n_tup_upd, n_tup_del, n_live_tup, n_dead_tup

FROM pg_stat_user_tables

WHERE n_dead_tup &gt; 0;
```

<h4>Cache Cleanup</h4>

```
function cleanupCache() {

  const now = Date.now();

  const ttl = 24 * 60 * 60 * 1000; // 24 hours

  for (const [key, value] of cache.entries()) {

    if (now - value.timestamp &gt; ttl) {

      cache.delete(key);

    }

  }

  logger.info(&#39;Cache cleanup completed&#39;);

}
```

<h4>File System Cleanup</h4>

```
find logs/ -name &quot;*.log&quot; -mtime +7 -delete

# Clean old session files

find sessions/ -name &quot;*.json&quot; -mtime +30 -delete
```

```
# Clean temporary files

find temp/ -name &quot;*&quot; -mtime +1 -delete
```

<strong>Monthly Maintenance</strong>

<h4>Security Audit</h4>

<li><p><strong>Review User Access</strong></p>

```
SELECT username, last_login, is_active

FROM users

WHERE last_login &lt; NOW() - INTERVAL &#39;30 days&#39;;
```

<li><p><strong>Review Admin Access</strong></p>

```
SELECT username, action, created_at

FROM admin_logs

WHERE created_at &gt; NOW() - INTERVAL &#39;30 days&#39;;
```

<li><p><strong>Update Dependencies</strong></p>

```
npm outdated

# Update packages

npm update

# Check for security vulnerabilities

npm audit
```

<h4>Performance Optimization</h4>

```
async function optimizeQueries() {

  // Update table statistics

  await knex.raw(&#39;ANALYZE&#39;);

  // Rebuild indexes if needed

  await knex.raw(&#39;REINDEX DATABASE novi_platform&#39;);

  logger.info(&#39;Database optimization completed&#39;);

}
```

Ø=Ý <strong>Security Protocols</strong>

<strong>Access Control</strong>

<h4>User Authentication</h4>

```
app.use(session({
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: false,
    cookie: {
        secure: process.env.NODE_ENV === &#39;production&#39;,
        httpOnly: true,
        maxAge: 24 * 60 * 60 * 1000, // 24 hours
        sameSite: &#39;strict&#39;
    }
}));
// Password requirements
const passwordRequirements = {
    minLength: 8,
    requireUppercase: true,
    requireLowercase: true,
    requireNumbers: true,
    requireSpecialChars: true
};
```

<h4>Role-based Access Control</h4>

```
const roles = {
    USER: &#39;user&#39;,
    ADMIN: &#39;admin&#39;,
    SUPER_ADMIN: &#39;super_admin&#39;
};
// Permission matrix
const permissions = {
    [roles.USER]: [&#39;read_own_orders&#39;, &#39;update_own_orders&#39;],
    [roles.ADMIN]: [&#39;read_all_orders&#39;, &#39;update_all_orders&#39;, &#39;manage_users&#39;],
    [roles.SUPER_ADMIN]: [&#39;all_permissions&#39;]
};
// Check permissions
```

```
function hasPermission(userRole, permission) {
    return permissions[userRole]?.includes(permission) ||
           permissions[userRole]?.includes(&#39;all_permissions&#39;);
}
```

<strong>Data Protection</strong>

<h4>Encryption</h4>

```
const crypto = require(&#39;crypto&#39;);
function encryptData(data, key) {
    const cipher = crypto.createCipher(&#39;aes-256-cbc&#39;, key);
    let encrypted = cipher.update(data, &#39;utf8&#39;, &#39;hex&#39;);
    encrypted += cipher.final(&#39;hex&#39;);
    return encrypted;
}
function decryptData(encryptedData, key) {
    const decipher = crypto.createDecipher(&#39;aes-256-cbc&#39;, key);
    let decrypted = decipher.update(encryptedData, &#39;hex&#39;, &#39;utf8&#39;);
    decrypted += decipher.final(&#39;utf8&#39;);
    return decrypted;
}
```

<h4>Input Validation</h4>

```
const sanitize = require(&#39;sanitize-html&#39;);
function sanitizeInput(input) {
    return sanitize(input, {
        allowedTags: [],
        allowedAttributes: {}
    });
}
// Validate data types
function validateOrderData(data) {
    const schema = Joi.object({
        customerName: Joi.string().max(255).required(),
        customerPhone: Joi.string().pattern(/^\+?[\d\s-]+$/),
        items: Joi.string().max(1000).required(),
        totalAmount: Joi.number().positive().optional()
    });
    return schema.validate(data);
}
```

<strong>Network Security</strong>

<h4>HTTPS Configuration</h4>

```
if (process.env.NODE_ENV === &#39;production&#39;) {
    app.use((req, res, next) =&gt; {
        if (req.header(&#39;x-forwarded-proto&#39;) !== &#39;https&#39;) {
            res.redirect(`https://${req.header(&#39;host&#39;)}${req.url}`);
        } else {
            next();
        }
    });
}
```

<h4>Rate Limiting</h4>

```
// API rate limiting
const apiLimiter = rateLimit({
    windowMs: 15 * 60 * 1000, // 15 minutes
    max: 100, // limit each IP to 100 requests per windowMs
    message: &#39;Too many requests from this IP&#39;
});
app.use(&#39;/api/&#39;, apiLimiter);
// Login rate limiting
const loginLimiter = rateLimit({
    windowMs: 15 * 60 * 1000, // 15 minutes
    max: 5, // limit each IP to 5 login attempts per windowMs
    message: &#39;Too many login attempts&#39;
});
```

```
app.use(&#39;/auth/login&#39;, loginLimiter);
```

Ø=Ü¾ <strong>Backup &amp; Recovery</strong>

<strong>Database Backup</strong>

<h4>Automated Backup Script</h4>

```
# backup.sh

# Set variables

DB_NAME=&quot;novi_platform&quot;

BACKUP_DIR=&quot;/backups&quot;

DATE=$(date +%Y%m%d_%H%M%S)

BACKUP_FILE=&quot;$BACKUP_DIR/backup_$DATE.sql&quot;

# Create backup

pg_dump $DB_NAME &gt; $BACKUP_FILE

# Compress backup

gzip $BACKUP_FILE

# Remove old backups (keep last 7 days)

find $BACKUP_DIR -name &quot;backup_*.sql.gz&quot; -mtime +7 -delete

# Log backup

echo &quot;Backup completed: $BACKUP_FILE.gz&quot; &gt;&gt; /var/log/backup.log
```

<h4>Backup Schedule</h4>

```
# Daily backup at 2 AM

0 2 * * * /path/to/backup.sh

# Weekly full backup

0 2 * * 0 /path/to/full_backup.sh
```

<strong>File System Backup</strong>

<h4>Application Files</h4>

```
# app_backup.sh

APP_DIR=&quot;/app&quot;

BACKUP_DIR=&quot;/backups/app&quot;

DATE=$(date +%Y%m%d_%H%M%S)
```

```
# Create backup directory
mkdir -p $BACKUP_DIR
# Backup application files
tar -czf $BACKUP_DIR/app_$DATE.tar.gz $APP_DIR
# Remove old backups (keep last 30 days)
find $BACKUP_DIR -name &quot;app_*.tar.gz&quot; -mtime +30 -delete
```

<strong>Recovery Procedures</strong>

<h4>Database Recovery</h4>

```
# restore.sh
DB_NAME=&quot;novi_platform&quot;
BACKUP_FILE=&quot;$1&quot;
if [ -z &quot;$BACKUP_FILE&quot; ]; then
    echo &quot;Usage: $0 &lt;backup_file&gt;&quot;
    exit 1
fi
# Stop application
pm2 stop novi-platform
# Drop and recreate database
dropdb $DB_NAME
createdb $DB_NAME
# Restore from backup
gunzip -c $BACKUP_FILE | psql $DB_NAME
# Start application
pm2 start novi-platform
echo &quot;Database restored successfully&quot;
```

<h4>Application Recovery</h4>

```
# app_restore.sh
APP_DIR=&quot;/app&quot;
BACKUP_FILE=&quot;$1&quot;
if [ -z &quot;$BACKUP_FILE&quot; ]; then
    echo &quot;Usage: $0 &lt;backup_file&gt;&quot;
    exit 1
fi
# Stop application
pm2 stop novi-platform
# Backup current files
mv $APP_DIR $APP_DIR.bak
# Extract backup
tar -xzf $BACKUP_FILE -C /
# Install dependencies
cd $APP_DIR
npm install
# Start application
pm2 start novi-platform
echo &quot;Application restored successfully&quot;
```

&¡ <strong>Performance Optimization</strong>

<strong>Memory Optimization</strong>

<h4>Memory Cleanup</h4>

```
function performMemoryCleanup() {
    // Clear caches
    cache.clear();
    // Clear message history
    messageHistory.splice(0, messageHistory.length - 100);
    // Force garbage collection
    if (global.gc) {
        global.gc();
    }
    logger.info(&#39;Memory cleanup completed&#39;);
}
// Schedule cleanup
setInterval(performMemoryCleanup, 30 * 60 * 1000); // Every 30 minutes
```

<h4>WhatsApp Client Optimization</h4>

```
const puppeteerOptions = {
    headless: true,
    args: [
        &#39;--no-sandbox&#39;,
        &#39;--disable-setuid-sandbox&#39;,
        &#39;--disable-dev-shm-usage&#39;,
        &#39;--disable-accelerated-2d-canvas&#39;,
        &#39;--no-first-run&#39;,
        &#39;--no-zygote&#39;,
        &#39;--disable-gpu&#39;,
        &#39;--disable-background-timer-throttling&#39;,
        &#39;--disable-backgrounding-occluded-windows&#39;,
        &#39;--disable-renderer-backgrounding&#39;,
        &#39;--memory-pressure-off&#39;,
        &#39;--max_old_space_size=512&#39;
    ]
};
```

<strong>Database Optimization</strong>

<h4>Query Optimization</h4>

```
const optimizedQuery = `
    SELECT o.*, b.name as business_name
    FROM orders o
    INNER JOIN businesses b ON o.business_id = b.id
    WHERE o.business_id = ?
    AND o.created_at &gt;= ?
    ORDER BY o.created_at DESC
    LIMIT ?
`;
// Use connection pooling
const knexConfig = {
    client: &#39;postgresql&#39;,
    connection: process.env.DATABASE_URL,
    pool: {
      min: 2,
      max: 10,
      acquireTimeoutMillis: 30000,
      createTimeoutMillis: 30000,
      destroyTimeoutMillis: 5000,
      idleTimeoutMillis: 30000,
      reapIntervalMillis: 1000,
      createRetryIntervalMillis: 100
    }
};
```

<h4>Caching Strategy</h4>

```
const Redis = require(&#39;ioredis&#39;);
const redis = new Redis(process.env.REDIS_URL);
class CacheService {
  async get(key) {
    try {
      const value = await redis.get(key);
      return value ? JSON.parse(value) : null;
    } catch (error) {
      logger.error(&#39;Cache get error:&#39;, error);
      return null;
    }
  }
  async set(key, value, ttl = 300) {
    try {
      await redis.setex(key, ttl, JSON.stringify(value));
    } catch (error) {
      logger.error(&#39;Cache set error:&#39;, error);
    }
    }
    async del(key) {
        try {
            await redis.del(key);
        } catch (error) {
            logger.error(&#39;Cache delete error:&#39;, error);
        }
    }
}
```

Ø=Ý' <strong>Troubleshooting Guide</strong>

<strong>Common Issues</strong>

<h4>WhatsApp Connection Issues</h4>

```
async function troubleshootWhatsApp() {
    // Check if client is connected
    if (!client.isConnected) {
        logger.error(&#39;WhatsApp not connected&#39;);
        // Check session files
        const sessionFiles = fs.readdirSync(&#39;./sessions&#39;);
        if (sessionFiles.length === 0) {
            logger.error(&#39;No session files found&#39;);
            return &#39;No session files&#39;;
        }
        // Try to reconnect
        try {
            await client.initialize();
            logger.info(&#39;WhatsApp reconnected successfully&#39;);
            return &#39;Reconnected&#39;;
        } catch (error) {
            logger.error(&#39;Failed to reconnect:&#39;, error);
            return &#39;Reconnection failed&#39;;
        }
    }
    return &#39;Connected&#39;;
}
```

<h4>Database Connection Issues</h4>

```
async function troubleshootDatabase() {
    try {
        // Test connection
        await knex.raw(&#39;SELECT 1&#39;);
        logger.info(&#39;Database connection healthy&#39;);
        return &#39;Connected&#39;;
    } catch (error) {
        logger.error(&#39;Database connection failed:&#39;, error);
        // Check connection string
        if (!process.env.DATABASE_URL) {
            logger.error(&#39;DATABASE_URL not set&#39;);
            return &#39;No connection string&#39;;
        }
        // Try to reconnect
        try {
            await knex.destroy();
            await knex.initialize();
            logger.info(&#39;Database reconnected&#39;);
            return &#39;Reconnected&#39;;
        } catch (reconnectError) {
            logger.error(&#39;Database reconnection failed:&#39;, reconnectError);
            return &#39;Reconnection failed&#39;;
        }
    }
}
```

<h4>Memory Issues</h4>

```
function troubleshootMemory() {
    const usage = process.memoryUsage();
    logger.info(&#39;Memory usage:&#39;, {
        heapUsed: `${Math.round(usage.heapUsed / 1024 / 1024)}MB`,
        heapTotal: `${Math.round(usage.heapTotal / 1024 / 1024)}MB`,
        external: `${Math.round(usage.external / 1024 / 1024)}MB`,
        rss: `${Math.round(usage.rss / 1024 / 1024)}MB`
    });
    // Check for memory leaks
    if (usage.heapUsed &gt; 500 * 1024 * 1024) { // 500MB
        logger.warn(&#39;High memory usage detected&#39;);
        performMemoryCleanup();
        // Restart if still high
        if (usage.heapUsed &gt; 800 * 1024 * 1024) { // 800MB
            logger.error(&#39;Critical memory usage, restarting application&#39;);
            process.exit(1);
        }
    }
}
```

<strong>Performance Issues</strong>

<h4>Slow Response Times</h4>

```
app.use((req, res, next) =&gt; {
    const start = Date.now();
    res.on(&#39;finish&#39;, () =&gt; {
        const duration = Date.now() - start;
        if (duration &gt; 5000) { // 5 seconds
            logger.warn(&#39;Slow response detected:&#39;, {
                url: req.url,
                method: req.method,
                duration: duration,
                userAgent: req.get(&#39;User-Agent&#39;)
            });
        }
    });
    next();
});
```

<h4>High CPU Usage</h4>

```
const os = require(&#39;os&#39;);
function monitorCPU() {
    const cpus = os.cpus();
    const totalIdle = cpus.reduce((acc, cpu) =&gt; acc + cpu.times.idle, 0);
    const totalTick = cpus.reduce((acc, cpu) =&gt;
        acc + cpu.times.user + cpu.times.nice + cpu.times.sys + cpu.times.idle, 0);
    const idle = totalIdle / cpus.length;
    const total = totalTick / cpus.length;
    const percentageCPU = 100 - (100 * idle / total);
    if (percentageCPU &gt; 80) {
        logger.warn(&#39;High CPU usage detected:&#39;, percentageCPU.toFixed(2) + &#39;%&#39;);
    }
}
```

Ø=Þ¨ <strong>Emergency Procedures</strong>

<strong>System Outage Response</strong>

<h4>Immediate Actions</h4>

<li><p><strong>Assess Impact</strong></p>
- Check system status
- Identify affected services
- Estimate downtime

<li><p><strong>Notify Stakeholders</strong></p>
- Send outage notification
- Update status page
- Contact key users

<li><p><strong>Begin Recovery</strong></p>
- Start backup systems

• Initiate recovery procedures
• Monitor progress

<h4>Recovery Checklist</h4>

```
const recoveryChecklist = [
    &#39;Stop all services&#39;,
    &#39;Backup current state&#39;,
    &#39;Check error logs&#39;,
    &#39;Identify root cause&#39;,
    &#39;Apply fixes&#39;,
    &#39;Test functionality&#39;,
    &#39;Restart services&#39;,
    &#39;Verify system health&#39;,
    &#39;Notify stakeholders of resolution&#39;
];
async function emergencyRecovery() {
    logger.error(&#39;EMERGENCY RECOVERY INITIATED&#39;);
    for (const step of recoveryChecklist) {
        logger.info(`Recovery step: ${step}`);
        // Execute recovery step
        await executeRecoveryStep(step);
    }
    logger.info(&#39;Emergency recovery completed&#39;);
}
```

<strong>Data Breach Response</strong>

<h4>Immediate Actions</h4>

<li><p><strong>Contain the Breach</strong></p>
• Isolate affected systems
• Disable compromised accounts
• Preserve evidence
<li><p><strong>Assess Impact</strong></p>
• Identify affected data
• Determine scope of breach
• Assess potential damage
<li><p><strong>Notify Authorities</strong></p>

• Contact legal team
• Report to relevant authorities
• Notify affected users

<h4>Recovery Plan</h4>

```
const breachRecoveryPlan = {
  immediate: [
    &#39;Isolate affected systems&#39;,
    &#39;Disable compromised accounts&#39;,
    &#39;Change all passwords&#39;,
    &#39;Enable enhanced logging&#39;
  ],
  shortTerm: [
    &#39;Conduct security audit&#39;,
    &#39;Update security measures&#39;,
    &#39;Train staff on security&#39;,
    &#39;Implement additional monitoring&#39;
  ],
  longTerm: [
    &#39;Review security policies&#39;,
    &#39;Update incident response plan&#39;,
    &#39;Conduct penetration testing&#39;,
    &#39;Implement security improvements&#39;
  ]
};
```

<strong>Communication Plan</strong>

<h4>Stakeholder Communication</h4>

```
const communicationTemplates = {
  outage: {
    subject: &#39;System Maintenance Notice&#39;,
    body: &#39;We are currently performing system maintenance. Services will be restored shortly. We apologize for any inconvenience.&#39;
  },
    security: {
        subject: &#39;Security Update&#39;,
        body: &#39;We have identified and resolved a security issue. All systems are now secure and
operational.&#39;
    },
    recovery: {
        subject: &#39;System Recovery Complete&#39;,
        body: &#39;All systems have been restored and are operating normally. Thank you for your patience.&#39;
    }
};
function sendNotification(type, recipients) {
    const template = communicationTemplates[type];
    // Send notifications to recipients
    logger.info(`Notification sent: ${type} to ${recipients.length} recipients`);
}
```

Ø=ÜË <strong>Operational Checklists</strong>

<strong>Daily Operations</strong>

- <input disabled="" type="checkbox"> Check system status
- <input disabled="" type="checkbox"> Review error logs
- <input disabled="" type="checkbox"> Monitor performance metrics
- <input disabled="" type="checkbox"> Verify backup completion
- <input disabled="" type="checkbox"> Check WhatsApp connection
- <input disabled="" type="checkbox"> Review security alerts

<strong>Weekly Operations</strong>

- <input disabled="" type="checkbox"> Perform database maintenance
- <input disabled="" type="checkbox"> Clean up old files
- <input disabled="" type="checkbox"> Review user access
- <input disabled="" type="checkbox"> Update security patches
- <input disabled="" type="checkbox"> Check system resources
- <input disabled="" type="checkbox"> Review performance trends

<strong>Monthly Operations</strong>

- <input disabled="" type="checkbox"> Conduct security audit
- <input disabled="" type="checkbox"> Update dependencies

- <input disabled="" type="checkbox"> Review backup procedures
- <input disabled="" type="checkbox"> Analyze performance data
- <input disabled="" type="checkbox"> Update documentation
- <input disabled="" type="checkbox"> Plan capacity upgrades

<strong>Emergency Contacts</strong>

- <strong>Technical Lead</strong>: <a href="mailto:tech@novi.com">tech@novi.com</a>
- <strong>System Administrator</strong>: <a href="mailto:admin@novi.com">admin@novi.com</a>
- <strong>Security Team</strong>: <a href="mailto:security@novi.com">security@novi.com</a>
- <strong>24/7 Support</strong>: +234 XXX XXX XXXX

<strong>Novi</strong><br><em>Smart Commerce Suite - Maintaining operational excellence through robust procedures and monitoring</em>