```
          { value: 'popular', label: 'Most Popular' },
          { value: 'name', label: 'Name A-Z' },
          { value: 'recent', label: 'Recently Updated' }
        ]}
      />
    </div>
  </div>
</div>

{/* Featured Templates */}
{selectedCategory === 'all' && !searchQuery && (
  <div>
    <h2 className="text-lg font-semibold text-gray-900 mb-4">Featured Templates</h2>
    <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
      {templates
        .filter(t => t.featured)
        .slice(0, 3)
        .map(template => (
          <FeaturedTemplateCard key={template.id} template={template} />
        ))}
    </div>
  </div>
)}

{/* Template Grid */}
<div>
  <div className="flex items-center justify-between mb-4">
    <h2 className="text-lg font-semibold text-gray-900">
      {searchQuery ? `Search Results (${filteredTemplates.length})` : 'All Templates'}
    </h2>
    <span className="text-sm text-gray-500">
      {filteredTemplates.length} templates found
    </span>
  </div>

  {loading ? (
    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
      {Array.from({ length: 6 }).map((_, i) => (
        <TemplateCardSkeleton key={i} />
      ))}
    </div>
  ) : filteredTemplates.length > 0 ? (
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
          {filteredTemplates.map(template => (
            <TemplateCard key={template.id} template={template} />
          ))}
        </div>
      ) : (
        <EmptyState
          icon={<Search className="w-12 h-12 text-gray-400" />}
          title="No templates found"
          description={
            searchQuery
              ? `No templates match "${searchQuery}"`
              : "No templates available in this category"
          }
        />
      )}
    </div>
  </div>
```

);
};

// Template Card Component
const TemplateCard: React.FC<{ template: Template }> = ({ template }) => {
const [isDeploying, setIsDeploying] = useState(false);
const [showDeployModal, setShowDeployModal] = useState(false);

const handleQuickDeploy = async () => {
if (template.requiresConfiguration) {
setShowDeployModal(true);
return;
}

```
  setIsDeploying(true);
  try {
    await deployTemplate(template.id, {});
    toast.success(`${template.name} deployed successfully!`);
  } catch (error) {
    toast.error(`Failed to deploy ${template.name}: ${error.message}`);
  } finally {
    setIsDeploying(false);
  }
```

```
};

return ( <> <Card className="hover:shadow-lg transition-shadow duration-200"> <div className="p-6"> {/* Header */} <div className="flex items-start justify-between mb-4"> <div className="flex items-center space-x-3"> <div className="flex-shrink-0"> <img src={template.icon || '/default-template-icon.svg'} alt={template.name} className="w-10 h-10 rounded-lg" /> </div> <div className="min-w-0 flex-1"> <h3 className="text-lg font-semibold text-gray-900 truncate"> {template.name} </h3> <p className="text-sm text-gray-600">{template.author}</p> </div> </div> {template.featured && ( <Badge variant="gold">Featured</Badge> )} </div>
{/* Description */}
<p className="text-gray-600 text-sm mb-4 line-clamp-3">
  {template.description}
</p>

{/* Tags */}
<div className="flex flex-wrap gap-2 mb-4">
  {template.tags.slice(0, 3).map((tag, index) => (
    <Badge key={index} variant="secondary" size="sm">
      {tag}
    </Badge>
  ))}
  {template.tags.length > 3 && (
    <Badge variant="secondary" size="sm">
      +{template.tags.length - 3}
    </Badge>
  )}
</div>

{/* Stats */}
<div className="flex items-center justify-between text-sm text-gray-500 mb-4">
  <div className="flex items-center space-x-4">
    <span className="flex items-center">
      <Download className="w-4 h-4 mr-1" />
      {formatNumber(template.downloads || 0)}
    </span>
    <span className="flex items-center">
      <Star className="w-4 h-4 mr-1" />
      {template.rating || 'N/A'}
    </span>
  </div>
  <span>{formatRelativeTime(template.updatedAt)}</span>
</div>

{/* Actions */}
<div className="flex space-x-2">
  <Button
    variant="primary"
    className="flex-1"
    onClick={handleQuickDeploy}
    disabled={isDeploying}
  >
    {isDeploying ? (
              <>
                <Loader className="w-4 h-4 mr-2 animate-spin" />
                Deploying...
              </>
            ) : (
              <>
                <Rocket className="w-4 h-4 mr-2" />
                Deploy
              </>
            )}
          </Button>
          <Button
            variant="secondary"
            onClick={() => setShowDeployModal(true)}
          >
            <Settings className="w-4 h-4" />
          </Button>
        </div>
      </div>
    </Card>

    {/* Deploy Modal */}
    {showDeployModal && (
      <DeployModal
        template={template}
        onClose={() => setShowDeployModal(false)}
        onDeploy={(config) => {
          deployTemplate(template.id, config);
          setShowDeployModal(false);
        }}
      />
    )}
  </>
```

);
};

// Deploy Modal Component const DeployModal: React.FC<{ template: Template; onClose: () => void; onDeploy: (config: any) => void; }> = ({ template, onClose, onDeploy }) => { const [config, setConfig] = useState(template.defaultConfig || {}); const [serverName, setServerName] = useState(`${template.name.toLowerCase().replace(/\s+/g, '-')}-${Date.now()}`); const [isDeploying, setIsDeploying] = useState(false); const [errors, setErrors] = useState<Record<string, string>>({});

const validateConfig = () => {
const newErrors: Record<string, string> = {};

```
  if (!serverName.trim()) {
    newErrors.serverName = 'Server name is required';
  } else if (!/^[a-zA-Z0-9-_]+$/.test(serverName)) {
    newErrors.serverName = 'Server name can only contain letters, numbers, hyphens, and
  underscores';
  }

  // Validate required fields from template schema
  if (template.configSchema) {
    for (const [key, schema] of Object.entries(template.configSchema)) {
      if (schema.required && !config[key]) {
        newErrors[key] = `${schema.label || key} is required`;
      }
    }
  }

  setErrors(newErrors);
  return Object.keys(newErrors).length === 0;
```

};

const handleDeploy = async () => {
if (!validateConfig()) return;

```
  setIsDeploying(true);
  try {
    await onDeploy({
      name: serverName,
      ...config
    });
    toast.success(`${template.name} deployed successfully!`);
    onClose();
  } catch (error) {
    toast.error(`Failed to deploy: ${error.message}`);
  } finally {
    setIsDeploying(false);
  }
```

};

```
return ( <Modal isOpen onClose={onClose} size="lg"> <div className="p-6"> <div className="flex
items-center space-x-3 mb-6"> <img src={template.icon || '/default-template-icon.svg'} alt=
{template.name} className="w-12 h-12 rounded-lg" /> <div> <h2 className="text-xl font-semibold
text-gray-900"> Deploy {template.name} </h2> <p className="text-gray-600">{template.description}
</p> </div> </div>
```

```
    <div className="space-y-6">
      {/* Server Name */}
      <div>
        <label className="block text-sm font-medium text-gray-700 mb-2">
          Server Name
        </label>
        <input
          type="text"
          value={serverName}
          onChange={(e) => setServerName(e.target.value)}
          className={`w-full px-3 py-2 border rounded-md focus:outline-none focus:ring-2
focus:ring-blue-500 ${
            errors.serverName ? 'border-red-300' : 'border-gray-300'
          }`}
          placeholder="my-mcp-server"
        />
        {errors.serverName && (
          <p className="mt-1 text-sm text-red-600">{errors.serverName}</p>
        )}
      </div>

      {/* Configuration Fields */}
      {template.configSchema && (
        <div>
          <h3 className="text-lg font-medium text-gray-900 mb-4">Configuration</h3>
          <div className="space-y-4">
            {Object.entries(template.configSchema).map(([key, schema]) => (
              <div key={key}>
                <label className="block text-sm font-medium text-gray-700 mb-2">
                  {schema.label || key}
                  {schema.required && <span className="text-red-500 ml-1">*</span>}
                </label>

                {schema.type === 'string' && (
                  <input
                    type={schema.secret ? 'password' : 'text'}
                    value={config[key] || ''}
                    onChange={(e) => setConfig({ ...config, [key]: e.target.value })}
                    className={`w-full px-3 py-2 border rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500 ${
                      errors[key] ? 'border-red-300' : 'border-gray-300'
                    }`}
                    placeholder={schema.placeholder}
                  />
                )}

                {schema.type === 'number' && (
                  <input
                    type="number"
                    value={config[key] || ''}
                    onChange={(e) => setConfig({ ...config, [key]: parseInt(e.target.value)
|| '' })}
                    className={`w-full px-3 py-2 border rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500 ${
                      errors[key] ? 'border-red-300' : 'border-gray-300'
                    }`}
                    placeholder={schema.placeholder}
                    min={schema.min}
                    max={schema.max}
                  />
                )}

                {schema.type === 'select' && (
                  <select
                    value={config[key] || ''}
                    onChange={(e) => setConfig({ ...config, [key]: e.target.value })}
                    className={`w-full px-3 py-2 border rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500 ${
                      errors[key] ? 'border-red-300' : 'border-gray-300'
                    }`}
                  >
                    <option value="">Select {schema.label || key}</option>
                    {schema.options?.map((option) => (
                      <option key={option.value} value={option.value}>
                        {option.label}
                      </option>
                    ))}
                  </select>
                )}

                {schema.type === 'array' && (
                  <EnvironmentEditor
                    value={config[key] || []}
                    onChange={(value) => setConfig({ ...config, [key]: value })}
                    placeholder={schema.placeholder}
                  />
                )}
```

```
              {schema.description && (
                <p className="mt-1 text-sm text-gray-500">{schema.description}</p>
              )}

              {errors[key] && (
                <p className="mt-1 text-sm text-red-600">{errors[key]}</p>
              )}
            </div>
          ))}
        </div>
      </div>
    )}

    {/* Template Info */}
    <div className="bg-blue-50 border border-blue-200 rounded-lg p-4">
      <h4 className="font-medium text-blue-900 mb-2">Template Information</h4>
      <div className="text-sm text-blue-800 space-y-1">
        <div>Version: {template.version}</div>
        <div>Type: {template.type}</div>
        {template.healthEndpoint && (
          <div>Health Check: {template.healthEndpoint}</div>
        )}
        {template.documentation && (
          <div>
            <a
              href={template.documentation}
              target="_blank"
              rel="noopener noreferrer"
              className="inline-flex items-center text-blue-600 hover:text-blue-800"
            >
              View Documentation
              <ExternalLink className="w-3 h-3 ml-1" />
            </a>
          </div>
        )}
      </div>
    </div>
  </div>

  {/* Actions */}
  <div className="flex justify-end space-x-3 mt-6 pt-6 border-t border-gray-200">
    <Button variant="secondary" onClick={onClose} disabled={isDeploying}>
      Cancel
      </Button>
      <Button
        variant="primary"
        onClick={handleDeploy}
        disabled={isDeploying}
      >
        {isDeploying ? (
          <>
            <Loader className="w-4 h-4 mr-2 animate-spin" />
            Deploying...
          </>
        ) : (
          <>
            <Rocket className="w-4 h-4 mr-2" />
            Deploy Server
          </>
        )}
      </Button>
    </div>
  </div>
</Modal>
```

);

};

~~~~
**Backend Template System:**
```javascript
// Template Service
class TemplateService {
  constructor() {
    this.templates = new Map();
    this.loadBuiltinTemplates();
  }

  loadBuiltinTemplates() {
    const builtinTemplates = [
      {
        id: 'filesystem',
        name: 'Filesystem Access',
        description: 'Provides secure access to local filesystem with configurable paths and permissions',
        author: 'MCP Team',
        category: 'Storage',
        type: 'container',
        image: 'mcp/filesystem:latest',
        icon: '/icons/filesystem.svg',
        featured: true,
        rating: 4.8,
        downloads: 15420,
        tags: ['filesystem', 'files', 'storage', 'local'],
        version: '1.2.0',
        healthEndpoint: '/health',
        requiresConfiguration: true,
        configSchema: {
          allowedPaths: {
            type: 'array',
            label: 'Allowed Paths',
            description: 'List of directories that the server can access',
            required: true,
            placeholder: 'e.g., /home/user/documents'
          },
          readOnly: {
            type: 'select',
            label: 'Access Mode',
            options: [
              { value: false, label: 'Read/Write' },
              { value: true, label: 'Read Only' }
~~~~

```
          ],
          default: false
        }
      },
      defaultConfig: {
        allowedPaths: ['/tmp'],
        readOnly: false
      },
      documentation: 'https://docs.mcp.dev/servers/filesystem'
    },
    {
      id: 'github',
      name: 'GitHub Integration',
      description: 'Connect to GitHub repositories for code analysis, issue management, 
and pull requests',
      author: 'GitHub',
      category: 'Development',
      type: 'container',
      image: 'mcp/github:latest',
      icon: '/icons/github.svg',
      featured: true,
      rating: 4.9,
      downloads: 23150,
      tags: ['github', 'git', 'repository', 'development'],
      version: '2.1.0',
      healthEndpoint: '/health',
      requiresConfiguration: true,
      configSchema: {
        token: {
          type: 'string',
          label: 'GitHub Token',
          description: 'Personal access token with repo permissions',
          required: true,
          secret: true,
          placeholder: 'ghp_xxxxxxxxxxxxxxxxxxxx'
        },
        repositories: {
          type: 'array',
          label: 'Repositories',
          description: 'List of repositories to access (format: owner/repo)',
          placeholder: 'e.g., octocat/Hello-World'
        }
      },
      documentation: 'https://docs.github.com/en/developers/apps/mcp-server'
      },
      {
        id: 'postgres',
        name: 'PostgreSQL Database',
        description: 'Query and manage PostgreSQL databases with schema introspection and
query execution',
        author: 'PostgreSQL Team',
        category: 'Database',
        type: 'container',
        image: 'mcp/postgres:latest',
        icon: '/icons/postgresql.svg',
        featured: false,
        rating: 4.6,
        downloads: 8930,
        tags: ['postgresql', 'database', 'sql', 'query'],
        version: '1.0.3',
        healthEndpoint: '/health',
        requiresConfiguration: true,
        configSchema: {
          connectionString: {
            type: 'string',
            label: 'Connection String',
            description: 'PostgreSQL connection string',
            required: true,
            secret: true,
            placeholder: 'postgresql://user:password@localhost:5432/database'
          },
          queryTimeout: {
            type: 'number',
            label: 'Query Timeout (ms)',
            description: 'Maximum time to wait for query execution',
            default: 30000,
            min: 1000,
            max: 300000
          }
        },
        defaultConfig: {
          queryTimeout: 30000
        }
      },
      {
        id: 'slack',
        name: 'Slack Workspace',
        description: 'Send messages, manage channels, and interact with Slack workspaces',
        author: 'Slack Technologies',
        category: 'Communication',
        type: 'nodejs',
        command: 'npx',
        args: ['@mcp/slack-server'],
        icon: '/icons/slack.svg',
        featured: false,
        rating: 4.7,
        downloads: 12450,
        tags: ['slack', 'messaging', 'communication', 'teams'],
        version: '1.5.2',
        requiresConfiguration: true,
        configSchema: {
          botToken: {
            type: 'string',
            label: 'Bot Token',
            description: 'Slack bot token (starts with xoxb-)',
            required: true,
            secret: true,
            placeholder: 'xoxb-xxxxxxxxxx-xxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxx'
          },
          signingSecret: {
            type: 'string',
            label: 'Signing Secret',
            description: 'Slack app signing secret for request verification',
            required: true,
            secret: true
          }
        }
      },
      {
        id: 'openapi',
        name: 'OpenAPI Client',
        description: 'Generate MCP client from OpenAPI/Swagger specifications',
        author: 'OpenAPI Initiative',
        category: 'API',
        type: 'python',
        command: 'python',
        args: ['-m', 'mcp_openapi'],
        icon: '/icons/openapi.svg',
        featured: false,
        rating: 4.4,
        downloads: 5670,
        tags: ['openapi', 'swagger', 'api', 'client'],
        version: '0.8.1',
        requiresConfiguration: true,
        configSchema: {
          specUrl: {
            type: 'string',
            label: 'OpenAPI Spec URL',
            description: 'URL to OpenAPI/Swagger specification',
            required: true,
            placeholder: 'https://api.example.com/openapi.json'
          },
          apiKey: {
            type: 'string',
            label: 'API Key',
            description: 'API key for authentication (if required)',
            secret: true
          }
        }
      }
    ];

    builtinTemplates.forEach(template => {
      this.templates.set(template.id, {
        ...template,
        createdAt: new Date().toISOString(),
        updatedAt: new Date().toISOString()
      });
    });
  }

  async getAllTemplates() {
    return Array.from(this.templates.values());
  }

  async getTemplate(id) {
    return this.templates.get(id);
  }

  async searchTemplates(query, filters = {}) {
    const templates = Array.from(this.templates.values());

    let filtered = templates;

    // Text search
    if (query) {
      const searchTerm = query.toLowerCase();
      filtered = filtered.filter(template =>
        template.name.toLowerCase().includes(searchTerm) ||
        template.description.toLowerCase().includes(searchTerm) ||
        template.tags.some(tag => tag.toLowerCase().includes(searchTerm))
      );
    }

    // Category filter
    if (filters.category && filters.category !== 'all') {
      filtered = filtered.filter(template => template.category === filters.category);
    }

    // Type filter
    if (filters.type) {
      filtered = filtered.filter(template => template.type === filters.type);
    }

    // Sort
    if (filters.sortBy) {
      filtered.sort((a, b) => {
        switch (filters.sortBy) {
          case 'popular':
            return (b.downloads || 0) - (a.downloads || 0);
          case 'name':
            return a.name.localeCompare(b.name);
          case 'recent':
            return new Date(b.updatedAt).getTime() - new Date(a.updatedAt).getTime();
          default:
            return 0;
        }
      });
    }

    return filtered;
  }

  async deployTemplate(templateId, config, serverName) {
    const template = this.templates.get(templateId);
    if (!template) {
      throw new Error('Template not found');
    }

    // Merge template defaults with user config
    const finalConfig = {
      ...template.defaultConfig,
      ...config
    };

    // Create server configuration
    const serverConfig = {
      name: serverName,
      type: template.type,
      ...finalConfig
    };

    if (template.type === 'container') {
      serverConfig.image = template.image;
      serverConfig.healthEndpoint = template.healthEndpoint;
    } else {
      serverConfig.command = template.command;
      serverConfig.args = template.args;
    }

    // Use existing server manager to create the server
    const serverManager = require('./EnhancedServerManager');
    const manager = new serverManager();

    const result = await manager.createServer(serverConfig);

    // Increment download count
    template.downloads = (template.downloads || 0) + 1;

    return result;
  }
}

module.exports = TemplateService;
```

Template API Endpoints:

```javascript
// Template routes
app.get('/api/templates', async (req, res) => {
  try {
    const { search, category, type, sortBy } = req.query;
    const templates = await templateService.searchTemplates(search, {
      category,
      type,
      sortBy
    });
    res.json({ success: true, data: templates });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
});

app.get('/api/templates/:id', async (req, res) => {
  try {
    const template = await templateService.getTemplate(req.params.id);
    if (!template) {
      return res.status(404).json({ success: false, error: 'Template not found' });
    }
    res.json({ success: true, data: template });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
});

app.post('/api/templates/:id/deploy', async (req, res) => {
  try {
    const { name, config } = req.body;
    const result = await templateService.deployTemplate(req.params.id, config, name);
    res.json({ success: true, data: result });
  } catch (error) {
    res.status(400).json({ success: false, error: error.message });
  }
});
```

Acceptance Criteria:

- [ ] Marketplace loads with all available templates
- [ ] Search and filtering work correctly

- [ ] Template cards display all relevant information
- [ ] One-click deployment works for simple templates
- [ ] Configuration modal works for complex templates
- [ ] Template deployment creates and starts servers correctly
- [ ] Download counts and ratings are tracked
- [ ] Featured templates are highlighted

Enhanced Real-time Updates

REQ-3.4: WebSocket Integration

Priority: P0
Estimated Effort: 10 hours

Functional Requirements:

- Real-time server status updates
- Live log streaming
- Health metric broadcasts
- System resource monitoring
- Connection management

Technical Implementation:

```
javascript
```

```javascript
// WebSocket Service
class WebSocketService {
  constructor(server) {
    this.wss = new WebSocket.Server({ server });
    this.clients = new Map(); // clientId -> { ws, subscriptions }
    this.setupConnectionHandling();
  }

  setupConnectionHandling() {
    this.wss.on('connection', (ws, req) => {
      const clientId = generateId();
      this.clients.set(clientId, {
        ws,
        subscriptions: new Set(),
        lastPing: Date.now()
      });

      ws.on('message', (message) => {
        this.handleMessage(clientId, message);
      });

      ws.on('close', () => {
        this.clients.delete(clientId);
      });

      ws.on('pong', () => {
        const client = this.clients.get(clientId);
        if (client) {
          client.lastPing = Date.now();
        }
      });

      // Send welcome message
      this.sendToClient(clientId, 'connected', { clientId });
    });

    // Setup ping/pong for connection health
    setInterval(() => {
      this.pingClients();
    }, 30000);
  }

  handleMessage(clientId, rawMessage) {
    try {
      const message = JSON.parse(rawMessage);

      switch (message.type) {
        case 'subscribe':
          this.handleSubscribe(clientId, message.data);
          break;
        case 'unsubscribe':
          this.handleUnsubscribe(clientId, message.data);
          break;
        case 'ping':
          this.sendToClient(clientId, 'pong', {});
          break;
      }
    } catch (error) {
      console.error('Failed to handle WebSocket message:', error);
    }
  }

  handleSubscribe(clientId, subscription) {
    const client = this.clients.get(clientId);
    if (!client) return;

    client.subscriptions.add(subscription);
    this.sendToClient(clientId, 'subscribed', { subscription });
  }

  handleUnsubscribe(clientId, subscription) {
    const client = this.clients.get(clientId);
    if (!client) return;

    client.subscriptions.delete(subscription);
    this.sendToClient(clientId, 'unsubscribed', { subscription });
  }

  broadcast(type, data, filter = null) {
    const message = JSON.stringify({ type, data, timestamp: new Date().toISOString() });

    for (const [clientId, client] of this.clients) {
      if (client.ws.readyState === WebSocket.OPEN) {
        if (!filter || filter(client)) {
          client.ws.send(message);
        }
      }
    }
  }

  broadcastToSubscribers(subscription, type, data) {
    this.broadcast(type, data, (client) =>
      client.subscriptions.has(subscription)
    );
  }

  sendToClient(clientId, type, data) {
    const client = this.clients.get(clientId);
    if (client && client.ws.readyState === WebSocket.OPEN) {
      const message = JSON.stringify({
        type,
        data,
        timestamp: new Date().toISOString()
      });
      client.ws.send(message);
    }
  }

  pingClients() {
    const now = Date.now();
    const staleThreshold = 60000; // 1 minute

    for (const [clientId, client] of this.clients) {
      if (now - client.lastPing > staleThreshold) {
        // Remove stale client
        client.ws.terminate();
        this.clients.delete(clientId);
      } else if (client.ws.readyState === WebSocket.OPEN) {
        client.ws.ping();
      }
    }
  }

  getConnectionCount() {
    return this.clients.size;
  }
}

module.exports = WebSocketService;
```

React WebSocket Hook:

typescript

```
// useWebSocket hook
export const useWebSocket = (url: string) => {
  const [socket, setSocket] = useState<WebSocket | null>(null);
  const [isConnected, setIsConnected] = useState(false);
  const [lastMessage, setLastMessage] = useState<any>(null);
  const subscriptions = useRef<Set<string>>(new Set());

  useEffect(() => {
    const ws = new WebSocket(url);

    ws.onopen = () => {
      setIsConnected(true);
      setSocket(ws);
    };

    ws.onmessage = (event) => {
      try {
        const message = JSON.parse(event.data);
        setLastMessage(message);
      } catch (error) {
        console.error('Failed to parse WebSocket message:', error);
      }
    };

    ws.onclose = () => {
      setIsConnected(false);
      setSocket(null);
    };

    ws.onerror = (error) => {
      console.error('WebSocket error:', error);
    };

    return () => {
      ws.close();
    };
  }, [url]);

  const subscribe = useCallback((subscription: string) => {
    if (socket && isConnected) {
      subscriptions.current.add(subscription);
      socket.send(JSON.stringify({
        type: 'subscribe',
        data: subscription
      }));
    }
  }, [socket, isConnected]);

  const unsubscribe = useCallback((subscription: string) => {
    if (socket && isConnected) {
      subscriptions.current.delete(subscription);
      socket.send(JSON.stringify({
        type: 'unsubscribe',
        data: subscription
      }));
    }
  }, [socket, isConnected]);

  const sendMessage = useCallback((type: string, data: any) => {
    if (socket && isConnected) {
      socket.send(JSON.stringify({ type, data }));
    }
  }, [socket, isConnected]);

  return {
    socket,
    isConnected,
    lastMessage,
    subscribe,
    unsubscribe,
    sendMessage
  };
};

// useRealTimeServers hook
export const useRealTimeServers = () => {
  const { lastMessage, subscribe, unsubscribe } = useWebSocket('ws://localhost:3001/ws');
  const [servers, setServers] = useState<MCPServer[]>([]);

  useEffect(() => {
    subscribe('servers');
    return () => unsubscribe('servers');
  }, [subscribe, unsubscribe]);

  useEffect(() => {
    if (lastMessage) {
      switch (lastMessage.type) {
        case 'server:created':
          setServers(prev => [...prev, lastMessage.data]);
          break;
        case 'server:updated':
          setServers(prev => prev.map(s =>
            s.id === lastMessage.data.id ? { ...s, ...lastMessage.data } : s
          ));
          break;
        case 'server:deleted':
          setServers(prev => prev.filter(s => s.id !== lastMessage.data.id));
          break;
        case 'server:status':
          setServers(prev => prev.map(s =>
            s.id === lastMessage.data.serverId
              ? { ...s, status: lastMessage.data.status }
              : s
          ));
          break;
      }
    }
  }, [lastMessage]);

  return servers;
};
```

Acceptance Criteria:

- [ ] WebSocket connections establish successfully
- [ ] Real-time updates work across all components
- [ ] Connection health monitoring works correctly
- [ ] Subscription system filters messages properly
- [ ] WebSocket reconnection works after disconnection

Testing Requirements

Unit Tests

```
javascript
```

```
describe('Dashboard Components', () => {
  test('ServerCard displays server information correctly', () => {
    const mockServer = {
      id: 'test-server',
      name: 'Test Server',
      type: 'container',
      status: 'running',
      healthStatus: 'healthy'
    };

    render(<ServerCard server={mockServer} />);

    expect(screen.getByText('Test Server')).toBeInTheDocument();
    expect(screen.getByText('running')).toBeInTheDocument();
    expect(screen.getByText('healthy')).toBeInTheDocument();
  });

  test('ServerGrid filters and sorts correctly', () => {
    const mockServers = [
      { id: '1', name: 'A Server', status: 'running' },
      { id: '2', name: 'B Server', status: 'stopped' },
      { id: '3', name: 'C Server', status: 'running' }
    ];

    render(<ServerGrid servers={mockServers} />);

    // Test filtering
    fireEvent.change(screen.getByDisplayValue('All Status'), { target: { value: 'running' } });
    expect(screen.getAllByText(/Server/)).toHaveLength(2);

    // Test sorting
    fireEvent.change(screen.getByDisplayValue('Sort by Name'), { target: { value: 'name' } });
    const serverElements = screen.getAllByText(/Server/);
    expect(serverElements[0]).toHaveTextContent('A Server');
  });
});

describe('Template System', () => {
  test('TemplateCard displays template information', () => {
    const mockTemplate = {
      id: 'test-template',
      name: 'Test Template',
      description: 'A test template',
      author: 'Test Author',
      downloads: 1000,
      rating: 4.5,
      tags: ['test', 'template']
    };

    render(<TemplateCard template={mockTemplate} />);

    expect(screen.getByText('Test Template')).toBeInTheDocument();
    expect(screen.getByText('A test template')).toBeInTheDocument();
    expect(screen.getByText('Test Author')).toBeInTheDocument();
    expect(screen.getByText('1,000')).toBeInTheDocument();
  });

  test('DeployModal validates configuration correctly', async () => {
    const mockTemplate = {
      id: 'test-template',
      name: 'Test Template',
      configSchema: {
        apiKey: {
          type: 'string',
          required: true,
          label: 'API Key'
        }
      }
    };

    render(
      <DeployModal
        template={mockTemplate}
        onClose={() => {}}
        onDeploy={() => {}}
      />
    );

    // Try to deploy without required field
    fireEvent.click(screen.getByText('Deploy Server'));

    expect(screen.getByText('API Key is required')).toBeInTheDocument();
  });
});

describe('Real-time Updates', () => {
  test('useWebSocket hook connects and receives messages', async () => {
    const mockWebSocket = {
      onopen: null,
      onmessage: null,
      onclose: null,
      onerror: null,
      send: jest.fn(),
      close: jest.fn()
    };

    global.WebSocket = jest.fn(() => mockWebSocket);

    const { result } = renderHook(() => useWebSocket('ws://localhost:3001/ws'));

    // Simulate connection
    act(() => {
      mockWebSocket.onopen();
    });

    expect(result.current.isConnected).toBe(true);

    // Simulate message
    act(() => {
      mockWebSocket.onmessage({
        data: JSON.stringify({ type: 'test', data: { message: 'hello' } })
      });
    });

    expect(result.current.lastMessage).toEqual({
      type: 'test',
      data: { message: 'hello' }
    });
  });
});
```

Integration Tests

```
javascript
```

```
describe('End-to-End Workflows', () => {
  test('Complete server creation and monitoring workflow', async () => {
    // Render dashboard
    render(<Dashboard />);

    // Create new server
    fireEvent.click(screen.getByText('Add Server'));

    // Fill out form
    fireEvent.change(screen.getByLabelText('Server Name'), {
      target: { value: 'test-server' }
    });
    fireEvent.change(screen.getByLabelText('Docker Image'), {
      target: { value: 'mcp/test:latest' }
    });

    // Submit form
    fireEvent.click(screen.getByText('Create Server'));

    // Wait for server to appear
    await waitFor(() => {
      expect(screen.getByText('test-server')).toBeInTheDocument();
    });

    // Start server
    fireEvent.click(screen.getByLabelText('Start test-server'));

    // Wait for status update
    await waitFor(() => {
      expect(screen.getByText('running')).toBeInTheDocument();
    });
  });

  test('Template deployment workflow', async () => {
    // Navigate to marketplace
    render(<Marketplace />);

    // Wait for templates to load
    await waitFor(() => {
      expect(screen.getByText('Filesystem Access')).toBeInTheDocument();
    });

    // Click deploy on filesystem template
    fireEvent.click(screen.getByText('Deploy'));

    // Configure template
    fireEvent.change(screen.getByLabelText('Server Name'), {
      target: { value: 'my-filesystem-server' }
    });
    fireEvent.change(screen.getByLabelText('Allowed Paths'), {
      target: { value: '/tmp' }
    });

    // Deploy server
    fireEvent.click(screen.getByText('Deploy Server'));

    // Verify deployment success
    await waitFor(() => {
      expect(screen.getByText('deployed successfully')).toBeInTheDocument();
    });
  });
});
```

Performance Requirements

Frontend Performance

- [ ] Initial page load: <3 seconds
- [ ] Component rendering: <100ms
- [ ] Real-time updates: <200ms latency
- [ ] Search/filter operations: <500ms
- [ ] Large server lists (100+ servers): <2 seconds render time

Backend Performance

- [ ] API response times: <500ms average
- [ ] WebSocket message delivery: <100ms
- [ ] Template deployment: <10 seconds
- [ ] Database queries: <200ms
- [ ] Health checks: <2 seconds timeout

Accessibility Requirements

WCAG 2.1 AA Compliance

- [ ] Keyboard navigation for all interactive elements

- [ ] Screen reader compatibility
- [ ] Color contrast ratios meet standards
- [ ] Alternative text for images and icons
- [ ] Focus indicators visible
- [ ] Form labels properly associated

Implementation Details

tsx

```
// Accessible Button Component
const Button: React.FC<ButtonProps> = ({
  children,
  onClick,
  disabled,
  variant = 'primary',
  size = 'md',
  ariaLabel,
  ...props
}) => {
  return (
    <button
      onClick={onClick}
      disabled={disabled}
      aria-label={ariaLabel}
      className={`
        focus:outline-none focus:ring-2 focus:ring-blue-500 focus:ring-offset-2
        disabled:opacity-50 disabled:cursor-not-allowed
        transition-colors duration-200
        ${getVariantClasses(variant)}
        ${getSizeClasses(size)}
      `}
      {...props}
    >
      {children}
    </button>
  );
};

// Accessible Form Components
const FormField: React.FC<FormFieldProps> = ({
  label,
  error,
  required,
  children,
  id
}) => {
  return (
    <div className="space-y-1">
      <label
        htmlFor={id}
        className="block text-sm font-medium text-gray-700"
      >
```

```
        {label}
        {required && <span className="text-red-500 ml-1" aria-label="required">*</span>}
      </label>
      {children}
      {error && (
        <p
          id={`${id}-error`}
          className="text-sm text-red-600"
          role="alert"
          aria-live="polite"
        >
          {error}
        </p>
      )}
    </div>
  );
};
```

Enhanced Error Handling

Error Boundary Implementation

tsx

```tsx
class ErrorBoundary extends React.Component<
  { children: React.ReactNode; fallback?: React.ComponentType<{ error: Error }> },
  { hasError: boolean; error: Error | null }
> {
  constructor(props: any) {
    super(props);
    this.state = { hasError: false, error: null };
  }

  static getDerivedStateFromError(error: Error) {
    return { hasError: true, error };
  }

  componentDidCatch(error: Error, errorInfo: React.ErrorInfo) {
    console.error('Error caught by boundary:', error, errorInfo);

    // Send error to monitoring service
    if (process.env.NODE_ENV === 'production') {
      // Example: Sentry.captureException(error, { extra: errorInfo });
    }
  }

  render() {
    if (this.state.hasError) {
      const FallbackComponent = this.props.fallback || DefaultErrorFallback;
      return <FallbackComponent error={this.state.error!} />;
    }

    return this.props.children;
  }
}

const DefaultErrorFallback: React.FC<{ error: Error }> = ({ error }) => (
  <div className="min-h-screen flex items-center justify-center bg-gray-50">
    <div className="max-w-md w-full bg-white rounded-lg shadow-lg p-6">
      <div className="flex items-center mb-4">
        <AlertCircle className="w-8 h-8 text-red-500 mr-3" />
        <h1 className="text-xl font-semibold text-gray-900">Something went wrong</h1>
      </div>
      <p className="text-gray-600 mb-4">
        An unexpected error occurred. Please try refreshing the page.
      </p>
      <details className="mb-4">
```

```
          <summary className="cursor-pointer text-sm text-gray-500">
            Technical details
          </summary>
          <pre className="mt-2 text-xs bg-gray-100 p-2 rounded overflow-auto">
            {error.message}
          </pre>
        </details>
        <button
          onClick={() => window.location.reload()}
          className="w-full px-4 py-2 bg-blue-600 text-white rounded-md hover:bg-blue-700"
        >
          Refresh Page
        </button>
      </div>
    </div>
  );
```

Success Metrics

User Experience Metrics

- [ ] Task completion rate >90% for common workflows
- [ ] Average time to deploy template <2 minutes
- [ ] User satisfaction score >4.5/5
- [ ] Support ticket volume <5% of deployments

Technical Metrics

- [ ] 99.9% uptime for web interface
- [ ] <100ms WebSocket message latency
- [ ] <2MB initial JavaScript bundle size
- [ ] >95 Lighthouse performance score
- [ ] Zero critical accessibility violations

Business Metrics

- [ ] 1000+ active installations within 3 months
- [ ] 50+ community-contributed templates
- [ ] 80% user retention after first month
- [ ] Average 10+ servers per active installation

Known Limitations

1. **No user authentication** - All operations are public access
2. **Local deployment only** - No multi-instance or clustering support
3. **Basic template validation** - Limited configuration schema validation
4. **No enterprise features** - Missing advanced access controls, audit logs
5. **Limited customization** - UI themes and branding not configurable

Dependencies

Frontend Dependencies

json

```json
{
  "react": "^18.2.0",
  "react-router-dom": "^6.8.0",
  "@types/react": "^18.0.0",
  "tailwindcss": "^3.2.0",
  "lucide-react": "^0.263.1",
  "recharts": "^2.5.0",
  "react-hook-form": "^7.43.0",
  "react-query": "^3.39.0"
}
```

Backend Dependencies

json

```json
{
  "ws": "^8.14.0",
  "express-rate-limit": "^6.7.0",
  "helmet": "^6.0.0",
  "cors": "^2.8.5",
  "compression": "^1.7.4"
}
```

Deployment Configuration

Production Docker Compose

yaml

```yaml
version: '3.8'
services:
  mcp-manager:
    build: .
    ports:
      - "3000:3000"
      - "3001:3001"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - mcp_data:/app/data
      - mcp_logs:/app/logs
    environment:
      - NODE_ENV=production
      - LOG_LEVEL=info
      - ENABLE_METRICS=true
      - WS_HEARTBEAT_INTERVAL=30000
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:3000/health"]
      interval: 30s
      timeout: 10s
      retries: 3

volumes:
  mcp_data:
  mcp_logs:
```

Security Headers

```javascript
// Production security middleware
app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'", "'unsafe-inline'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      imgSrc: ["'self'", "data:", "https:"],
      connectSrc: ["'self'", "ws:", "wss:"],
      fontSrc: ["'self'"],
      objectSrc: ["'none'"],
      mediaSrc: ["'self'"],
      frameSrc: ["'none'"]
    }
  },
  hsts: {
    maxAge: 31536000,
    includeSubDomains: true,
    preload: true
  }
}));

// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // limit each IP to 100 requests per windowMs
  message: 'Too many requests from this IP'
});
app.use('/api/', limiter);
```

Next Phase Handoff

Upon completion of Phase 3, the following will be ready for Phase 4:

- Complete user interface with excellent UX
- Real-time updates and WebSocket infrastructure
- Marketplace with template system
- Production-ready deployment configuration
- Comprehensive testing suite
- Accessibility compliance

- Performance optimization
- Security hardening
- Documentation and examples

Phase 4 will focus on polish, documentation, community features, and preparation for open source release. <h2 className="text-lg font-semibold text-gray-900">MCP Servers</h2> <div className="flex space-x-2"> <Button variant="secondary" size="sm"> <RefreshCw className="w-4 h-4" /> </Button> <Button variant="primary" size="sm"> <Plus className="w-4 h-4 mr-1" /> Add Server </Button> </div> </div> </div>

```
          {loading ? (
            <div className="p-8 text-center">
              <LoadingSpinner />
            </div>
          ) : (
            <ServerGrid servers={servers} />
          )}
        </div>
      </div>

      {/* Sidebar */}
      <div className="space-y-6">
        {/* System Resources */}
        <SystemResourceCard metrics={systemMetrics} />

        {/* Recent Activity */}
        <RecentActivityCard activities={recentActivity} />

        {/* Quick Actions */}
        <QuickActionsCard />
      </div>
    </div>
  </div>
```

);
};

// Enhanced Server Grid Component const ServerGrid: React.FC<{ servers: MCPServer[] }> = ({ servers }) => { const [viewMode, setViewMode] = useState<'grid' | 'list'>('grid'); const [sortBy, setSortBy] = useState<'name' | 'status' | 'health'>('name'); const [filterStatus, setFilterStatus] = useState<string>('all');

const filteredAndSortedServers = useMemo(() => {
let filtered = servers;

```
  if (filterStatus !== 'all') {
    filtered = servers.filter(s => s.status === filterStatus);
  }

  return filtered.sort((a, b) => {
    switch (sortBy) {
      case 'name':
        return a.name.localeCompare(b.name);
      case 'status':
        return a.status.localeCompare(b.status);
      case 'health':
        return (a.healthStatus || 'unknown').localeCompare(b.healthStatus || 'unknown');
      default:
        return 0;
    }
  });
```

}, [servers, filterStatus, sortBy]);

return (<div> {/* Controls */} <div className="p-4 border-b border-gray-200 bg-gray-50"> <div className="flex items-center justify-between"> <div className="flex items-center space-x-4"> <Select value={filterStatus} onChange={setFilterStatus} options={[{ value: 'all', label: 'All Status' }, { value: 'running', label: 'Running' }, { value: 'stopped', label: 'Stopped' }, { value: 'error', label: 'Error' }]} /> <Select value={sortBy} onChange={setSortBy} options={[{ value: 'name', label: 'Sort by Name' }, { value: 'status', label: 'Sort by Status' }, { value: 'health', label: 'Sort by Health' }]} /> </div>

```
        <div className="flex items-center space-x-2">
          <button
            onClick={() => setViewMode('grid')}
            className={`p-2 rounded ${viewMode === 'grid' ? 'bg-blue-100 text-blue-700' :
  'text-gray-400'}`}
          >
            <Grid className="w-4 h-4" />
          </button>
          <button
            onClick={() => setViewMode('list')}
            className={`p-2 rounded ${viewMode === 'list' ? 'bg-blue-100 text-blue-700' :
  'text-gray-400'}`}
          >
            <List className="w-4 h-4" />
          </button>
        </div>
      </div>
    </div>

    {/* Server Display */}
    <div className="p-4">
      {viewMode === 'grid' ? (
        <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
          {filteredAndSortedServers.map(server => (
            <ServerCard key={server.id} server={server} />
          ))}
        </div>
      ) : (
        <div className="space-y-2">
          {filteredAndSortedServers.map(server => (
            <ServerListItem key={server.id} server={server} />
          ))}
        </div>
      )}
    </div>
  </div>
```

);
};

// Enhanced Server Card Component
const ServerCard: React.FC<{ server: MCPServer }> = ({ server }) => {

```
const { startServer, stopServer, restartServer } = useServerActions();
const [isExpanded, setIsExpanded] = useState(false);

const handleAction = async (action: 'start' | 'stop' | 'restart') => { try { switch (action) { case 'start': await
startServer(server.id); break; case 'stop': await stopServer(server.id); break; case 'restart': await
restartServer(server.id); break; } } catch (error) { console.error(Failed to ${action} server:, error); } };

return ( <Card className="hover:shadow-lg transition-shadow duration-200"> <div className="p-4">
{/* Header */} <div className="flex items-start justify-between mb-3"> <div className="flex items-
center space-x-3"> <div className="flex-shrink-0"> {getServerTypeIcon(server.type)} </div> <div
className="min-w-0 flex-1"> <h3 className="text-sm font-semibold text-gray-900 truncate">
{server.name} </h3> <p className="text-xs text-gray-500 truncate"> {server.image ||
${server.command} ${server.args?.join(' ')}} </p> </div> </div>
```

```
  <div className="flex items-center space-x-2">
    <StatusBadge status={server.status} />
    <HealthIndicator health={server.healthStatus} />
  </div>
</div>

{/* Metrics */}
<div className="grid grid-cols-2 gap-4 mb-4 text-sm">
  <div>
    <span className="text-gray-500">Uptime:</span>
    <span className="ml-1 font-medium">
      {formatUptime(server.uptime)}
    </span>
  </div>
  <div>
    <span className="text-gray-500">Port:</span>
    <span className="ml-1 font-medium">
      {server.port || 'N/A'}
    </span>
  </div>
</div>

{/* Expandable Health Info */}
{isExpanded && server.healthStatus && (
  <div className="mb-4 p-3 bg-gray-50 rounded-lg text-sm">
    <div className="flex justify-between items-center">
      <span className="text-gray-600">Last Health Check:</span>
      <span className="font-medium">
        {server.lastHealthCheck ?
          formatRelativeTime(server.lastHealthCheck) :
          'Never'
        }
      </span>
    </div>
    {server.averageResponseTime && (
      <div className="flex justify-between items-center mt-1">
        <span className="text-gray-600">Avg Response:</span>
        <span className="font-medium">{server.averageResponseTime}ms</span>
      </div>
    )}
  </div>
)}
```

```
    {/* Actions */}
    <div className="flex items-center justify-between">
      <div className="flex space-x-2">
        {server.status === 'running' ? (
          <>
            <Button
              variant="danger"
              size="xs"
              onClick={() => handleAction('stop')}
            >
              <Square className="w-3 h-3" />
            </Button>
            <Button
              variant="secondary"
              size="xs"
              onClick={() => handleAction('restart')}
            >
              <RotateCcw className="w-3 h-3" />
            </Button>
          </>
        ) : (
          <Button
            variant="success"
            size="xs"
            onClick={() => handleAction('start')}
          >
            <Play className="w-3 h-3" />
          </Button>
        )}
      </div>

      <div className="flex space-x-2">
        <Button
          variant="ghost"
          size="xs"
          onClick={() => setIsExpanded(!isExpanded)}
        >
          {isExpanded ? <ChevronUp className="w-3 h-3" /> : <ChevronDown className="w-3 h-3"
/>}
        </Button>
        <Button
          variant="ghost"
          size="xs"
          as={Link}
          to={`/servers/${server.id}`}
        >
          <ExternalLink className="w-3 h-3" />
        </Button>
      </div>
    </div>
  </div>
</Card>
```

);
};

// System Resource Card const SystemResourceCard: React.FC<{ metrics: SystemMetrics }> = ({ metrics }) => { return (<Card> <div className="p-4"> <h3 className="text-sm font-semibold text-gray-900 mb-4">System Resources</h3>

```
<div className="space-y-4">
  <div>
    <div className="flex justify-between text-sm mb-1">
      <span className="text-gray-600">CPU Usage</span>
      <span className="font-medium">{metrics.cpu.usage}%</span>
    </div>
    <div className="w-full bg-gray-200 rounded-full h-2">
      <div
        className="bg-blue-600 h-2 rounded-full transition-all duration-300"
        style={{ width: `${metrics.cpu.usage}%` }}
      />
    </div>
  </div>

  <div>
    <div className="flex justify-between text-sm mb-1">
      <span className="text-gray-600">Memory</span>
      <span className="font-medium">
        {formatBytes(metrics.memory.used)} / {formatBytes(metrics.memory.total)}
      </span>
    </div>
    <div className="w-full bg-gray-200 rounded-full h-2">
      <div
        className="bg-green-600 h-2 rounded-full transition-all duration-300"
        style={{ width: `${(metrics.memory.used / metrics.memory.total) * 100}%` }}
      />
    </div>
  </div>

  <div>
    <div className="flex justify-between text-sm mb-1">
      <span className="text-gray-600">Disk</span>
      <span className="font-medium">
        {formatBytes(metrics.disk.used)} / {formatBytes(metrics.disk.total)}
      </span>
    </div>
    <div className="w-full bg-gray-200 rounded-full h-2">
      <div
        className="bg-orange-600 h-2 rounded-full transition-all duration-300"
        style={{ width: `${(metrics.disk.used / metrics.disk.total) * 100}%` }}
      />
    </div>
  </div>
```

```
      </div>
    </div>
  </Card>
```

);

};

~~~
**Acceptance Criteria:**
- [ ] Dashboard loads in <2 seconds with real-time data
- [ ] Server cards display all essential information clearly
- [ ] Grid and list view modes work correctly
- [ ] Filtering and sorting function properly
- [ ] Server actions (start/stop/restart) work from dashboard
- [ ] Real-time updates reflect immediately in UI
- [ ] Responsive design works on mobile devices

### REQ-3.2: Server Detail Views
**Priority:** P0
**Estimated Effort:** 20 hours

**Functional Requirements:**
- Comprehensive server information display
- Tabbed interface for different data views
- Real-time log streaming
- Health metrics visualization
- Configuration editing capabilities

**Technical Implementation:**
```tsx
// Server Detail Page Component
const ServerDetail: React.FC = () => {
  const { serverId } = useParams<{ serverId: string }>();
  const { server, loading, error } = useServer(serverId);
  const [activeTab, setActiveTab] = useState<'overview' | 'logs' | 'health' | 'config'>
('overview');

  if (loading) return <LoadingSpinner />;
  if (error || !server) return <ErrorMessage message="Server not found" />;

  const tabs = [
    { id: 'overview', label: 'Overview', icon: <Info className="w-4 h-4" /> },
    { id: 'logs', label: 'Logs', icon: <FileText className="w-4 h-4" /> },
    { id: 'health', label: 'Health', icon: <Activity className="w-4 h-4" /> },
    { id: 'config', label: 'Configuration', icon: <Settings className="w-4 h-4" /> }
  ];

  return (
    <div className="space-y-6">
      {/* Header */}
~~~

```
      <div className="bg-white rounded-lg border border-gray-200 p-6">
        <div className="flex items-start justify-between">
          <div className="flex items-center space-x-4">
            <div className="flex-shrink-0">
              {getServerTypeIcon(server.type, 'large')}
            </div>
            <div>
              <h1 className="text-2xl font-bold text-gray-900">{server.name}</h1>
              <p className="text-gray-600">
                {server.image || `${server.command} ${server.args?.join(' ')}`}
              </p>
              <div className="flex items-center space-x-4 mt-2">
                <StatusBadge status={server.status} size="lg" />
                <HealthIndicator health={server.healthStatus} size="lg" />
                {server.port && (
                  <span className="text-sm text-gray-500">Port: {server.port}</span>
                )}
              </div>
            </div>
          </div>

          <ServerActionsPanel server={server} />
        </div>
      </div>

      {/* Tabs */}
      <div className="bg-white rounded-lg border border-gray-200">
        <div className="border-b border-gray-200">
          <nav className="flex space-x-8 px-6" aria-label="Tabs">
            {tabs.map((tab) => (
              <button
                key={tab.id}
                onClick={() => setActiveTab(tab.id as any)}
                className={`${
                  activeTab === tab.id
                    ? 'border-blue-500 text-blue-600'
                    : 'border-transparent text-gray-500 hover:text-gray-700 hover:border-
gray-300'
                } whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm flex items-
center space-x-2`}
              >
                {tab.icon}
                <span>{tab.label}</span>
              </button>
            ))}
          </nav>
        </div>

        <div className="p-6">
          {activeTab === 'overview' && <OverviewTab server={server} />}
          {activeTab === 'logs' && <LogsTab serverId={server.id} />}
          {activeTab === 'health' && <HealthTab serverId={server.id} />}
          {activeTab === 'config' && <ConfigurationTab server={server} />}
        </div>
      </div>
    </div>
  );
};

// Overview Tab Component
const OverviewTab: React.FC<{ server: MCPServer }> = ({ server }) => {
  const { metrics } = useServerMetrics(server.id);
  const { recentLogs } = useRecentLogs(server.id, 10);

  return (
    <div className="grid grid-cols-1 lg:grid-cols-3 gap-6">
      {/* Server Info */}
      <div className="lg:col-span-2 space-y-6">
        {/* Basic Information */}
        <div>
          <h3 className="text-lg font-medium text-gray-900 mb-4">Server Information</h3>
          <div className="grid grid-cols-2 gap-4 text-sm">
            <div>
              <span className="text-gray-500">Type:</span>
              <span className="ml-2 font-medium capitalize">{server.type}</span>
            </div>
            <div>
              <span className="text-gray-500">Status:</span>
              <span className="ml-2 font-medium">{server.status}</span>
            </div>
            <div>
              <span className="text-gray-500">Created:</span>
              <span className="ml-2 font-medium">{formatDate(server.createdAt)}</span>
            </div>
            <div>
              <span className="text-gray-500">Last Updated:</span>
              <span className="ml-2 font-medium">{formatRelativeTime(server.updatedAt)}
</span>
```

```
      </div>
      {server.processId && (
        <div>
          <span className="text-gray-500">Process ID:</span>
          <span className="ml-2 font-medium">{server.processId}</span>
        </div>
      )}
      {server.containerId && (
        <div>
          <span className="text-gray-500">Container ID:</span>
          <span className="ml-2 font-medium font-mono text-xs">
            {server.containerId.substring(0, 12)}...
          </span>
        </div>
      )}
    </div>
  </div>

  {/* Performance Metrics */}
  {metrics && (
    <div>
      <h3 className="text-lg font-medium text-gray-900 mb-4">Performance Metrics</h3>
      <div className="grid grid-cols-2 md:grid-cols-4 gap-4">
        <MetricCard
          title="Uptime"
          value={formatUptime(metrics.uptime)}
          small
        />
        <MetricCard
          title="Avg Response"
          value={`${metrics.averageResponseTime}ms`}
          small
        />
        <MetricCard
          title="Health Checks"
          value={`${metrics.healthyChecks}/${metrics.totalChecks}`}
          small
        />
        <MetricCard
          title="Success Rate"
          value={`${Math.round(metrics.uptime)}%`}
          small
        />
      </div>
    </div>
  )}

  {/* Recent Logs Preview */}
  <div>
    <div className="flex items-center justify-between mb-4">
      <h3 className="text-lg font-medium text-gray-900">Recent Logs</h3>
      <Button
        variant="secondary"
        size="sm"
        onClick={() => setActiveTab('logs')}
      >
        View All Logs
      </Button>
    </div>
    <div className="bg-gray-900 rounded-lg p-4 max-h-64 overflow-y-auto">
      {recentLogs.length > 0 ? (
        <div className="space-y-1 font-mono text-sm">
          {recentLogs.map((log, index) => (
            <div key={index} className="flex space-x-2">
              <span className="text-gray-500 whitespace-nowrap">
                {new Date(log.timestamp).toLocaleTimeString()}
              </span>
              <span className={`font-bold ${getLogLevelColor(log.level)}`}>
                [{log.level.toUpperCase()}]
              </span>
              <span className="text-green-400 break-all">{log.message}</span>
            </div>
          ))}
        </div>
      ) : (
        <div className="text-gray-500 text-center py-8">
          No recent logs available
        </div>
      )}
    </div>
  </div>
</div>

{/* Sidebar */}
<div className="space-y-6">
  {/* Quick Actions */}
  <Card>
    <div className="p-4">
```

```
        <h4 className="font-medium text-gray-900 mb-4">Quick Actions</h4>
        <div className="space-y-2">
          <Button variant="primary" className="w-full" size="sm">
            <Download className="w-4 h-4 mr-2" />
            Export Logs
          </Button>
          <Button variant="secondary" className="w-full" size="sm">
            <Copy className="w-4 h-4 mr-2" />
            Copy Configuration
          </Button>
          <Button variant="secondary" className="w-full" size="sm">
            <RefreshCw className="w-4 h-4 mr-2" />
            Force Health Check
          </Button>
        </div>
      </div>
    </Card>

    {/* Configuration Summary */}
    <Card>
      <div className="p-4">
        <h4 className="font-medium text-gray-900 mb-4">Configuration</h4>
        <div className="space-y-2 text-sm">
          {server.type === 'container' && (
            <div>
              <span className="text-gray-500">Image:</span>
              <div className="font-mono text-xs mt-1 p-2 bg-gray-100 rounded">
                {server.image}
              </div>
            </div>
          )}
          {server.command && (
            <div>
              <span className="text-gray-500">Command:</span>
              <div className="font-mono text-xs mt-1 p-2 bg-gray-100 rounded">
                {server.command} {server.args?.join(' ')}
              </div>
            </div>
          )}
          {server.workingDirectory && (
            <div>
              <span className="text-gray-500">Working Directory:</span>
              <div className="font-mono text-xs mt-1 p-2 bg-gray-100 rounded">
                {server.workingDirectory}
```

```
                  </div>
                </div>
              )}
            </div>
          </div>
        </Card>
      </div>
    </div>
  );
};

// Health Tab Component
const HealthTab: React.FC<{ serverId: string }> = ({ serverId }) => {
  const [timeRange, setTimeRange] = useState('24h');
  const { healthHistory, healthSummary } = useHealthMetrics(serverId, timeRange);

  return (
    <div className="space-y-6">
      {/* Health Summary */}
      <div className="grid grid-cols-1 md:grid-cols-4 gap-4">
        <MetricCard
          title="Uptime"
          value={`${Math.round(healthSummary.uptime)}%`}
          icon={<TrendingUp className="w-6 h-6 text-green-500" />}
        />
        <MetricCard
          title="Total Checks"
          value={healthSummary.totalChecks}
          icon={<Activity className="w-6 h-6 text-blue-500" />}
        />
        <MetricCard
          title="Avg Response"
          value={`${healthSummary.averageResponseTime}ms`}
          icon={<Clock className="w-6 h-6 text-purple-500" />}
        />
        <MetricCard
          title="Last Check"
          value={healthSummary.lastCheck ?
            formatRelativeTime(healthSummary.lastCheck.timestamp) :
            'Never'
          }
          icon={<CheckCircle className="w-6 h-6 text-emerald-500" />}
        />
      </div>
      {/* Time Range Selector */}
      <div className="flex justify-between items-center">
        <h3 className="text-lg font-medium text-gray-900">Health History</h3>
        <Select
          value={timeRange}
          onChange={setTimeRange}
          options={[
            { value: '1h', label: 'Last Hour' },
            { value: '24h', label: 'Last 24 Hours' },
            { value: '7d', label: 'Last 7 Days' },
            { value: '30d', label: 'Last 30 Days' }
          ]}
        />
      </div>

      {/* Health Chart */}
      <Card>
        <div className="p-6">
          <HealthChart data={healthHistory} timeRange={timeRange} />
        </div>
      </Card>

      {/* Recent Health Checks */}
      <Card>
        <div className="p-6">
          <h4 className="font-medium text-gray-900 mb-4">Recent Health Checks</h4>
          <div className="space-y-3">
            {healthHistory.slice(0, 10).map((check, index) => (
              <div key={index} className="flex items-center justify-between py-2 border-b
last:border-b-0">
                <div className="flex items-center space-x-3">
                  <div className={`w-3 h-3 rounded-full ${
                    check.status === 'healthy' ? 'bg-green-500' : 'bg-red-500'
                  }`} />
                  <div>
                    <div className="text-sm font-medium">
                      {check.status === 'healthy' ? 'Healthy' : 'Unhealthy'}
                    </div>
                    <div className="text-xs text-gray-500">
                      {formatDate(check.timestamp)}
                    </div>
                  </div>
                </div>
                <div className="text-right text-sm">
                  {check.responseTime && (
                    <div className="font-medium">{check.responseTime}ms</div>
                  )}
                  {check.errorMessage && (
                    <div className="text-red-600 text-xs">{check.errorMessage}</div>
                  )}
                </div>
              </div>
            ))}
          </div>
        </div>
      </Card>
    </div>
  );
};
```

Acceptance Criteria:

- [ ] Server detail page loads with all information correctly
- [ ] Tabbed navigation works smoothly
- [ ] Overview tab shows comprehensive server information
- [ ] Health tab displays metrics and charts correctly
- [ ] Real-time data updates work across all tabs
- [ ] Configuration editing is functional and validated
- [ ] Actions panel works for all server operations

REQ-3.3: Marketplace and Templates

Priority: P1
Estimated Effort: 18 hours

Functional Requirements:

- Browse and search available MCP server templates
- Category-based filtering
- One-click deployment from templates
- Template configuration before deployment
- Popular and featured templates

Technical Implementation:

tsx

```
// Marketplace Component
const Marketplace: React.FC = () => {
  const [templates, setTemplates] = useState<Template[]>([]);
  const [loading, setLoading] = useState(true);
  const [searchQuery, setSearchQuery] = useState('');
  const [selectedCategory, setSelectedCategory] = useState('all');
  const [sortBy, setSortBy] = useState<'popular' | 'name' | 'recent'>('popular');

  useEffect(() => {
    loadTemplates();
  }, []);

  const loadTemplates = async () => {
    try {
      const response = await api.get('/templates');
      setTemplates(response.data.data);
    } catch (error) {
      console.error('Failed to load templates:', error);
    } finally {
      setLoading(false);
    }
  };

  const filteredTemplates = useMemo(() => {
    let filtered = templates;

    // Filter by search query
    if (searchQuery) {
      filtered = filtered.filter(template =>
        template.name.toLowerCase().includes(searchQuery.toLowerCase()) ||
        template.description.toLowerCase().includes(searchQuery.toLowerCase()) ||
        template.tags.some(tag => tag.toLowerCase().includes(searchQuery.toLowerCase()))
      );
    }

    // Filter by category
    if (selectedCategory !== 'all') {
      filtered = filtered.filter(template => template.category === selectedCategory);
    }

    // Sort templates
    filtered.sort((a, b) => {
      switch (sortBy) {
        case 'popular':
          return (b.downloads || 0) - (a.downloads || 0);
        case 'name':
          return a.name.localeCompare(b.name);
        case 'recent':
          return new Date(b.updatedAt).getTime() - new Date(a.updatedAt).getTime();
        default:
          return 0;
      }
    });

    return filtered;
  }, [templates, searchQuery, selectedCategory, sortBy]);

  const categories = useMemo(() => {
    const categorySet = new Set(templates.map(t => t.category));
    return Array.from(categorySet).sort();
  }, [templates]);

  return (
    <div className="space-y-6">
      {/* Header */}
      <div className="bg-white rounded-lg border border-gray-200 p-6">
        <div className="flex items-center justify-between">
          <div>
            <h1 className="text-2xl font-bold text-gray-900">MCP Server Marketplace</h1>
            <p className="text-gray-600 mt-1">
              Discover and deploy popular MCP servers with one click
            </p>
          </div>
          <Button variant="primary">
            <Plus className="w-4 h-4 mr-2" />
            Submit Template
          </Button>
        </div>
      </div>

      {/* Search and Filters */}
      <div className="bg-white rounded-lg border border-gray-200 p-6">
        <div className="grid grid-cols-1 md:grid-cols-4 gap-4">
          <div className="md:col-span-2">
            <SearchBar
              value={searchQuery}
              onChange={setSearchQuery}
              placeholder="Search templates..."
            />
          </div>
          <div>
            <Select
              value={selectedCategory}
              onChange={setSelectedCategory}
              options={[
                { value: 'all', label: 'All Categories' },
                ...categories.map(cat => ({ value: cat, label: cat }))
              ]}
            />
          </div>
          <div>
            <Select
              value={sortBy}
              onChange={setSortBy}
              options={[
                { value: 'popular', label: 'Most Popular'# MCP Manager - Phase 3 PRD: User Inte

**Timeline:** Weeks 9-12
**Goal:** Complete dashboard implementation, server detail views, real-time updates, and basic
**Success Criteria:** Production-ready UI with excellent user experience and marketplace functi

## Phase 3 Architecture Overview


┌───────────────────────────────────────────┐
│      Enhanced Frontend          │
├───────────────────────────────────────────┤
│ Dashboard   │ Server Details │ MP │
│ - Overview  │ - Config     │ -  │
│ - Stats   │ - Logs      │ Te │
│ - Actions   │ - Health     │ mp │
│       │ - Metrics     │ la │
│       │         │ te │
│       │         │ s  │
└───────────────────────────────────────────┘
│        │          │
│      WebSocket      REST API
│      Real-time     Requests
│      Updates          │
```

```
└──────────────────────────────────────┘
│          Enhanced Backend            │
│  - Template Engine                   │
│  - Marketplace API                   │
│  - Configuration Generator           │
│  - Real-time Event Broadcaster       │
└──────────────────────────────────────┘
```

```
## Enhanced Frontend Architecture

### Component Structure
```

```
src/
├── components/
│   ├── Layout/
│   │   ├── Header.tsx
│   │   ├── Sidebar.tsx
│   │   ├── Breadcrumbs.tsx
│   │   └── Layout.tsx
│   ├── Dashboard/
│   │   ├── Overview.tsx
│   │   ├── ServerGrid.tsx
│   │   ├── QuickActions.tsx
│   │   ├── SystemStats.tsx
│   │   └── RecentActivity.tsx
│   ├── ServerDetail/
│   │   ├── ServerInfo.tsx
│   │   ├── ConfigurationPanel.tsx
│   │   ├── LogsPanel.tsx
│   │   ├── HealthMetrics.tsx
│   │   ├── ActionsPanel.tsx
│   │   └── ServerDetail.tsx
│   ├── Marketplace/
│   │   ├── TemplateGrid.tsx
│   │   ├── TemplateCard.tsx
│   │   ├── CategoryFilter.tsx
│   │   ├── SearchBar.tsx
│   │   ├── DeployModal.tsx
│   │   └── Marketplace.tsx
│   ├── Forms/
│   │   ├── ServerForm.tsx
│   │   ├── ConfigurationForm.tsx
│   │   ├── EnvironmentEditor.tsx
│   │   └── ValidationDisplay.tsx
│   ├── Common/
│   │   ├── Button.tsx
│   │   ├── Input.tsx
│   │   ├── Select.tsx
│   │   ├── Modal.tsx
│   │   ├── Card.tsx
│   │   ├── StatusBadge.tsx
│   │   ├── LoadingSpinner.tsx
│   │   ├── ErrorBoundary.tsx
│   │   ├── Toast.tsx
│   │   └── ConfirmDialog.tsx
│   └── Charts/
│       ├── HealthChart.tsx
│       ├── UptimeChart.tsx
│       ├── ResourceChart.tsx
│       └── MetricsGrid.tsx
├── hooks/
│   ├── useApi.ts
│   ├── useServers.ts
│   ├── useWebSocket.ts
│   ├── useHealth.ts
│   ├── useLogs.ts
│   ├── useTemplates.ts
│   ├── useLocalStorage.ts
│   └── useToast.ts
├── services/
│   ├── api.ts
│   ├── websocket.ts
│   ├── templates.ts
│   ├── validation.ts
│   └── export.ts
```

```
├── context/
│   ├── AppContext.tsx
│   ├── ToastContext.tsx
│   └── ThemeContext.tsx
├── utils/
│   ├── formatting.ts
│   ├── validation.ts
│   ├── constants.ts
│   └── helpers.ts
└── types/
├── server.ts
├── template.ts
├── health.ts
├── logs.ts
└── ui.ts
```

~~~
## Core Requirements

### REQ-3.1: Enhanced Dashboard
**Priority:** P0
**Estimated Effort:** 16 hours

**Functional Requirements:**
- Real-time server overview with metrics
- System resource monitoring
- Recent activity feed
- Quick action shortcuts
- Responsive design for all devices

**Technical Implementation:**
```tsx
// Enhanced Dashboard Component
const Dashboard: React.FC = () => {
  const { servers, loading, error } = useServers();
  const { systemMetrics } = useSystemMetrics();
  const { recentActivity } = useActivity();

  return (
    <div className="space-y-6">
      {/* System Overview Cards */}
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
        <MetricCard
          title="Total Servers"
          value={servers.length}
          icon={<Server className="w-8 h-8 text-blue-500" />}
          trend={{ value: 12, isPositive: true }}
        />
        <MetricCard
          title="Running"
          value={servers.filter(s => s.status === 'running').length}
          icon={<CheckCircle className="w-8 h-8 text-green-500" />}
          trend={{ value: 5, isPositive: true }}
        />
        <MetricCard
          title="Healthy"
          value={servers.filter(s => s.healthStatus === 'healthy').length}
          icon={<Heart className="w-8 h-8 text-emerald-500" />}
          trend={{ value: 2, isPositive: false }}
~~~

```
    />
    <MetricCard
      title="Avg Response"
      value={`${systemMetrics.averageResponseTime}ms`}
      icon={<Activity className="w-8 h-8 text-purple-500" />}
      trend={{ value: 15, isPositive: false }}
    />
  </div>

  {/* Main Content Grid */}
  <div className="grid grid-cols-1 lg:grid-cols-3 gap-6">
    {/* Server Grid */}
    <div className="lg:col-span-2">
      <div className="bg-white rounded-xl border border-gray-200 overflow-hidden">
        <div className="p-6 border-b border-gray-200">
          <div className="flex items-center justify-between">
            <h2 className="text-lg font-semibold text-gray-900
```